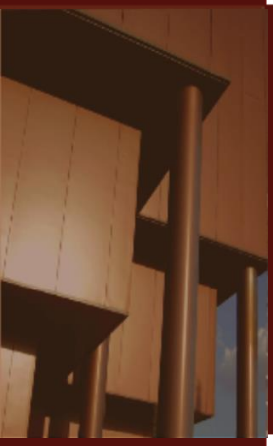
Trabajo Fin de Grado
Ingeniería de las Tecnologías Industriales

Métodos iterativos voraces para resolver la planificación integrada de consultas y quirófanos

Autora: Claudia Cañete Yaque

Tutores: Víctor Fernández-Viagas Escudero

José Manuel Molina Pariente



Dpto. de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Ingeniería de las Tecnologías Industriales

Métodos iterativos voraces para resolver la planificación integrada de consultas y quirófanos

Autora:

Claudia Cañete Yaque

Tutores:

Víctor Fernández-Viagas Escudero

José Manuel Molina Pariente

Dpto. de Organización Industrial y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Métodos iterativos voraces para resolver la planificación integrada de consultas y quirófanos

Autora: Claudia Cañete Yaque

Tutores: Víctor Fernández-Viagas Escudero
José Manuel Molina Pariente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

AGRADECIMIENTOS

Agradecer en primer lugar a mis tutores, Víctor y José Manuel, por sus aportaciones y por guiarme en todo momento en la realización de este trabajo de la manera en la que lo han hecho, dedicándome todo el tiempo que he necesitado siempre. Agradecer también a esta Escuela por la gran formación recibida a lo largo de estos cuatro años de carrera.

En segundo lugar, agradecer a mi familia, sin la que este recorrido no hubiera sido posible. A mis padres, Jacinto y Raquel, por asfaltar mi camino y darme siempre todas las facilidades. A mis hermanas, Raquel y Laura, por apoyarme. A Javier por acompañarme.

RESUMEN

El objetivo de este trabajo es la resolución de un problema real de secuenciación que plantea el Hospital Universitario Puerta del Mar en Cádiz. Se trata de la creación de un programa quirúrgico y de consultas del hospital mediante el cual se debe obtener la asignación de cirujanos a consultas y pacientes en las distintas etapas sanitarias para un horizonte temporal de una semana. Este objetivo deberá cumplir con las restricciones que indica el propio hospital y que han de tenerse en cuenta a la hora de realizar el programa. Así, el fin de este trabajo será la optimización de la planificación, minimizando el número de pacientes atendidos con retraso, así como equilibrar la carga de trabajo de los distintos cirujanos en quirófanos y consultas maximizando la utilización de los recursos disponibles. Se desarrollará una heurística compuesta por algoritmos aproximados en lenguaje Python y su interfaz en un Libro Excel.

La gestión de las listas de espera es un problema acuciante en los hospitales españoles. Una eficiente programación de los servicios ofrecidos en los hospitales será un factor esencial para optimizar el funcionamiento de los mismos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Programación de la Producción

La programación de la producción se define como "la asignación de recursos para la fabricación de un conjunto de productos" (Framinan et al., 2014). Esta asignación da como resultado un programa de producción. El programa establece qué, cuánto y cuándo se llevará a cabo esta producción, incluyendo fechas de inicio y finalización del proceso de cada producto en cada recurso, la cantidad a producir, dentro de un horizonte temporal establecido.

El objetivo principal de la programación de la producción es el de organizar la actividad de producción de una empresa de forma eficiente asignando recursos limitados a actividades que deben ser realizadas para cumplir las metas, ya sea en una empresa manufacturera o una empresa de servicios. Un óptimo plan de producción puede suponer una ventaja competitiva sustancial sobre empresas del sector. Estas empresas realizarán su plan de producción buscando la mejora de distintos factores, optimizando diversas funciones objetivo. Las funciones objetivo se clasifican en las siguientes categorías mostradas en la ***Figura 1-1***: coste, tiempo, calidad, flexibilidad (Framinan et al., 2014). Las empresas pueden buscar distintos objetivos como reducir costes, maximizar la producción, reducir inventarios, etc.

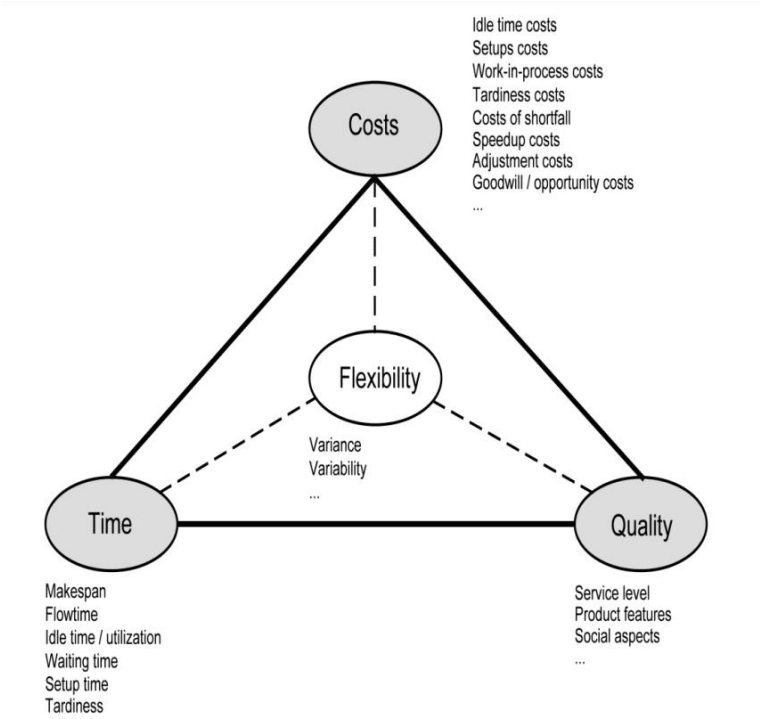


Figura 1-1: *Categorización de las funciones objetivo* (Framinan et al., 2014)

Para el desarrollo del programa, han de tenerse en cuenta una serie de restricciones que presenta el problema a resolver, pudiendo ser restricciones de tiempo, de recursos, o económicas. A su vez, cada producto puede tener cualidades distintas: prioridad, tiempo de procesado, fecha de comienzo, fecha límite, etc. Además, los recursos, que pueden ser personas del equipo o máquinas, también presentan características concretas, como velocidad de procesado, disponibilidad, coste de procesado, especialización, etc.

De esta manera, para definir un problema de programación de operaciones se requiere concretar los

siguientes factores (González de Diego & Araúzo Araúzo, 2015):

- Los recursos: son las personas, máquinas, centros de trabajo o instalaciones que permiten la realización de las operaciones de proceso sobre los trabajos
- Los trabajos: aquellas operaciones que necesitan de los recursos para ser procesados en cierto periodo de tiempo
- Las restricciones o condiciones a satisfacer: Existen restricciones referentes a los recursos (disponibilidad, capacidad, configuración), a las operaciones (restricciones de sucesión, es decir, una operación debe procesarse antes que otra, o temporales, las cuales definen la fecha de finalización de las operaciones)
- La función objetivo: función cuyo valor se pretende minimizar o maximizar.

Habiendo definido el problema de programación, se procederá a desarrollar algoritmos y/o modelos matemáticos con el fin de resolver el problema y encontrar una solución factible. Este proceso se introducirá en el siguiente apartado, y la solución aportada al problema propuesto en este trabajo se desarrollará a lo largo de este documento.

Una vez desarrollada la programación, se obtendrá como resultado un programa de producción, que puede ser representado en un Diagrama de Gantt como el que se muestra en la ***Figura 1-2*** (Framinan et al., 2014), en el que detalla comienzo y finalización de cada producto en cada máquina. En el caso de la figura, sería el programa de producción para un modelo Flow Shop: cada trabajo de la secuencia debe procesarse en las máquinas siguiendo la misma ruta.

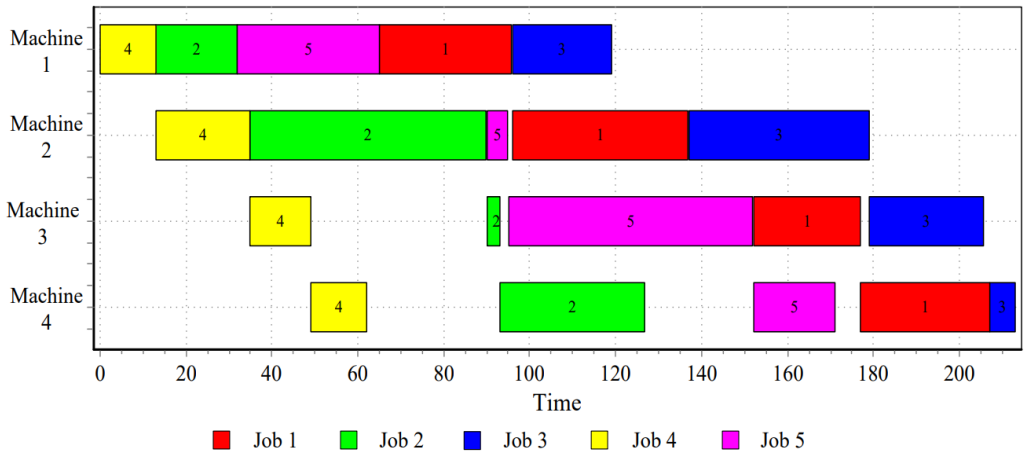


Figura 1-2*: Diagrama de Gantt para la programación de un Flowshop con cinco trabajos* (Framinan et al., 2014)

En este trabajo se tratará la programación de la producción de una empresa de servicios como es un hospital. Se desarrollará un programa para la planificación del Hospital Puerta del Mar, en Cádiz, cuyo fin es la reducción al máximo de los retrasos, respecto a unos tiempos máximo de respuesta establecidos por decretos ley de la Consejería de Salud, en la atención de los pacientes de las listas de espera, así como equilibrar las cargas de trabajo de los cirujanos maximizando la utilización de los recursos disponibles.

1.2 Programación mediante algoritmos y metaheurísticas

Los problemas de programación de operaciones son típicamente resueltos mediante modelos exactos o mediante algoritmos de optimización. Este trabajo está enfocado a la resolución mediante algoritmos.

Un algoritmo es un "conjunto de reglas que, aplicada sistemáticamente a unos datos de entrada apropiados, resuelven un problema en un numero finito de pasos elementales" (Marí, 2006). Los algoritmos se suelen clasificar en algoritmos exactos y algoritmos aproximados. Los algoritmos exactos son aquellos que garantizan una solución óptima, la mejor posible para el problema dado. Se usan para resolver problemas polinomiales. Por otro lado, los algoritmos aproximados son aquellos que no garantizan la solución óptima, pero sí obtienen una solución buena, un óptimo local. Son usados para resolver problemas complejos no polinomiales.

Como se ha comentado antes, un problema puede ser clasificado en polinomial (P) y no polinomial (NP). Algunos problemas pueden ser modelados como problemas lineales y por ello resueltos de forma rápida a través de algoritmos en un tiempo polinomial. Esto significa que instancias de este problema, por grandes que sean, pueden igualmente ser resueltas en un tiempo computacional relativamente corto. Sería entonces un problema P (Pinedo, 2005).

Por otro lado, los problemas NP son problemas complejos, los cuales no se pueden resolver con métodos exactos en un tiempo computacional razonable, ya que se tardaría años en explorar todos los puntos de su región admisible con la velocidad de procesamiento de los ordenadores actuales. El número de posibles soluciones aumenta exponencialmente con el tamaño del problema. Los métodos de resolución tradicionales como la programación lineal, programación entera, o mixta no son eficientes en términos de tiempo computacional. Por ello, para problemas NP, en general, se usan métodos heurísticos que, aunque no garanticen optimalidad, encuentran soluciones buenas. Estos pueden ser metaheurísticas, heurísticas constructivas, o heurísticas de mejora. Las heurísticas son procedimientos que tratan de aportar soluciones a un problema específico con buen rendimiento, en lo que a calidad de las soluciones y los recursos empleados se refiere (Melián et al., 2003).

Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas generales difíciles de optimización combinatoria, en los que los heurísticos clásicos no son ni efectivos ni eficientes. Estos métodos se basan en conceptos derivados de la inteligencia artificial, la ciencia biológica, las matemáticas, las ciencias naturales y físicas para incrementar el rendimiento de las heurísticas (Osman & Kelly, 1996).

Una metaheurística incorpora varias heurísticas para explorar una mayor parte de la región admisible. A continuación, en la ***Figura 1-3*** se muestra la estructura general de una metaheurística (Canca Ortiz & González Rodríguez, 2015). En primer lugar, se realiza la codificación y se obtiene una solución inicial. En segundo lugar, teniendo en cuenta las restricciones, se evalúa dicha solución, se continúa explorando el espacio de la región admisible, y finalmente cuando se cumple el criterio de parada se obtiene la solución aportada por dicha metaheurística.

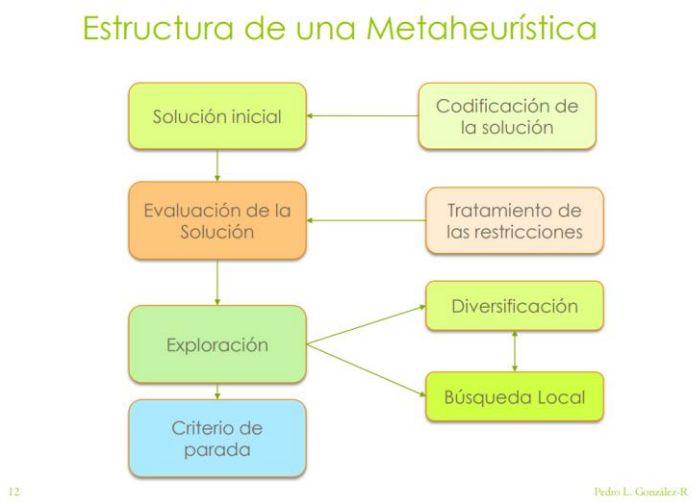


***Figura 1-3**: Estructura de una Metaheurística* (Canca Ortiz & González Rodríguez, 2015).

En este proyecto se desarrollará una metaheurística para resolver el problema NP de secuenciación que aporte soluciones lo más cercanas a la óptima posible. En esta metaheurística específica para este problema se aplica el algoritmo Iterative Greedy (Ruiz & Stützle, 2007).

1.2.1 Iterative Greedy

En este proyecto se va a integrar el algoritmo Iterative Greedy en las metaheurísticas que se han planteado para resolver el problema. Se trata de un algoritmo que genera unas secuencias de trabajos iterando en tres fases: inicialización, destrucción y construcción. Tras estas fases, se procede a realizar una búsqueda local opcional.

Para la inicialización se obtiene la primera secuencia a evaluar mediante la heurística NEH. Esta heurística da como resultado una secuencia que, inicialmente vacía, va insertando uno a uno todos los trabajos en todas las posiciones posibles. Para cada trabajo insertado, se guarda la que mejor resultado de la función objetivo consiga, y sobre esta secuencia se itera con el siguiente trabajo.

A continuación, en la fase de destrucción, un número aleatorio de componentes de posiciones también aleatorias son eliminados de la secuencia. En la fase de construcción se aplica la heurística constructiva para completar la secuencia. Se van insertando, en el mismo orden en el que fueron eliminados, los componentes que faltan de la secuencia. Una vez se han introducido de nuevo todos los componentes de la secuencia, un criterio de aceptación determina si la nueva solución sustituirá a la inicial, en función de si la función objetivo mejora con ella.

Por último, se genera una búsqueda local. Esta consiste en insertar, uno a uno, cada elemento de la secuencia en todas las posiciones. Se evalúa cada nueva secuencia y en caso de mejorar la mejor función objetivo obtenida hasta este punto, ésta secuencia será denominada como la mejor hasta y con la que será comparada el resto. La destrucción, construcción y búsqueda local se realizan iterativamente hasta que un criterio de parada anteriormente establecido es cumplido. Este criterio de parada puede ser un número fijo de iteraciones, un margen de diferencia mínimo entre secuencias, cierto número de iteraciones cuando ya no mejora la función objetivo, etc. En la ***Figura 1-4*** se presenta la estructura general del algoritmo Iterative Greedy.(Ruiz & Stützle, 2007)

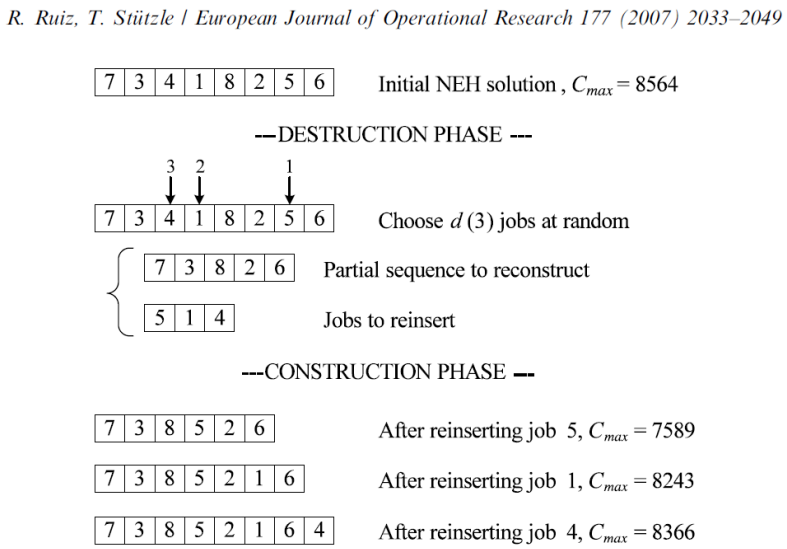


Figura 1-4: *Ejemplo de aplicación del algoritmo Iterative Greedy de fases inicialización, destrucción y construcción* (Ruiz & Stützle, 2007)

1.3 Actualidad en entornos sanitarios

La programación de operaciones que se lleva a cabo en este proyecto está enfocada a entornos sanitarios. A lo largo de la última década, debido a factores como el aumento de la esperanza de vida, la aparición de nuevas enfermedades o restricciones de presupuesto, los sistemas sanitarios han sufrido un aumento en la presión por mantener una buena calidad de atención al paciente con el menor coste posible (Fei et al., 2010). En este contexto, técnicas de programación de la producción se aplican para incrementar la eficiencia del sector sanitario, en particular la programación de las cirugías en quirófanos, por su elevado coste.

Uno de los principales problemas a los que se enfrenta continuamente el sistema sanitario público español es la gran lista de espera por el que todos los pacientes deben pasar antes de ser atendidos. Esto genera una ineficiencia para los hospitales, que ven que no pueden satisfacer las demandas de sus pacientes a tiempo, lo que puede conllevar un gran riesgo para aquellos que se encuentren en situaciones más críticas y cuya salud dependa en gran medida de la rapidez con la que se les trate (González-Busto & García, 1999). En Andalucía, a junio de 2021, había un total de 843.945 personas en listas de espera, de los cuales 21.000 pacientes estaban soportando una espera mayor a los plazos garantizados.(Informe CCOO Listas de Espera 2021, n.d.)

A continuación, se puede observar en la ***Tabla 1-1*** la lista de espera quirúrgica del Hospital Universitario Puerta del Mar en Cádiz a diciembre del 2021. Es apreciable la demora media en días que sufren las intervenciones de los grupos más solicitados, así como el gran número de pacientes que son atendidos fuera de plazo en grupos como 'Otros procedimientos terapéuticos con uso de quirófano sobre piel y mama', 'Fusión espinal', o 'Artroplastia de rodilla'.

Tabla 1-1: *Lista de Espera Quirúrgica del Hospital Universitario Puerta del Mar, Cádiz. Diciembre 2021.*(*Hospital Universitario Puerta Del Mar | Servicio Andaluz de Salud*, 2021)

Pacientes inscritos en lista de espera quirúrgica por grupo de procedimiento. Diciembre 2021				
Orden	**Grupo de procedimientos**	**Total pacientes**	**Fuera de plazo + >365 días**	**Demora media (días)**
1	Otros procedimientos terapéuticos con uso de quirófano sobre piel y mama	605	270	404
2	Procedimientos sobre dientes	486	1	59
3	Procedimientos sobre cristalino y cataratas	386	3	50
4	Injerto de piel	328	31	96
5	Otros procedimientos terapéuticos sobre párpados, conjuntiva y córnea	311	38	184
6	Fusión espinal	298	110	152
7	Otros procedimientos terapéuticos sobre músculos y tendones	210	19	175
8	Reparación de hernia inguinal y femoral	189	22	93
9	Artroplastia de rodilla	183	52	158
10	Eliminación de venas varicosa de miembro inferior	171	0	59
11	Laminectomía. Escisión del disco intervertebral	162	42	121
12	Bunionectomía o reparación de deformidades de los dedos de los pies	156	18	121
13	Otros procedimientos terapéuticos con uso de quirófano sobre genitales masculinos	154	48	170
14	Circuncisión	143	24	96
15	Otra reparación de hernia	138	14	111
16	Otros procedimientos terapéuticos con uso de quirófano sobre nariz, boca y faringe	133	13	137
17	Otros procedimientos terapéuticos sobre genitales masculinos (sin uso de quirófano en EEUU)	130	2	107
18	Escisión de lesión de piel	127	29	181
19	Descompresión de nervio periférico	120	18	100

20	Colecistectomía y exploración del conducto común	106	10	80
21	Resto de Procedimientos	1.441	261	157
Total Hospital Universitario Puerta del Mar		**5.977**	**1.025**	**153**

Las decisiones en relación con la planificación, no solo en el sector sanitario sino en cualquier empresa, se suelen dividir en tres niveles jerárquicos donde las decisiones se toman sucesivamente: estratégico, táctico y operativo.

En el nivel estratégico, se toman decisiones en relación con la capacidad y la distribución de recursos. Las variables de decisión son el número y el tipo de cirugías que se van a planificar en el horizonte temporal, distribución del hospital, número de pacientes que se pueden abarcar, así como el número de recursos necesarios (Molina-Pariente, Fernandez-Viagas, et al., 2015). Todas ellas, decisiones que afectarán un largo periodo de tiempo.

Las decisiones del nivel táctico y operativo difieren en el horizonte temporal para el que se plantean. Las decisiones a nivel táctico son decisiones a medio plazo, mientras que las decisiones a nivel operativo son a corto plazo. En el nivel táctico, los recursos se asignan a espacios de tiempo dentro del horizonte temporal, estableciendo el calendario quirúrgico. Finalmente, se decide la fecha, el número de quirófano, y la hora para cada cirugía, que pertenecen al nivel operativo(Molina-Pariente, Fernandez-Viagas, et al., 2015). En este nivel, del que es objeto este proyecto, se toman decisiones en intervalos de tiempo muy cortos dado el constante cambio de circunstancias.

Existen dos tipos de políticas para la programación sanitaria (Fei et al., 2008):

- En primer lugar, *block-scheduling policy*. Cada cirujano es asignado previamente a un quirófano o consulta, donde atenderá a sus pacientes. Este cirujano no podrá realizar consultas o cirugías fuera de este espacio que le ha sido asignado. Cada cirujano tiene su propio 'bloque', no compartido con otros cirujanos. A modo de ejemplo, en la consulta donde sea asignado el cirujano, no podrán pasar consulta otros cirujanos.
- En segundo lugar, *open scheduling*, en el que se asigna a cada cirujano un quirófano o consulta en función de dónde se les requiera en cada momento. Los cirujanos no tienen bloque propio, sino que pueden atender a pacientes en distintas consultas y quirófanos según se les haya asignado pacientes.

En este proyecto se tratará la política *block-scheduling* para las consultas por ser con la que se rige el Hospital Universitario Puerta del Mar de Cádiz para el cual se realiza la planificación. En cada consulta sólo podrá atender un cirujano. Sin embargo, los quirófanos se rigen por la política *open-scheduling*, ya que en cada quirófano podrá operar más de un grupo quirúrgico al día.

2 DESCRIPCIÓN DEL PROBLEMA

Con este estudio se plantea una solución a un problema real actual del Hospital Universitario Puerta del Mar de Cádiz. Este problema consiste en generar la planificación semanal a partir de una lista de espera de pacientes. Se trata de obtener tanto una asignación de cirujanos a consultas y quirófanos del hospital como una secuencia de pacientes que indique el orden en el que se asignan para un ciclo sanitario compuesto de tres etapas. Los objetivos son minimizar el número de pacientes de la lista de espera atendidos con retraso, a la vez que se equilibra la carga de trabajo de los cirujanos y se maximiza la utilización que se hace de los recursos disponibles. Todo ello en base a un número de restricciones que presenta el hospital. Se trata por tanto de un problema de optimización multiobjetivo resuelto con algoritmos aproximados.

2.1 Funciones objetivo

Uno de los objetivos principales es eliminar los retrasos en la atención al paciente en la lista de espera. En muchas ocasiones la condición del paciente depende del tiempo en el que es tratado. Por tanto, intentar que ese paciente sea atendido cuanto antes puede suponer una diferencia sustancial para aquellos en riesgo. En nuestro problema, además, se tiene en cuenta la gravedad de cada paciente. Esto es, la función objetivo empeora si el paciente que va tarde tiene mayor peso en la lista de espera. Esto se consigue estableciendo que la función objetivo sea igual al producto de los días de demora, tiempo acumulado – tiempo de respuesta, por el peso del paciente, w, siguiendo la ecuación 1. En la ecuación se modela como el máximo entre 0 y los días de diferencia para así, en aquellos casos en los que no haya retraso, se tome como 0 los días de demora. En algunas ocasiones, será mejor solución aquella en la que dos pacientes leves vayan tarde que un solo paciente grave con retraso. Con ello se intenta atender siempre antes a los pacientes con mayor gravedad cuyas vidas corren peligro y cuyo estado puede empeorar sustancialmente en cuestión de días. A esta función objetivo de minimización se la denominará 'Tardiness' a lo largo de este documento.

$$FO_1 = \sum_{todos\ los\ pacientes} w \cdot \mathrm{Max}\,[0, (TiempoAcumulado - TiempodeRespuesta)] \qquad (1)$$

Por otro lado, el programa intenta equilibrar el número de horas de consulta y quirófanos acumuladas por cada cirujano para igualar a carga de trabajo de cada uno de ellos. Con ello se pretende que no haya ningún cirujano que acumule una cantidad de horas de consulta mientras que otros cirujanos apenas han atendido consultas durante la semana. Para ello, se crean dos variables que guarden todas las horas que lleva acumuladas cada cirujano en consultas y en quirófano, hcc y hcor respectivamente. La función objetivo será la suma de las varianzas de las horas en quirófano y la varianza de las horas en consulta. Se intenta que la varianza de los valores del vector sea la mínima posible. Esta función objetivo de minimización se denominará 'Equilibrio' a lo largo de este documento

$$FO_2 = \sum var(horas\ acumuladas\ consulta) + var(horas\ acumuladas\ en\ quirófano) \qquad (2)$$

Por último, se añade una tercera función objetivo con el fin de maximizar la utilización de los recursos disponibles. Un recurso disponible inutilizado supone un coste muy alto para el hospital, y además implica el dejar de atender a pacientes que, aunque no cumplan el tiempo de respuesta aún y puedan esperar para ser atendidos, podrían beneficiarse de una atención temprana. De esta forma, se busca conseguir atender al mayor número de pacientes posible dentro del horizonte temporal utilizando la máxima cantidad de horas disponibles de consultas y quirófanos. La función objetivo se expresa como la suma de las horas totales aprovechadas de

consultas y quirófanos.

$$FO_3 = \sum horas\ ocupada\ cada\ consulta + \sum horas\ ocupado\ cada\ quirófano \quad (3)$$

Por simplificación del problema, se ha decidido linealizar y unir las tres funciones objetivo en una sola. Para linealizar, cada función objetivo deberá tener un valor entre 0 y 1. Se divide el valor de cada función objetivo entre el valor más alejado al óptimo posible para así no obtener nunca un valor mayor que 1.

La función objetivo Tardiness habría que dividirla entre el máximo tardiness que podrá haber dada cualquier lista de espera de pacientes. Sería el múltiplo del máximo peso, el máximo número de días de demora y el máximo número de pacientes tarde. El máximo peso de los pacientes en la lista de espera es de 15. El máximo número de días de retraso es de 337 teniendo en cuenta que el mínimo tiempo de respuesta es de cuatro semanas y el tiempo acumulado en la lista de espera pueda llegar a ser hasta de un año. El máximo número de pacientes que van tarde será la totalidad de la lista de espera, por ello, se multiplica por la longitud de ésta. Así, en la función objetivo total, la función objetivo Tardiness quedaría:

$$FO_1{}' = \frac{FO_1}{15 * 337 * len(pacientes)} \quad (4)$$

La función equilibrio se divide entre la varianza máxima, que será cuando la mitad de los cirujanos acumulen todas las horas posibles. Esta será una varianza de 16 para los quirófanos, el cuadrado del promedio de horas acumuladas por cirujano, 4, teniendo en cuenta que los quirófanos funcionan ocho horas diarias. Sin embargo, la varianza máxima de horas acumuladas en consulta es de 9, ya que las consultas sólo están disponibles seis horas diarias. Como la función objetivo es la suma de la varianza en las consultas y la varianza en los quirófanos, la suma de éstas dos en su máximo será 25. Por tanto, se concluye que el valor máximo que podrá alcanzar la función objetivo de equilibrio es de 25.

$$FO_2{}' = \frac{FO_2}{25} \quad (5)$$

La función objetivo de maximizar la utilización de los recursos se dividirá entre el total de horas disponibles a la semana. Además, se multiplica por -1 para convertirla en una función de minimización, ya que originalmente es una función que trata de maximizar la utilización de los recursos atendiendo al mayor número de pacientes posible.

$$FO_3' = -\frac{FO_3}{8h * 7dias * número\ de\ quirófanos + 6h * 7dias * número\ de\ consultas} \quad (6)$$

Finalmente, la función objetivo global linealizada sería la suma de las funciones objetivo anteriores, cada una de ellas multiplicada por su respectivo peso.

$$FO_{global} = w_1 * \frac{FO_1}{15 * 337 * len(pacientes)} + w_2 * \frac{FO_2}{25} - w_3 * \frac{FO_3}{8h * 7dias * número\ de\ quirófanos + 6h * 7dias * número\ de\ consultas} \quad (7)$$

En los experimentos realizados todos los pesos son el mismo, teniendo en cuenta que, por obtener mayor valor en la mayoría de problemas, la utilización cobraría más importancia. A la hora de aplicar la

planificación desarrollada a un hospital real, cada unidad quirúrgica tendrá distintas prioridades y por tanto cada una de ellas aplicaría un peso distinto a cada función objetivo. Aquellas que prioricen equilibrar las cargas de trabajo de los cirujanos, darán un mayor peso a dicha función objetivo. En los casos en los que se priorice minimizar los pacientes que son atendidos con retraso darán mayor peso a la función objetivo Tardiness. Igual pasaría con la función objetivo de Utilización. Como en este trabajo se tratan todas las unidades de manera conjunta, se ha decidido aplicar a todas las funciones objetivo el mismo peso de 1.

2.2 Recursos

Los recursos limitados que conforman este problema son por un lado las consultas y quirófanos, y por otro lado los cirujanos. Estos recursos han de asignarse de forma eficiente para optimizar las funciones objetivo del problema. El número de consultas y quirófanos depende de cuántos estén disponibles ese día, que se deberá indicar en la interfaz del usuario, es un dato del problema. Con respecto al horario, para este problema estarán disponibles 8 horas al día los quirófanos y 6 horas al día las consultas. Se atiende menos horas en consulta para no saturar excesivamente los quirófanos ya que las duraciones de los pacientes en consulta son mucho menores que el tiempo que duran sus intervenciones. Disminuyendo el tiempo en consultas permite compensar de cierta manera los pacientes en lista de espera en la etapa de cirugía. Por otro lado, cada cirujano tiene unas características concretas:

- Cada uno pertenece a una unidad quirúrgica concreta, que en este problema estarán numeradas del 0 al 7, pero representan unidades como dermatología, neurología, pediatría, etc.
- Cada paciente de la lista de espera se encuentra en una etapa del ciclo sanitario, explicadas en el siguiente punto, que indican si esperan una cita para consulta o para quirófano.
- Cada cirujano tiene un nivel de experiencia en el quirófano, basado en los años de trabajados en la materia. En este problema el nivel de experiencia estará definido en un rango del 1 al 20, siendo 20 la máxima que puede llegar a tener. Esta experiencia definirá qué cirujanos son aptos para llevar a cabo la cirugía de los pacientes, que deberán cumplir con el mínimo nivel de experiencia que requiera dicha intervención.
- Además, tienen ciertos días de descanso a la semana definidos. En estos días no podrán atender a ningún paciente. En este problema se han definido los días de descanso de manera aleatoria para cirujano, determinando que cada uno de ellos trabajará un 70% de la semana.

2.3 Trabajos

En este problema los trabajos a procesar son los pacientes que se encuentran en la lista de espera. Cada uno de ellos se encontrará en una etapa, y el método debe asignarlo en la que le corresponda. Todos los pacientes deben pasar por una consulta PAE (Primera Atención Especializada), en segundo lugar, la intervención quirúrgica, y por último una consulta postoperatoria. Además, cada paciente tiene ciertos atributos asignados. Estos son:

- Duración en cada etapa (será la misma para todas las consultas PAE y las consultas postoperatorias)
- Tiempo de respuesta límite para cada etapa. Es decir, el tiempo máximo que debe pasar entre que termina la anterior etapa y lo asignan en la siguiente. Si excede dicho tiempo límite, el paciente será considerado atendido con retraso.
- Unidad a la que pertenece. El cirujano que lo atienda tanto en consultas como en la intervención deberá pertenecer a la misma unidad. Esto es, si el paciente requiere atención en traumatología, el cirujano que lo atienda deberá ser traumatólogo.

- Prioridad con respecto al resto de pacientes, es decir, su peso dentro de la lista de espera. Los pacientes más urgentes tendrán un peso mayor que aquellos que puedan esperar más a ser atendidos. La urgencia definirá el tiempo de respuesta para la cirugía. En este problema, el peso se mide en un rango del 1 al 15. Los pacientes con un peso de 15 serán los más urgentes, y los que tengan un peso de 1 los menos graves.
- Grado de experiencia que requiere para la operación. Cada paciente requerirá un nivel de experiencia para su cirugía. Aquellas con gran complejidad requerirán cirujanos con una experiencia alta, y no podrán operarlos los cirujanos que no cumplan con dicho nivel de experiencia. Además, el cirujano asistente también debe cumplir con el nivel de experiencia requerido para el cirujano asistente. Habrá sin embargo pacientes que por distintos motivos no necesiten cirujano asistente en su operación, siendo identificado con un nivel de experiencia igual a 0.

2.4 Restricciones

El problema cuenta con una serie de restricciones que limitan las soluciones y han de tenerse en cuenta a la hora de realizar la planificación.

El proceso por el que debe pasar cada paciente, como se ha comentado anteriormente, consta de tres etapas:

1. En primer lugar, una consulta PAE, primera atención especializada, en la que son atendidos por un cirujano correspondiente a su unidad.
2. A continuación, el paciente pasará por cirugía. En ella un equipo de dos cirujanos, uno principal y otro asistente, realizará la operación.
3. Finalmente, el paciente es asignado a una consulta postoperatoria en la cual es atendido por el mismo cirujano que le operó.

A pesar de haber tres etapas, sólo habrá una única lista de espera desde la cual se irán asignando a la etapa que corresponda. Es un enfoque distinto a lo que se ha hecho hasta ahora en la literatura, donde normalmente existe una lista de espera para cada etapa, resolviéndose los problemas de manera separada.

No existen para este problema restricciones de precedencia o sucesión, ni tampoco restricciones estrictas de localización temporal. Cada paciente puede ser atendido antes o después de otros aleatoriamente, y el propio método se quedará con la secuencia de pacientes que optimice la función objetivo. De igual manera, un paciente tiene asignado un tiempo de respuesta para ser atendido, pero por el hecho de no cumplirlo no significa que no sea admisible dicha solución. Simplemente, la función objetivo será peor que para otra secuencia de pacientes en la que sí se cumplan todos los tiempos de respuesta. El tiempo de respuesta para la cirugía se mide desde que el paciente termina su consulta PAE. El tiempo de respuesta para la consulta postoperatoria se mide desde que el paciente es operado. Cuando el paciente excede estos tiempos límites, se tomará que el paciente va tarde.

Por otro lado, sí existen restricciones referentes a las consultas, quirófanos y cirujanos de la plantilla. Como se indicó en el apartado de recursos, el número de consultas y quirófanos disponibles deben indicarse como dato. Esto influirá en la capacidad que tiene el hospital para atender pacientes durante el día. En cada quirófano se pueden operar todo tipo de intervenciones independientemente de la unidad a la que pertenezcan, no hay quirófanos especializados para ningún tipo de cirugía en concreto. Además, pueden entrar más de un equipo quirúrgico al día. Cada equipo estará constituido por un cirujano principal y un asistente. El principal debe pertenecer a la unidad del paciente en cuestión y además cumplir con el nivel de experiencia requerido para la operación. El cirujano asistente, sin embargo, no tiene por qué pertenecer a la misma unidad, pero sí debe cumplir con el nivel de experiencia requerido. Existen intervenciones que no requieren de un cirujano asistente y el cirujano principal será el único encargado de levarla a cabo.

Con respecto a las restricciones de las consultas, en cada una de ellas sólo podrá atender un cirujano al

día, siguiendo la política *block-scheduling*. En los turnos de consulta, se podrán atender tanto consultas PAE como post operatorias. Estas consultas post operatorias deben ser realizadas por el mismo cirujano que llevó a cabo la operación del paciente.

Además, un cirujano podrá ser asignado un día a cirugías o a consultas, pero no podrá estar en ambos el mismo día, ni tampoco podrá estar en dos de ellas el mismo día. Es decir, si el cirujano es asignado a consultas el lunes, sólo podrá ocupar una de ellas, no podrá estar asignado a las consultas 2 y 3 el mismo día, ni tampoco podrá ser asignado a cirugías. Igual ocurre con los quirófanos.

3 MÉTODOS PROPUESTOS

Este trabajo consiste en el desarrollo de un programa que resuelva el problema explicado en el capítulo anterior. Partiendo de los datos de entrada de la lista de espera y la plantilla de cirujanos, el programa debe devolver una planificación semanal, qué pacientes están asignados a las distintas consultas y quirófanos, y qué cirujanos son los encargados de atenderles. Además, han de cumplirse todas las restricciones optimizando la función objetivo.

El programa se ha desarrollado en lenguaje Python, un lenguaje de código abierto creado en la década de los noventa. Con el paso del tiempo, Python ha ido ganando usuarios debido a su sencillez y a sus amplias posibilidades en entornos de inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge (Blog Becas Santander, 2021).

Se han desarrollado en total seis metaheurísticas diferentes, IG0, IG0-LS, IG1, IG1-LS, IG2, IG2-LS. De estas seis se analizará cuál de ellas presenta un mejor funcionamiento para el problema de planificación y esa metaheurística será la que se aporte como solución final.

Para evaluar cuál de ellas obtiene mejores resultados y funciona mejor para el problema propuesto, se genera una batería de instancias y se analiza la validez de los resultados devueltos por cada algoritmo para un horizonte temporal de un día, un corto plazo. Esto se expondrá en el capítulo 4. Una vez evaluada, aquella metaheurística que aporte mejores resultados será la elegida para llevar a cabo una simulación en el horizonte continuo de una semana para una instancia de las generadas anteriormente. Se analizará a continuación la eficiencia del programa y la validez de la planificación. Se expondrán los resultados en el capítulo 5.

Se explicará el código desgranando desde las funciones más simples de decodificación, en el apartado 3.1, hasta finalmente acabar con la metaheurística completa planteada en el apartado siguiente 3.2.

3.1 Decodificación de la solución

A continuación, se procede a realizar un desglose de la función solución, presente en las seis heurísticas.

En primer lugar, se explican las funciones más importantes del código mediante las cuales se obtiene la función objetivo y se genera la planificación. Existen dos funciones, DS1 y DS2. Se diferencian en los datos que reciben como entrada. DS1, mostrado en la ***Figura 3-1***, recibe como entrada únicamente la secuencia de pacientes. A partir de esta secuencia, se van asignando pacientes en ese orden y a la vez asigna los cirujanos a sus turnos en consulta o quirófano según se les necesite. Se va recorriendo el vector secuencia de pacientes, y en ese orden, uno por uno, va asignándolo a una consulta o quirófano buscándole un cirujano disponible.

Por otro lado, DS2, mostrada en la ***Figura 3-2***, recibe como entrada tanto la secuencia de pacientes como la asignación de cirujanos a las consultas. Estos cirujanos deberán ceñirse a dicha asignación y no podrán atender a pacientes fuera de esos turnos. En este caso, los pacientes sólo podrán ser atendidos por los cirujanos que ya se encuentren asignados a consultas.

Ambas devuelven los valores de las funciones objetivo, el equilibrio de los cirujanos y los pacientes atendidos con retraso.

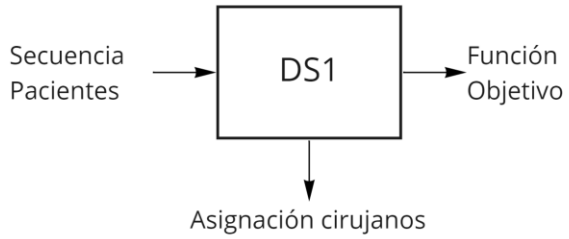


***Figura 3-1**: Esquema función DS1.*

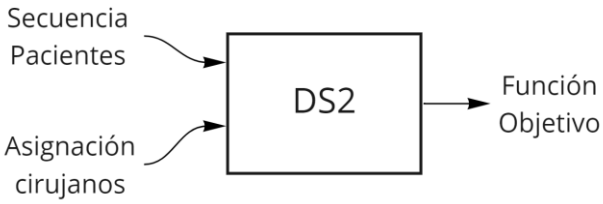


***Figura 3-2:** Esquema función DS2.*

La estructura de funcionamiento es la misma para las funciones DS1 y DS2. Se presenta el funcionamiento general en la ***Figura 3-3*** y ***Figura 3-4***. Esta función recorre el vector secuencia de pacientes, y para cada uno de ellos, según en qué etapa se encuentre, le asigna una consulta PAE, postoperatoria, o una intervención quirúrgica. El paciente se encuentra en la etapa 0 si espera una consulta PAE, la etapa 1 si espera una intervención quirúrgica, y la etapa 2 si espera una consulta postoperatoria. Cuando todos los pacientes hayan pasado por el bucle, se determina el equilibrio de horas de trabajo de los cirujanos, que se presenta como la varianza de las horas acumuladas en consulta y la varianza de las horas acumuladas en el quirófano. Además, se determina qué pacientes han superado los tiempos de respuesta de sus correspondientes etapas, y se obtiene el valor de la función objetivo llamando a la función 'tardiness'.

```
def DS1 (secuencia):
    leerexcel()
    for i not in secuencia:
        tiempo acumulado en lista de espera+=1
    for i in secuencia:
        if estado=0:
            busquedaPAE(i)
        if estado=1:
            busquedaOR(i)
        if estado=2:
            busquedaPOSTOP(i)
    actualizar datos de pacientes y cirujanos #tiempos acumulados, estados,
    cirujanos asociados, etc.
    eq=[varianza de tiempos en consultas, varianza de tiempos en quirófano]
    tardes=lateness(pacientes)
    utilizacion = porcentaje utilización de consultas y quirófanos
    FO = eq/25 + tardes/(332*15*len(pacientes)) - utilizacion
    return eq, tardes, utilizacion, FO #[[varianza en tiempos en consultas,
    varianza en tiempos en quirófano], lateness ]
```

***Figura 3-3**: Pseudocódigo función DS1*

```
def DS2 (secuencia, cirujanosconsultas):
    leerexcel()
    for i not in secuencia:
        tiempo acumulado en lista de espera+=1
    for i in secuencia:
        if estado=0:
            busquedaPAE(i)
        if estado=1:
            busquedaOR(i)
        if estado=2:
            busquedaPOSTOP(i)
    actualizar datos de pacientes y cirujanos #tiempos acumulados, estados, cirujanos
    asociados, etc.
    eq=[varianza de tiempos en consultas, varianza de tiempos en quirófano]
    tardes=lateness(pacientes)
    utilizacion = porcentaje utilización de consultas y quirófanos
    FO = eq/25 + tardes/(332*15*len(pacientes)) - utilizacion
    return eq, tardes, utilizacion, FO #[[varianza en tiempos en consultas, varianza
    en tiempos en quirófano], lateness ]
```

Figura 3-4: *Pseudocódigo función DS2.*

La función tardiness recorre el vector de los tiempos acumulados en la lista de espera de cada paciente. Para cada uno de ellos, según en qué etapa se encuentre, analiza si dicho paciente ha excedido el tiempo de respuesta de su etapa. Si es así, se añade a un vector tardy el producto del peso de cada paciente por el número de días que va con tardanza. Así, será peor cuando los pacientes que van tarde son urgentes. Se muestra en la ***Figura 3-5***. La función tardiness devuelve la suma de todos los elementos del vector tardy.

```
def tardiness(pacientes):
    global pesos
    x=0
    tardy=[]
    for x in range(len(pacientes)):
        if estado[x]==0:
            if tiempoacum[x]>trp[x]:
                tardy.append((tiempoacum[x]-trp[x])*pesos[x])
        if estado[x]==1:
            if tiempoacum[x]>trq[x]:
                tardy.append((tiempoacum[x]-trq[x])*pesos[x])
        if estado[x]==2:
            if tiempoacum[x]>trc[x]:
                tardy.append((tiempoacum[x]-trc[x])*pesos[x])
    return sum(tardy) #tardy indica cuántos pacientes van con retraso
    actualmente
```

Figura 3-5: *Pseudocódigo función objetivo Tardiness*

A continuación, se va a explicar la lógica que siguen las subfunciones para asignar una consulta PAE, una intervención quirúrgica, y una consulta postoperatoria de DS1 y DS2.

3.1.1 DS1

La función DS1 es aquella que recibe como entrada únicamente el vector secuencia de pacientes.

Si el paciente se encuentra en la etapa 0, es decir, está esperando una consulta PAE, se llama a la función búsquedaPAE para poder atenderlo. Aquí se le buscará un hueco y un cirujano. En la búsqueda de una consulta PAE, se asigna el cirujano según se le necesite en la consulta para atender al paciente. Lo primero que se realiza es una lista de los cirujanos de la unidad, el vector 'cirujanosdelaunidad', ordenada de menor a mayor número de horas acumuladas en consulta. Así, se asignará primero el paciente a aquel cirujano que lleve menos horas de consulta para poder así compensar con el resto. A continuación, para cada cirujano de esta lista se comprueba si está asignado a alguna consulta, y si es así, si en dicha consulta hay hueco para atender a este nuevo paciente. Si hay hueco, el paciente es asignado a dicha consulta con dicho cirujano. En caso de no tener ya una consulta asignada, se comprueba si hay alguna consulta libre, y se asigna el cirujano a dicha consulta si el cirujano no descansa dicho día. El funcionamiento se muestra en la ***Figura 3-6***.

```
def busquedaPAE(paciente):
    cirujanosdelaunidad=lista de cirujanos de la unidad sin descanso ese día,
    ordenados de menor a mayor horas acumuladas en consulta
    for i in cirujanosdelaunidad:
        if i in cirujanosconsultas[día]: #está asignado a una consulta ese día
            if ctconsultas[consulta]+dp[paciente]<8: #si hay hueco
                return [consulta, cirujano asignado]
    Si no ha sido asignado ya, se busca nuevo hueco
        for each consulta
            if cirujanosconsultas[dia][consulta]==[] #la consulta está libre
                for x in cirujanosdelaunidad:
                    if x not in cirujanosOR[dia]:
                        return [consulta, x]
```

Figura 3-6: *Pseudocódigo BúsquedaPAE de DS1*

De igual manera, para la búsqueda de una consulta Postoperatoria del paciente, lo primero que se comprueba es si el cirujano asociado está ya asignado a una consulta. De ser así, se comprueba si hay hueco para el nuevo paciente. En caso de no haber hueco, este paciente no puede ser asignado dicho día.

Si el cirujano asociado no tiene previamente una consulta asignada, se le puede asignar en una consulta que esté libre dicho día, siempre y cuando el cirujano no tenga el día de descanso. El funcionamiento se muestra en la ***Figura 3-7***.

```
def busquedaPOSTOP(paciente):
    if cirujanoasociado está asociado a consulta and not in cirujanosOR[dia] and
    descanso[dia][cirujanoasociado]==0:
        if ctconsultas[consulta]+dp[paciente]<8: #si hay hueco
            return [consulta, cirujanoasociado]

    Si no ha sido asignado ya:
        #se le busca nueva consulta
        for x in consultas:
            if cirujanosconsultas[dia][x]==[] #consulta libre
                if cirujanoasociado not in cirujanosconsultas[dia] and descanso[dia][
                cirujanoasociado]==0: #el cirujano no tiene ya una consulta asignada
                    return [x, cirujanoasociado]
```

Figura 3-7: *Pseudocódigo BúsquedaPOSTOP de DS1*

Con estas funciones, el cirujano es asignado a una consulta según se le requiere cada día y es la secuencia de pacientes la que define qué cirujanos atienden en cada consulta cada día de la semana. En estas funciones se prima que el paciente sea atendido siempre por un cirujano que ya esté asignado en alguna consulta o quirófano. Con ello se evitan tantos cambios de personal que provocan retrasos y movimientos innecesarios.

Cuando el paciente se encuentra en la etapa 1, se llama a la función búquedaOR para planificarle la intervención quirúrgica. Para asignar el equipo y un quirófano a la operación de un paciente, el procedimiento es más complejo. En primer lugar, se obtienen las listas de cirujanosexpunidad y asistexpunidad. Son los cirujanos de la unidad que no tengan descanso ese día y cumplan con la experiencia para ser cirujano principal y cirujano asistente respectivamente. A continuación, si la cirugía no requiere de cirujano asistente, se busca entre los cirujanosexpunidad si ya hay alguno asignado a un quirófano y con tiempo suficiente para operar también al nuevo paciente. Si no se cumplen estas dos condiciones, se busca un nuevo OR para atender al paciente. Para cada OR, se mira en primer lugar si hay hueco para atender un nuevo paciente. Si es así, se busca entre los cirujanoexpunidad si hay alguno que no esté asignado a consultas ese día y que de estar asignado a un OR lo esté al OR en cuestión únicamente. Si se cumplen las condiciones, se le asigna. Si tras recorrer todos los quirófanos y cirujanos no se encuentra ninguna asignación posible, eso indica que el paciente no podrá ser operado dicho día.

En cambio, si la intervención quirúrgica sí requiere de cirujano asistente, hay que buscar dos cirujanos disponibles. Al igual que en el caso anterior, se busca si algún cirujanoexpunidad está ya en un OR ese día y hay hueco para un nuevo paciente. De ser así, se busca en asistexpunidad algún cirujano asistente, diferente del cirujano principal, que no esté asignado a consultas y que pueda asistir en la intervención. Si no se encuentra asignación posible, hay que recorrer cada quirófano disponible buscando un intervalo de tiempo libre. Para el intervalo, se busca un cirujanoexpunidad que no esté ya asignado a consultas. Una vez se tenga el cirujano principal, se realiza la misma búsqueda para el cirujano asistente. Si no se encuentra ningún asistente disponible, se pasa al siguiente cirujano disponible. Si no se encuentra con ningún cirujano disponible, se pasa al siguiente quirófano con hueco disponible. Si ninguna asignación es posible, el paciente deberá esperar al siguiente día para ser atendido. El procedimiento se muestra en la ***Figura 3-8***.

```
def busquedaOR (paciente):
    cirujanosexpunidad=lista de cirujanos de la unidad que cumplen con la
    experiencia requerida para el cirujano principal de la cirugía y no
    tengan descanso ese día, ordenados de menor a mayor horas acumuladas en
    quirófano

    asistexpunidad = lista de cirujanos de la unidad que cumplen con la
    experiencia requerida para el cirujano asistente de la  cirugía y no
    tengan descanso ese día , ordenados de menor a mayor horas acumuladas en
     quirófano
    if hs[paciente]=0: #la operación no requiere de asistente
        for i in cirujanosexpunidad:
            Se busca si hay alguno ya asignado a un OR donde haya hueco para
             realizar otra operación más.
            return [OR, cirujanoprincipal, 'NaN']
        Si no hay, se busca si hay huecos en otros ORs. En el que haya hueco
        , se busca en cirujanosexpunidad que no esté asignado a consultas y
        pueda asignarse a ese OR.
            return [OR, cirujanoprincipal, 'NaN']

    if hs[paciente]!=0 #sí es necesario un asistente
        for i in cirujanosexpunidad:
            Se busca si hay alguno ya asignado a un OR donde haya hueco para
             realizar otra operación más.
                Se busca en asistexpunidad un asistente que sea distinto a i
                , que no esté asignado a consultas, y que de estar asignado
                a un OR sea este mismo
                    return [OR, cirujanoprincipal, asistente]

        Si no se ha asignado ya, hay que buscar un OR nuevo.
        for i in cirujanosexpunidad:
            while n<número de ORs:
                if i no está en consultas o en otros quirófanos:
                    for k in asistexpunidad:
                        Se busca k que no esté en consultas ni en otros
                        quirófanos
                            return [OR, cirujanoprincipal, asistente]
```

Figura 3-8: *Pseudocódigo función BúsquedaOR de DS1 y DS2*

3.1.2 DS2

La function DS2 es aquella que tiene como entrada la secuencia de pacientes y la asignación de cirujanos a consultas.

La estructura del funcionamiento de DS2 (secuencia, cirujanosconsultas) es la misma que la de DS1. También permanece idéntica la función 'búsquedaOR'. Varían las funciones búsquedaPOSTOP y búsquedaPAE, para la asignación de los pacientes en estas etapas. Estas funciones se simplifican, ya que la

asignación de cirujanos a consultas viene ya dada y no es posible una modificación. Por tanto, no se asignan nuevos cirujanos a consultas. El procedimiento quedaría como se explica a continuación.

Para asignar una consulta PAE a un paciente, al igual que en DS1, lo primero que se realiza es una lista de los cirujanos de la unidad ordenada de menor a mayor número de horas acumuladas en consulta. Así, se asignará primero el paciente a aquel cirujano que lleve menos horas de consulta para poder así compensar con el resto. A continuación, para cada cirujano de esta lista se comprueba si está asignado a alguna consulta, y si es así, si en dicha consulta hay hueco para atender a este nuevo paciente. Si hay hueco, el paciente es asignado a dicha consulta con dicho cirujano. En caso de no haber hueco o no haber un cirujano de la lista asignado a consultas, el paciente quedará sin atender dicho día. El procedimiento detallado se muestra en la ***Figura 3-9***.

```
def busquedaPAE(paciente):
    cirujanosdelaunidad=lista de cirujanos de la unidad, ordenados de
    menor a mayor horas acumuladas en consulta
    for i in cirujanosdelaunidad:
        if i in cirujanosconsultas[día]: #está asignado a una consulta
        ese día
            if ctconsultas+dp[paciente]<8: #si hay hueco
                return [consulta, cirujano asignado]
```

Figura 3-9: *Pseudocódigo función BúsquedaPAE de DS2*

Para asignar una consulta postoperatoria a un paciente, ya que el paciente sólo puede ser atendido por el cirujano que le realizó la intervención quirúrgica, el procedimiento únicamente consta de verificar si el cirujano asociado está asignado a una consulta, y si es así si tiene hueco libre para atender al paciente. En caso de no tener hueco en su consulta, o no estar asignado a consultas ese día, el paciente no podrá ser atendido dicho día. Se presenta el procedimiento en la ***Figura 3-10***.

```
def busquedaPOSTOP (paciente):
    if cirjasoc[paciente] in cirujanosconsultas[día]: #si el cirujano
    asociado tiene consulta ese día
        if ctconsultas[dia][consulta]+dp[paciente]<8: #si hay hueco
                return [consulta, cirujano asignado]
```

Figura 3-10: *Pseudocódigo función BúsquedaPOSTOP de DS2*

En todas estas funciones, siempre se intenta que sean elegidos en primer lugar los cirujanos con menos horas acumuladas para así compensar con el resto y disminuir la varianza, que es lo que se pretende con las funciones objetivo.

3.2 Metaheurísticas Propuestas

Una vez explicado el funcionamiento que sigue el código para la planificación y la obtención de la función objetivo, se procede a explicar la integración del algoritmo Iterative Greedy en cada una de las metaheurísticas para la generación de las entradas de las funciones.

Como toda metaheurística, en primer lugar, se obtiene una solución inicial, a partir de ella se diversifica, y finalmente se itera hasta cumplir un criterio de parada.

Como se explicó en la introducción, el algoritmo Iterative Greedy es un algoritmo que consiste en aplicar iterativamente tres fases: inicialización, destrucción, donde algunos trabajos son eliminados de la secuencia de forma aleatoria, y la fase de construcción, en la que estos trabajos son nuevamente insertados en la secuencia. En la inicialización, se crea una secuencia inicial utilizando la heurística de construcción NEH.

Esta heurística de construcción consiste en los siguientes pasos:

1. En primer lugar, los tiempos de procesamientos de los pacientes en todas las etapas son ordenados de mayor a menor. La secuencia se ordena siguiendo este orden.
2. A continuación, uno por uno, se va insertando cada paciente en todas las posiciones de la secuencia, y aquella que mejor función objetivo obtenga es la secuencia resultante en la que se insertará el siguiente paciente. La evaluación de la función objetivo se realiza mediante las funciones DS1 y DS2.
3. Hay que repetir hasta que se obtenga una secuencia que contenga todos los pacientes de la lista de espera.

En la fase de destrucción, un número aleatorio de trabajos es extraído de la secuencia. En el código desarrollado este número es de cuatro trabajos, ya que es lo que se suele utilizar en la literatura para problemas de secuenciación. A continuación, en la fase de construcción, se insertan, en el mismo orden en el que fueron extraídos, los trabajos anteriores. Uno por uno, se van insertando en todas las posiciones posibles de la secuencia y se guarda aquella que mejor función objetivo obtenga para continuar con la inserción del siguiente trabajo.

Tras las fases de destrucción y construcción, se podría aplicar una búsqueda local. Aquí es donde se diferencian los algoritmos IG(x) de IG(x)-LS. Los algoritmos IG(X) no tienen una búsqueda local, y los algoritmos IGX-LS sí tienen integrada una búsqueda local tras la fase de construcción. En este trabajo se han realizado tres heurísticas diferentes y sus respectivas parejas con búsqueda local.

Se itera hasta que se cumpla un criterio de parada determinado. En este trabajo se ha establecido que el tiempo de parada será proporcional al tamaño de la instancia evaluada. Seguirá la siguiente fórmula:

$$T = I * R * H * 0.025\ (s) \tag{8}$$

donde I es el número de pacientes en la lista de espera, R es la cantidad de recursos disponibles, es decir, la suma de quirófanos y consultas, y H es el número de días en el horizonte temporal (Ramírez Rojas, 2020).

Cada algoritmo completo se procesaría cada día del horizonte temporal, encontrando así una planificación diaria. En la ***Figura 3-11*** se representa el pseudocódigo del algoritmo Iterative Greedy.

```
procedure IteratedGreedy_for_PFSP
    π := NEH_heuristic;
    π := IterativeImprovement_Insertion(π);
    π_b := π;
    while (termination criterion not satisfied) do
        π' := π;                                  % Destruction phase
        for i := 1 to d do
            π' := remove one job at random from π' and insert it in π'_R;
        endfor
        for i := 1 to d do                        % Construction phase
            π' := best permutation obtained by inserting job π_R(i) in all possible positions of π';
        endfor
        π'' := IterativeImprovement_Insertion(π');   % Local Search
        if C_max(π'') < C_max(π) then             % Acceptance Criterion
            π := π'';
            if C_max(π) < C_max(π_b) then         % check if new best permutation
                π_b := π;
            endif
        elseif (random ≤ exp{−(C_max(π'') − C_max(π))/Temperature}) then
            π := π'';
        endif
    endwhile
    return π_b
end
```

Figura 3-11: *Procedimiento del algoritmo Iterative Greedy con Local Search*

Siguiendo la estructura de la metaheurística mostrada en la ***Figura 1-3***, las metaheurísticas propuestas se estructuran todas de la siguiente manera:

- La solución inicial, obtenida mediante el procedimiento NEH. Esta solución inicial es el punto de partida para la exploración del resto de región admisible.
- La evaluación de la solución realizada por las funciones DS1 y/o DS2, las cuales reciben como entrada las secuencias y devuelven la Función Objetivo a comparar.
- El proceso de exploración, realizado en la fase de destrucción y construcción y en Local Search. Se trata de ir explorando distintos espacios de la región admisible variando las secuencias mediante la búsqueda de sus vecinos.
- Por último, el criterio de parada propuesto es el mostrado anteriormente en la Ecuación 8.

3.2.1 Metaheurísticas IG0 e IG0-LS

Presentada en la ***Figura 3-12***, esta metaheurística se basa en la obtención de la mejor secuencia de pacientes aplicando únicamente la función DS1. Partiendo de unos datos iniciales (lista de espera y plantilla de cirujanos), se aplica el algoritmo NEH para obtener una secuencia de pacientes inicial. A continuación, con la secuencia se pacientes obtenida del NEH, se aplica Iterative Greedy para optimizarla. Ha de tenerse en cuenta que al tener sólo un vector que optimizar, únicamente se aplica el algoritmo Iterative Greedy para esta lista de pacientes, buscando la óptima comparando según las funciones objetivos obtenidas a través de la función DS1. Una vez alcanzado el tiempo de parada, se obtiene la mejor combinación de secuencia de pacientes y asignación de cirujanos a consultas, conformando la solución que mejor función objetivo devuelve. La metaheurística IG0-LS aplicaría además tras las fases de destrucción y construcción en el Iterative Greedy la búsqueda local por inserción.

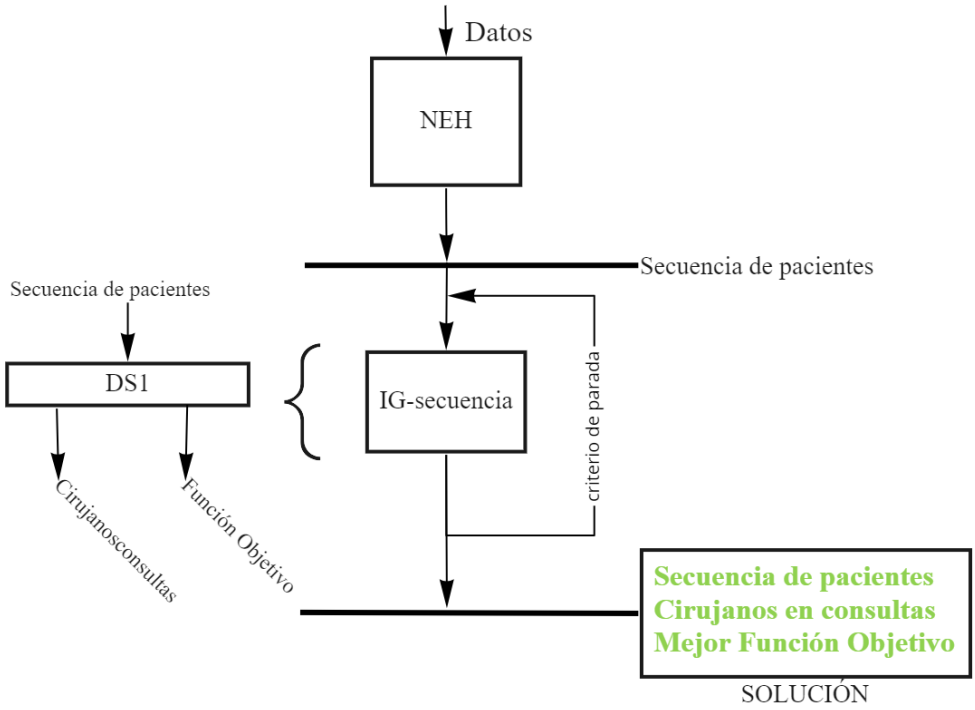


Figura 3-12: *Esquema heurística IG0.*

3.2.2 Metaheurísticas IG1 e IG1-LS

Se explica ahora el funcionamiento de la metaheurística IG1, mostrada en la ***Figura 3-13***. Esta se diferencia de IG0 en que en este caso se busca optimizar por separado tanto la secuencia de pacientes como la asignación de cirujanos a consultas. Ya que no se puede optimizar una de ellas sin la otra, hay que hacerlo por

separado. En esta metaheurística, se realiza primero la optimización de la secuencia de pacientes aplicando el algoritmo Iterative Greedy para ésta a través de la función DS1. Una vez obtenida la mejor secuencia de pacientes, ésta se deja fija y se optimiza la asignación de cirujanos a consultas. En la primera iteración, se da como entrada de DS2 el vector CirujanosConsultas obtenido al optimizar la secuencia de pacientes. Se vuelve a aplicar el Iterative Greedy pero ahora calculando la función objetivo con la función DS2, que tiene como entrada la secuencia de pacientes fija, y los cirujanos en consulta que va variando en cada iteración y es el vector que se intenta optimizar. Una vez terminados ambos procesos de Iterative Greedy, se obtiene la solución, la secuencia de pacientes y la asignación de cirujanos a consultas que optimizan la función objetivo. La metaheurística IG1-LS aplicaría además tras las fases de destrucción y construcción en el Iterative Greedy la búsqueda local por inserción.

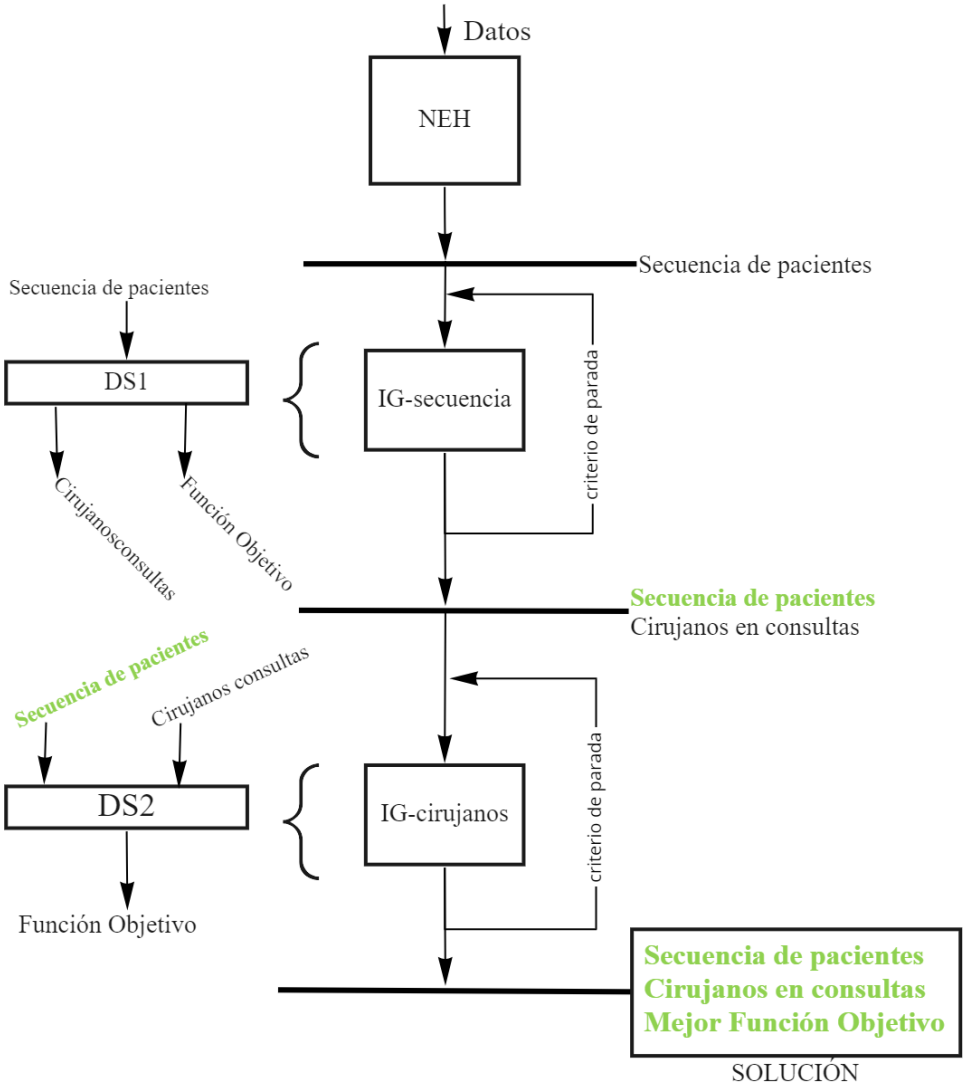


Figura 3-13: *Esquema heurística IG1.*

3.2.3 Metaheurísticas IG2 e IG2-LS

Se procede a explicar el funcionamiento de la heurística IG2, mostrada en la ***Figura 3-14***. Esta heurística es similar a IG1. La diferencia está en que, en este caso, sólo se aplica DS1 para obtener la asignación de cirujanos a consultas que genera con la secuencia dada por NEH. A partir de esta secuencia y esta asignación, en primer lugar, se aplica Iterative Greedy dejando fija la asignación y variando la secuencia de pacientes, introduciendo ambos vectores en DS2. En segundo lugar, una vez se ha optimizado la secuencia, ésta se deja fija y se va variando la asignación, y al igual que en el Iterative Greedy anterior, introduciendo ambos vectores en DS2. Finalmente se tendrían ambos vectores optimizados. La metaheurística IG2-LS

aplicaría además tras las fases de destrucción y construcción en el Iterative Greedy la búsqueda local por inserción.

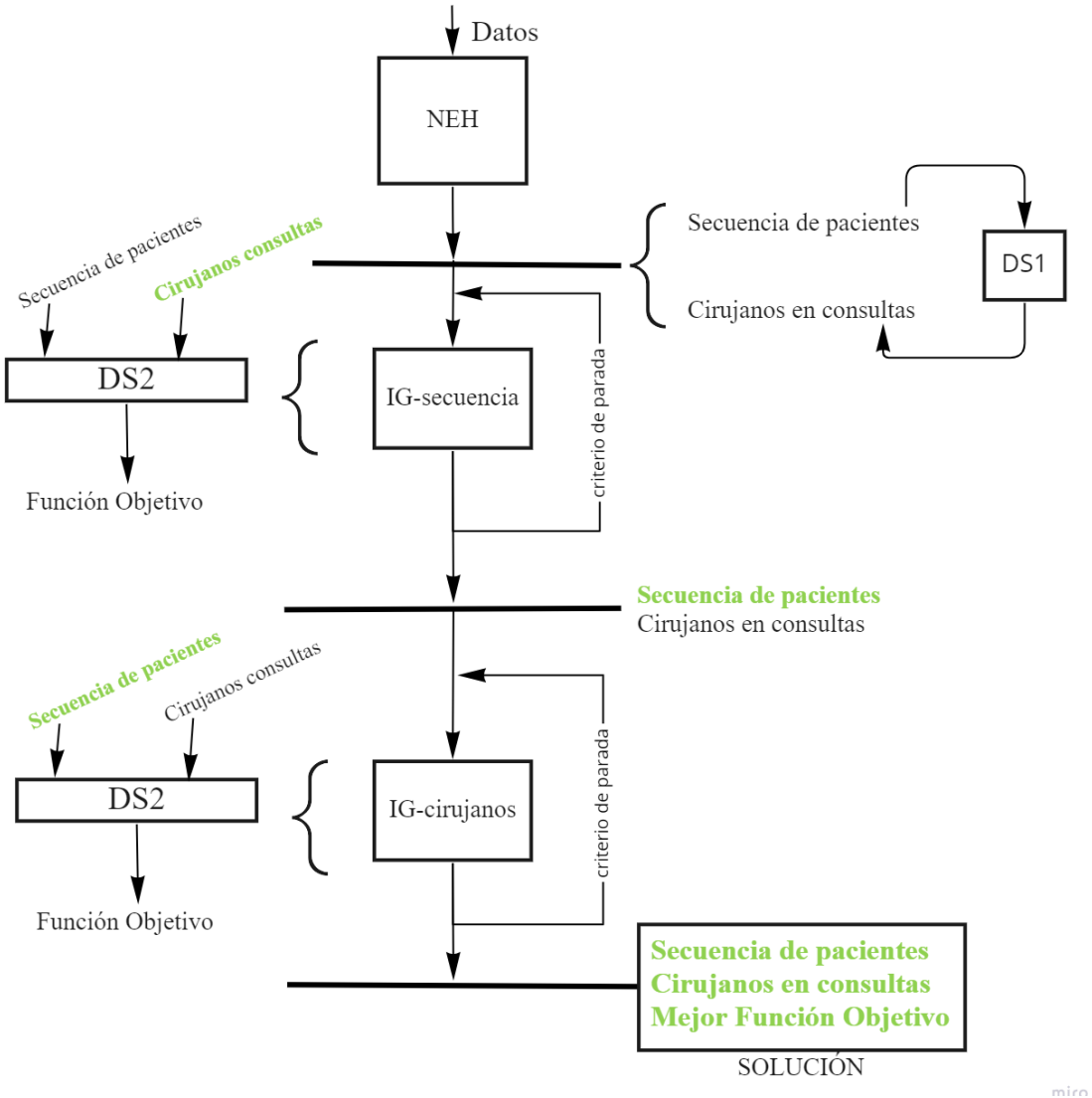


Figura 3-14: *Esquema heurística IG2.*

4 RESULTADOS COMPUTACIONALES

Para evaluar la validez de las metaheurísticas propuestas, se lleva a cabo la simulación para una serie de instancias de datos. Se observa cómo se comportan las distintas metaheurísticas para conjuntos de datos de distintos tamaños. Este estudio de simulación puede ser resumido en dos pasos (Framiñán, Leisten & Ruiz, 2014):

1. La generación de una batería de instancias de distintos tamaños aplicables al modelo.
2. El análisis de los algoritmos. Se comparan los distintos algoritmos aplicados y se asume que el rendimiento relativo que tiene cada uno para esta batería de instancias será similar al rendimiento relativo que tendrían para casos reales. La metaheurística que mejores resultados proporcione será seleccionada para resolver el caso real.

4.1 Generación de Instancias

Se ha generado una batería de instancias tomando como referencia a Molina-Pariente, Hans, et al., (2015)y Ramírez Rojas, (2020)Se genera la batería de instancias aplicando el mismo procedimiento que en los artículos mencionados, con las modificaciones necesarias para la adaptación al problema en estudio.

Para la generación de instancias se ha creado una función denominada 'GenerarInstancia', mostrada en la ***Figura 4-1***, cuya entrada son los parámetros mostrados en la ***Tabla 4-1***.

```
def generarinstancia(numor, numc, alfa, beta)
    l=1
    mds=3.5
    a=8
    CT=numc*8*7+numor*8*7
    while pacientes<P:
        Se añade un paciente a la lista de espera pacientes
        Se determinan los datos del último paciente añadido(experiencias, duraaciones,
         tiempos de espera, pesos, estados, tiempos acumulados)
        Duración media= duración media de los pacientes en el sistema
        P=((numc+numor)*8h*Beta)/(Duración Media)
    S=alfa*CT/mds/l/a+1
    Las unidades se establecen según los pacientes que haya con dicha unidad
    descansos=matriz aleatoria de S x 7
    Se cambia si hay algún día en el que falte algún cirujano de alguna unidad
    Las experiencias son aleatorias, se cambian si hay alguna intervención que no se
    pueda llevar a cabo por falta de experiencia
    Se asignan los cirujanos asociados
    Se escribe esta instancia en un Excel llamado 'instancia.xlsx'.
```

Figura 4-1: *Pseudocódigo de la función 'GenerarInstancia'.*

Tabla 4-1: *Parámetros para la generación de instancias*

Parámetro	Descripción	Valor
Numor	Número de quirófanos disponibles en el hospital	Se establece que puede tener valor de 3, 6 y 9
Numc	Número de consultas disponibles en el hospital	Se establece que puede tener valor de 3, 5 y 7
Alfa	Factor de control de cobertura del cirujano para recurso disponible. Se aplica al calcular el número de cirujanos de la instancia	Su valor puede ser 1.5 o 2. De esta forma, cada recurso disponible estará siempre cubierto por al menos un cirujano y no quedará ningún recurso sin cubrir
Beta	Porcentaje de saturación. Se aplica al calcular el número de pacientes en la lista de espera	Su valor es de 150% y de 200%. Así, se contemplarán los escenarios en los que la capacidad esté completa al 150% y 200% y no haya capacidad suficiente para atender a todos los pacientes.

En la ***Tabla 4-1*** se presentan los parámetros que van a ir variando y a partir de los cuales se genera cada instancia. Se establecen como parámetros los números de consultas y quirófanos disponibles del hospital. Sirve para comprobar cómo se comportan las metaheurísticas para distintas capacidades del hospital, cuantas más consultas y quirófanos haya disponibles, más capacidad tendrá el hospital y más pacientes deberá tener en su lista de espera. El número de quirófanos será de 3, 6 o 9, y el número de consultas será de 3, 5 y 7. El siguiente parámetro es alfa, α, que indica el nivel de cobertura de cada recurso disponible. Esto es, si alfa es 1.5, indica que cada espacio temporal de recurso disponible estará cubierto por 1.5 cirujanos. Este parámetro sirve para crear la plantilla de cirujanos de la instancia, asegurando que siempre habrá cirujanos para cubrir las consultas y quirófanos. El valor de alfa será de 1.5 y 2. Por último, se encuentra el parámetro beta, β, que representa la saturación de pacientes. Será de 150% y 200%. Este parámetro se utiliza al crear la lista de pacientes. Se crea una lista de espera con un número de pacientes elevado que supera en 50% y 100% la capacidad del hospital durante la semana. Implica por tanto que no habrá recursos disponibles para atender a todos los pacientes de dicha lista, y se asume que sería necesario un horizonte temporal más amplio para poder atender a todos.

Los datos que permanecen inalterados para todas las instancias son *mds*, el número de días que un cirujano trabaja cada semana. Se establece en 4,9 ya que, por probabilidad, al crear los descansos aleatoriamente con una matriz binaria se trabaja un 70% de la semana. El número de horas trabajadas al día, *a*, se queda fija en 8 horas.

Se comienza a crear la instancia por los pacientes. El número de pacientes, *P*, viene determinado por la Ecuación 4.

$$P = \frac{(numc * 6h + numor * 8h) * Beta}{Duración\ Media} \quad (9)$$

Se va generando cada paciente uno por uno, y la duración media de sus intervenciones es el promedio de la duración del paciente en la etapa que le corresponda. Así, se va rellenando una lista de espera uno a uno hasta que se completa la capacidad del hospital al porcentaje Beta. Para cada paciente, se obtienen sus datos de la siguiente manera:

- La unidad a la que pertenece se establece de manera aleatoria.
- La etapa en la que se encuentra se establece de manera aleatoria

- El peso que tiene en la lista de espera se le asigna teniendo en cuenta que un 50% de pacientes tienen prioridad baja, un 30% prioridad media, y un 20% prioridad alta.
- El tiempo acumulado del paciente en la lista de espera se genera aleatoriamente entre 1 y 200 días.
- La duración de las consultas se genera de forma aleatoria entre 0.3 y 0.6 horas. Se toma este dato del hospital, donde en 8 horas atienden a 25 pacientes. Por tener en cuenta los retrasos, se pone el límite superior en 0.6 horas.
- La duración de las cirugías sigue una distribución normal. La media, μ, es aleatoriamente escogida entre 1, 2, 3, 4 horas. La desviación típica, σ, es aleatoriamente escogida entre [0.1μ, 0.2μ, 0.3μ, 0.4μ, 0.5μ].
- La experiencia requerida del cirujano principal y del cirujano asistente para la intervención son aleatoriamente generadas.
- Los tiempos de respuesta de las consultas son aleatoriamente generados en un rango de 4 a 6 semanas, dato real del hospital.
- El tiempo de respuesta para las cirugías son los que establece el Sistema Nacional de Salud. 45 días para aquellas con urgencia alta, 180 días para urgencia media, y 360 días para urgencia baja.
- El cirujano asignado de cada paciente se asigna de forma aleatoria para aquellos que se encuentren esperando la consulta postoperatoria, teniendo en cuenta que debe ser un cirujano de su misma unidad.

Una vez se tienen todos los pacientes de la lista de espera, se crean los cirujanos. Se crea la plantilla de cirujanos para un horizonte temporal de una semana. El número de cirujanos para cada instancia vendrá determinado según la fórmula

$$S = \frac{\alpha * Capacidad\ Total}{mds * l * a} \quad (10)$$

donde α es la cobertura por parte de los cirujanos de cada recurso, la capacidad total es la cantidad de horas disponibles de recursos en una semana,

$$Capacidad\ Total = (numor * 8horas + numc * 6horas) * 7días \quad (11)$$

mds, como ya se ha explicado antes, es 4,9, número de días disponibles de cada cirujano en una semana, *l* indica el número de semanas en el horizonte temporal, que será 1, y *a* es 8, la cantidad de horas que trabajan los cirujanos al día.

Las unidades de cada cirujano se asignan en función de cuántos pacientes de dicha unidad haya en la lista de espera. La experiencia se asigna de forma aleatoria, teniendo en cuenta que debe haber siempre cirujanos de cada unidad que puedan realizar las cirugías de los pacientes.

Para los descansos de los cirujanos, se crea una matriz de tamaño S*7, en la que, de forma aleatoria, se asigna un 1 si tienen descanso, o un 0 si está disponible para trabajar ese día. La probabilidad de descanso se ha establecido en un 30%. Un 70% de la semana será de trabajo. Se tiene en cuenta además que no puede haber ningún día de la semana en la que no haya un cirujano de cada unidad.

Los datos generados en cada instancia de los pacientes y los cirujanos se muestran en ***Tabla 4-3*** y ***Tabla 4-4***.

Tabla 4-2: *Datos de cada instancia*

Numc	**Numor**	**α**	**β**
{3, 6, 9}	{3, 5, 7}	{1.5, 2}	{1.5, 2}

Tabla 4-3: *Datos de los pacientes*

P	$\frac{(numc * 6h + numor * 8h) * Beta}{Duración\ Media}$
u	[0, 7]
estado	[0, 2]
w	[0, 15] con probabilidad 50% [0,5], 30% [5,10], 20% [10,15]
tiempoacum	[1, 200]
dp, dc	[0.3, 0.6]
dq	N (μ, σ) μ = {1, 2, 3, 4} σ = {0.1μ, 0.2μ, 0.3μ, 0.4μ, 0.5μ}
hp	[1, 20]
hs	[0, 20]
trp, trc	[28, 42] días
trq	45, 180 o 365 según la urgencia
durmedia	Promedio de los tiempos de proceso en cada etapa
cirujasoc	Aleatorio con la misma unidad

Tabla 4-4: *Datos de los cirujanos*

S	$\frac{\alpha * Capacidad\ Total}{mds * l * a}$
u	[0, 7]
exp	[1, 20]
descanso	Aleatorio, probabilidad 30%

4.2 Simulación de resultados

Se generan en total 36 instancias de tamaños diferentes, una para cada combinación de los cuatro parámetros variables: número de consultas y quirófanos, alfa y beta.

Para cada instancia se crea un libro Excel de datos denominado 'instancia.xlsx'. Desde este archivo cada algoritmo IG0, IG0-LS, IG1, IG1-LS, IG2, IG2-LS lee los datos y los procesa generando una

planificación para nuestro horizonte temporal elegido de un día. El pseudocódigo de este procedimiento es el mostrado en la **Figura 4-2**. Se genera un bucle para cada parámetro, se llama a la función 'generarinstancia', y con la instancia creada se procesa cada metaheurística, recogiendo los datos devueltos en el Excel 'analisis.xlsx'.

La ejecución completa de las 36 instancias genera un libro Excel denominado 'analisis.xlsx', en el que se recogen los datos de cada instancia y los resultados devueltos. Los datos recogidos son los valores de los parámetros de entrada, número de cirujanos y de pacientes de cada instancia, el valor de la función objetivo total y el tiempo de ejecución para cada metaheurística. Una vez obtenida esa tabla, se calculan los valores ARPD, *Average Relative Percentage Deviation*, para cada instancia. Esto es, el porcentaje de desviación relativa con respecto a la mejor función objetivo obtenida. Cuanto menor sea este valor, menor desviación y más se acerca al mejor valor obtenido. Para su cálculo se sigue la ecuación 12. Los resultados generados se muestran en la **Tabla 4-5**.

```
Simulación

sumavarianzas=0
sumatardes=0
for numor in [3,6,9]:
    for numc in [3,5,7]:
        for alfa in [1.5,2]:
            for beta in [1,1.25]:
                generarinstancia(numor, numc, alfa, beta)
                se copia el archivo instancia.xlsx a data.xlsx
                ig0=IG0()
                se reinicia sumavarianzas y sumatardes
                ig0ls=IG0LS()
                se reinicia sumavarianzas y sumatardes
                ig1=IG1()
                se reinicia sumavarianzas y sumatardes
                ig1ls=IG1LS()
                se reinicia sumavarianzas y sumatardes
                ig2=IG2()
                se reinicia sumavarianzas y sumatardes
                ig2ls=IG2LS()
Los resultados se incluyen en el excel analisis.xlsx
```

Figura 4-2*: Pseudocódigo del procesamiento de cada instancia en las seis heurísticas construidas.*

$$ARPD = \left|\frac{FO_i - minFO}{minFO} * 100\right| \quad (12)$$

Tabla 4-5: *Tabla de resultados obtenidos para la batería de 36 instancias con valores ARPD*

	Consultas	Quiróf.	alfa	beta	Cirujanos	Pacientes	IG0	IG0-LS	IG1	IG1-LS	IG2	IG2-LS
0	3	3	1,5	1,5	12	53	0,51%	0,47%	0,51%	0,47%	0,51%	0%
1	3	3	1,5	2	12	77	9,28%	4,95%	0%	0,41%	0%	0,41%
2	3	3	2	1,5	15	71	2,68%	1,99%	1,15%	1,15%	2,68%	0%
3	3	3	2	2	15	56	20,40%	20,22%	10,65%	0%	5,98%	4,87%
4	5	3	1,5	1,5	15	69	7,27%	4,95%	4,08%	0%	3,24%	5,48%
5	5	3	1,5	2	15	109	1,57%	0%	1,57%	0,02%	1,57%	0,02%
6	5	3	2	1,5	20	66	6,23%	0%	6,58%	2,72%	3,10%	0%
7	5	3	2	2	20	90	6,31%	6,25%	0%	0%	6,31%	6,25%
8	7	3	1,5	1,5	18	88	0%	0%	0%	0%	0%	0%
9	7	3	1,5	2	18	112	2,67%	0,42%	2,15%	0%	2,06%	0,42%
10	7	3	2	1,5	24	75	1,71%	1,62%	1,58%	1,62%	1,71%	0%
11	7	3	2	2	24	111	8,61%	3,72%	7,70%	0%	6,46%	5,90%
12	3	6	1,5	1,5	18	69	4,78%	4,78%	1,86%	4,78%	0%	3,19%
13	3	6	1,5	2	18	97	6,03%	0%	0%	0%	6,27%	0%
14	3	6	2	1,5	24	66	1,21%	0%	1,21%	0%	1,21%	0%
15	3	6	2	2	24	126	9,96%	3,28%	9,11%	0%	9,11%	3,02%
16	5	6	1,5	1,5	21	88	8,86%	5,31%	5,75%	4,61%	6,01%	0%
17	5	6	1,5	2	21	109	3,46%	0%	3,46%	0,04%	3,43%	3,46%
18	5	6	2	1,5	28	124	2,42%	1,18%	0%	1,43%	2,20%	0,64%
19	5	6	2	2	28	140	1,91%	0%	1,87%	0,95%	1,16%	0,95%
20	7	6	1,5	1,5	25	108	2,34%	1,39%	4,21%	1,39%	1,96%	0%
21	7	6	1,5	2	25	148	9,76%	13,81%	9,76%	0%	9,76%	9,54%
22	7	6	2	1,5	33	123	8,46%	0%	8,95%	6,99%	8,95%	7,22%
23	7	6	2	2	33	144	6,90%	0%	1,69%	0%	5,45%	0%
24	3	9	1,5	1,5	25	107	3,65%	0%	3,65%	0,25%	3,65%	2,59%
25	3	9	1,5	2	25	165	4,96%	0,31%	4,92%	0,20%	0,71%	0%
26	3	9	2	1,5	33	122	8,39%	0%	6,32%	2,53%	4,96%	0,18%
27	3	9	2	2	33	160	10,32%	1,26%	9,77%	0,41%	10,62%	0%
28	5	9	1,5	1,5	28	142	3,77%	0%	3,77%	0%	3,77%	0%
29	5	9	1,5	2	28	184	8,41%	0%	8,41%	0%	8,41%	0%
30	5	9	2	1,5	37	128	0,76%	0,38%	0%	0,04%	0,76%	0,38%
31	5	9	2	2	37	159	3,32%	0%	3,20%	0,59%	3,32%	0,59%
32	7	9	1,5	1,5	31	158	0,05%	0%	0,19%	0,05%	0,19%	0,19%
33	7	9	1,5	2	31	197	3,92%	3,95%	0,10%	0%	0,23%	2,70%
34	7	9	2	1,5	41	124	1,60%	0,43%	1,56%	0%	1,56%	1,56%
35	7	9	2	2	41	208	3,01%	1,83%	2,21%	0%	3,01%	2,21%

4.3 Comparación de algoritmos

Una vez ejecutada y completada la simulación de las instancias, se lleva a cabo un análisis del funcionamiento de los algoritmos y se escoge aquél que mejor rendimiento demuestre para realizar la simulación en horizonte continuo de una semana.

Para el análisis del funcionamiento se realizan dos comparaciones: por un lado, se obtienen los valores del porcentaje de desviación con respecto a la mejor solución para cada algoritmo, para así analizar, en total, qué heurística ha conseguido mejores resultados para el grupo de instancias. Como se ha comentado

anteriormente, la mejor metaheurística será aquella que menor valor de desviación obtenga, ya que significa que se habrá acercado más veces al mejor valor obtenido. En la **Tabla 4-6** se presenta para cada metaheurística, el promedio de los valores de ARPD que ha devuelto. Es la metaheurística IG1-LS la que mejores resultados devuelve, la que menor desviación presenta en general con respecto a las mejores soluciones.

Tabla 4-6: *Promedio de valores ARPD para la batería de 36 instancias*

	IG0	**IG0-LS**	**IG1**	**IG1-LS**	**IG2**	**IG2-LS**
Promedio ARPD	5,2%	2,3%	3,6%	**0,9%**	3,6%	1,7%

Por otro lado, se analiza qué heurística ha conseguido encontrar la solución óptima o más cercana a la óptima el mayor número de veces. En la ***Tabla 4-7***, se indica con un 1 y resaltado en azul si la metaheurística ha llegado a la mejor solución, y con un 0 si no lo ha hecho. En la última fila de la tabla se ha sumado el número de veces que la metaheurística ha sido la mejor, y de nuevo, es IG1-LS aquella que obtiene los mejores resultados.

La metaheurística IG1-LS se determina por tanto como la que mejores soluciones aporta para distintos tamaños de instancias. Es aquella que combina las funciones DS1 y DS2. DS1 la aplica al optimizar la secuencia de pacientes, y con dicha secuencia fija, optimiza mediante DS2 la asignación de cirujanos a consultas. Si muestra este comportamiento para la batería de instancias, se asume que presentará un comportamiento relativo similar a largo plazo y también para casos reales.

Cabe destacar que, como es de esperar, aquellas metaheurística que aplican la búsqueda local en el Iterative Greedy obtienen mejores soluciones que aquellas que no lo hacen. IG2-LS es la segunda metaheurística que mejor se comporta con respecto a los valores obtenidos de ARPD, pero sin embargo IG0-LS obtiene la mejor solución más veces.

El tiempo computacional de las instancias ha sido, de media, 2,43 horas por instancia.

Tabla 4-7: *Análisis mejores soluciones de la Función Objetivo*

	IG0	IGOLS	IG1	IG1LS	IG2	IG2LS
	0	0	0	0	0	1
	0	0	1	0	1	0
	0	0	0	0	0	1
	0	0	0	1	0	0
	0	0	0	1	0	0
	0	1	0	0	0	0
	0	1	0	0	0	1
	0	0	1	1	0	0
	1	1	1	1	1	1
	0	0	0	1	0	0
	0	0	0	0	0	1
	0	0	0	1	0	0
	0	0	0	0	1	0
	0	1	1	1	0	1
	0	1	0	1	0	1
	0	0	0	1	0	0
	0	0	0	0	0	1
	0	1	0	0	0	0
	0	0	1	0	0	0
	0	1	0	0	0	0
	0	0	0	0	0	1
	0	0	0	1	0	0
	0	1	0	0	0	0
	0	1	0	1	0	1
	0	1	0	0	0	0
	0	0	0	0	0	1
	0	1	0	0	0	0
	0	0	0	0	0	1
	0	1	0	1	0	1
	0	1	0	1	0	1
	0	0	1	0	0	0
	0	1	0	0	0	0
	0	1	0	0	0	0
	0	0	0	1	0	0
	0	0	0	1	0	0
	0	0	0	1	0	0
TOTAL	1	15	6	16	3	14

5 CASO DE ESTUDIO: HORIZONTE CONTINUO

A continuación, se presenta el experimento computacional realizado para la heurística escogida en el capítulo anterior, IG1-LS a lo largo de un horizonte continuo de una semana.

5.1 Interfaz Usuario

Se ha creado el programa para su uso a través del Excel del responsable de la planificación del hospital con el objetivo de facilitar la tarea de éste. Se ha elegido por ser una interfaz clara, simple, y de uso común. En un libro Excel, denominado para este trabajo 'data.xlsx', se incluirán todos los datos de los pacientes en la lista de espera, así como los cirujanos de la plantilla y el número de consultas y quirófanos disponibles durante la semana. Una vez se ejecute el programa a través de un interpretador de Python, esta hoja Excel se actualizará con los nuevos datos: actualización de los tiempos acumulados en la lista de espera, la actualización de la etapa en la que se encuentra el paciente si es que ha avanzado en la planificación hecha, actualización del cirujano asociado si es que ha pasado por la intervención, etc. Además, ejecutado el código, se creará otro libro Excel denominado 'solucion.xlsx' donde se recoge la planificación obtenida: qué cirujano estará asignado a cada puesto cada día, y qué pacientes son atendidos y dónde.

Para utilizar esta interfaz, los datos que debe introducir el usuario en el libro Excel 'data.xlsx' son:

En la pestaña 'Pacientes', mostrada en la Figura 5-1, se introducirán los datos de cada paciente:

- Nombre del paciente/Número asociado al paciente.
- DP: Duración de la consulta PAE. Todas las consultas PAE se estiman que tengan la misma duración, independientemente de la unidad del paciente.
- TRP: Tiempo de respuesta para la consulta PAE. Éste se mide desde que el paciente entra en la lista de espera.
- DQ: Duración de la cirugía. Sólo tendrá que ser notificada cuando el paciente entre en la etapa 1, en la que espera una cirugía.
- TRQ: Tiempo de respuesta para la cirugía. Ésta se mide desde que el paciente termina su consulta PAE.
- DC: duración de la consulta Postoperatoria. Al igual que las consultas PAE, todas las consultas postoperatorias deben tener la misma duración independientemente de la unidad a la que pertenece el paciente.
- TRC: Tiempo de respuesta para ser atendido en una consulta postoperatoria. Este tiempo se mide desde que el paciente termina en quirófano.
- U: unidad a la que pertenece el paciente.
- W: peso del paciente en la lista de espera. Nivel de urgencia.
- HP: Nivel de experiencia del cirujano principal que requiere la cirugía. Este número irá de 1 a 20.
- HS: Nivel de experiencia del cirujano asistente que requiere la cirugía. Este número irá de 0 a 20. Si es 0, significa que la cirugía no requiere de cirujano asistente.
- CIRUJANO: Cirujano asociado para la consulta postoperatoria. El cirujano asociado es el cirujano que realizó la operación quirúrgica.
- ESTADO: Etapa en la que se encuentra el paciente.
    - Estado 0: el paciente espera que le asignen una consulta PAE.
    - Estado 1: el paciente espera ser operado.
    - Estado 2: el paciente espera una consulta postoperatoria.
- TACUM: tiempo que lleva el paciente acumulado en dicha etapa en la lista de espera.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Número paciente	dp	trp	dq	trq	u	hp	hs	dc	trc	w	estado	cirujano	tacum
2	0	0	0,309175042	39	0,951127952	45	6	3	8	0,550160168	39	11	1	NaN	11
3	1	1	0,309175042	39	2,267155831	45	3	5	9	0,550160168	39	10	1	NaN	121
4	2	2	0,309175042	39	4,913043619	360	7	17	18	0,550160168	39	4	1	NaN	6
5	3	3	0,309175042	39	5,045602895	45	4	18	6	0,550160168	39	14	1	NaN	104
6	4	4	0,309175042	39	2,103798701	360	6	12	6	0,550160168	39	4	1	NaN	90
7	5	5	0,309175042	39	4,877168464	360	3	10	0	0,550160168	39	0	0	NaN	137
8	6	6	0,309175042	39	0,746241446	360	3	17	20	0,550160168	39	5	2	9	38
9	7	7	0,309175042	39	2,949667915	360	3	14	1	0,550160168	39	2	0	NaN	35
10	8	8	0,309175042	39	0,629817502	45	4	18	18	0,550160168	39	10	1	NaN	123
11	9	9	0,309175042	39	2,376143638	180	7	7	2	0,550160168	39	9	1	NaN	39
12	10	10	0,309175042	39	0,863616783	360	3	17	16	0,550160168	39	1	0	NaN	148

Pacientes Cirujanos Datos +

Figura 5-1: *Ejemplo pestaña 'Pacientes' en libro 'data.xlsx'*

Datos que se deben introducir de los cirujanos de la plantilla en la hoja 'Cirujanos' (Ver **Figura 5-2**):

- Número asignado del cirujano.
- G: Grado de experiencia que tiene el cirujano. Este dato definirá si se le pueden asignar ciertas cirugías o no está capacitado aún para ello.
- S: Unidad a la que pertenece.

	A	B	C	D	E
1		Número Cirujano	G	S	
2	0	0	19	5	
3	1	1	20	1	
4	2	2	20	4	
5	3	3	20	0	
6	4	4	20	5	
7	5	5	20	7	
8	6	6	8	0	
9	7	7	20	6	
10	8	8	20	2	
11	9	9	20	3	
12	10	10	9	2	
13	11	11	10	1	
14	12	12	16	3	
15	13	13	16	4	
16	14	14	8	0	
17	15	15	13	7	
18	16	16	19	6	

Pacientes Cirujanos Datos +

Figura 5-2: *Ejemplo pestaña 'Cirujanos' en libro 'data.xlsx'*

Datos que se debe introducir del hospital en la hoja 'Datos' (Ver Figura 5-4):

- Número de consultas
- Número de quirófanos

		Número de quirófanos	Número de consultas
0		6	5

Pacientes | Cirujanos | Datos

Figura 5-3: *Ejemplo pestaña 'Datos' en libro 'data.xlsx'*

El código en Python leerá los datos de este Libro Excel y, una vez se ejecute el código, el programa creará un nuevo libro Excel llamado 'Solución.xlsx', en el que se va a mostrar para cada día la planificación obtenida. Cada pestaña de esta hoja tendrá la planificación de un día. Ver **Figura 5-4**.

Se encuentran los siguientes datos:

- Espacio: 'C' indica que se trata de una consulta. Por orden, serán las consultas 1, 2, 3, 4, 5. 'O' indica que se trata de un quirófano, y al igual que las consultas, serán por orden quirófano 1, quirófano 2, etc.
- Los pacientes que serán atendidos en las consultas, en la columna 'Pacientes'
- Los cirujanos que son asignados a las consultas, teniendo en cuenta que sólo puede haber un cirujano en cada consulta y este atenderá a todos los pacientes asignados a dicha consulta.
- Los pacientes que serán operados y en qué quirófano
- Los cirujanos que actuarán como cirujano principal en las distintas operaciones
- Los cirujanos que actuarán como cirujano asistente en las distintas operaciones, en la columna 'Asistentes'.

		Espacio	CIRUJANOS	ASISTENTES	PACIENTES
0		C	[34]	[]	[159, 116, 105, 131, 44, 34, 67, 93, 53, 30]
1		C	[63]	[]	[155, 101, 8, 140, 66, 22, 29, 31, 118, 2, 90, 125, 76]
2		C	[19]	[]	[16, 49, 144, 6, 77, 89, 87, 115, 36, 151, 157, 70]
3		C	[39]	[]	[103, 19, 45, 23, 97, 68, 83, 124, 55, 11, 3, 121, 58]
4		C	[47]	[]	[117, 109, 112, 5, 127, 35]
5		C	[21]	[]	[71, 129, 154, 74, 1, 17, 143, 113, 62, 38, 39]
6		C	[38]	[]	[91, 156, 37, 92, 26, 9, 96, 13]
7		O	[6, 18, 6, 6]	[59, 32, 13, 18]	[47, 61, 133, 79]
8		O	[60, 9, 42, 9, 9, 60]	[9, 45, 33, 26, 26, 26]	[108, 142, 4, 85, 149, 153]
9		O	[55, 55, 15, 55]	[15, 64, 24, 61]	[141, 110, 18, 150]
10		O	[28, 14, 22, 17, 14]	[57, 52, 49, 37, 57]	[15, 99, 82, 63, 136]
11		O	[7]	[29]	[104]
12		O	[]	[]	[]

LUNES | MARTES | MIERCOLES | JUEVES | VIERNES | SABADO | DOMINGO

Figura 5-4: *Ejemplo planificación del lunes en libro 'solucion.xlsx'.*

En el caso de la **Figura 5-4**, en la Consulta 1 atenderá el cirujano número 34 durante el lunes, y los pacientes atendidos en ella serán todos los que aparecen en la celda E2, empezando por el paciente 159 y finalizando la jornada con el paciente 30. En el quirófano 1 se realizarán 4 intervenciones. La primera de ellas

cuenta con el cirujano número 6 como principal y el cirujano 59 como asistente. El paciente operado es el 47. Tras esta operación, se intervendrá al paciente 61 con el cirujano 18 como principal y el cirujano 32 como asistente.

5.2 Análisis de Resultados

Una vez se tiene el Excel con la lista de espera completa, se procede a ejecutar el código para obtener la planificación que seguirá el hospital durante la semana. Ésta se recoge en la hoja Excel 'Solución.xlsx'. Se ha procesado para una instancia de 4 quirófanos y 3 consultas, una saturación de lista de espera, beta, del 200%, y el parámetro alfa de cobertura de recursos por los cirujanos es de 2. Con estos parámetros se genera la instancia procesada, la cual tiene 320 pacientes y 18 cirujanos. La planificación obtenida para las consultas, para cada día, asumiendo que comenzaría el turno a las ocho de la mañana, se muestra en Figura 5-5 a Figura 5-11 Se detalla el paciente que es atendido en cada intervalo de tiempo por cada cirujano El eje vertical representa la consulta en la que será atendido, y el eje horizontal el intervalo de tiempo. El horario de consultas se supone desde las 8 de la mañana hasta las dos de la tarde.

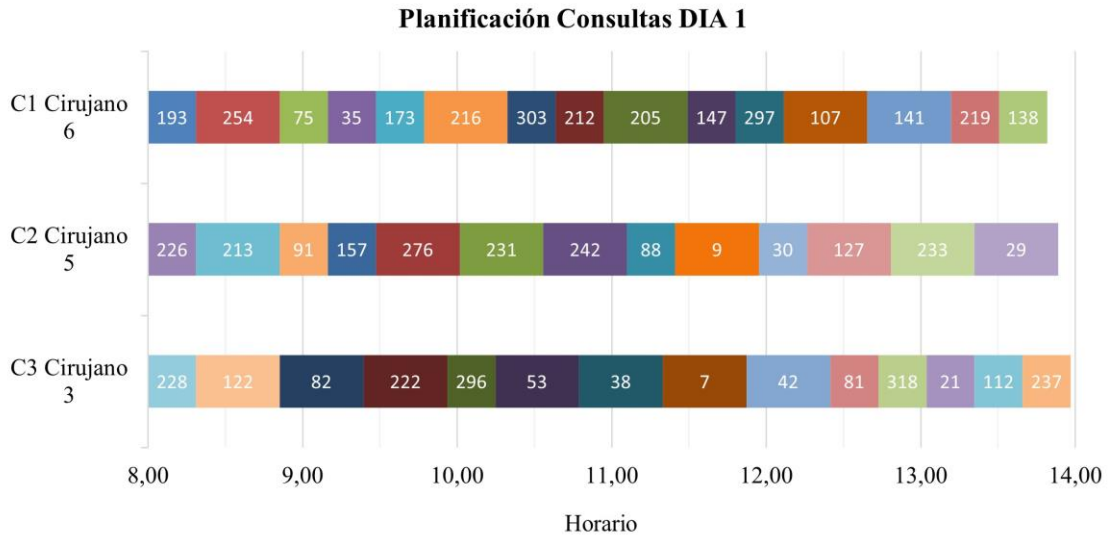


Figura 5-5*: Programación Consultas Día 1*

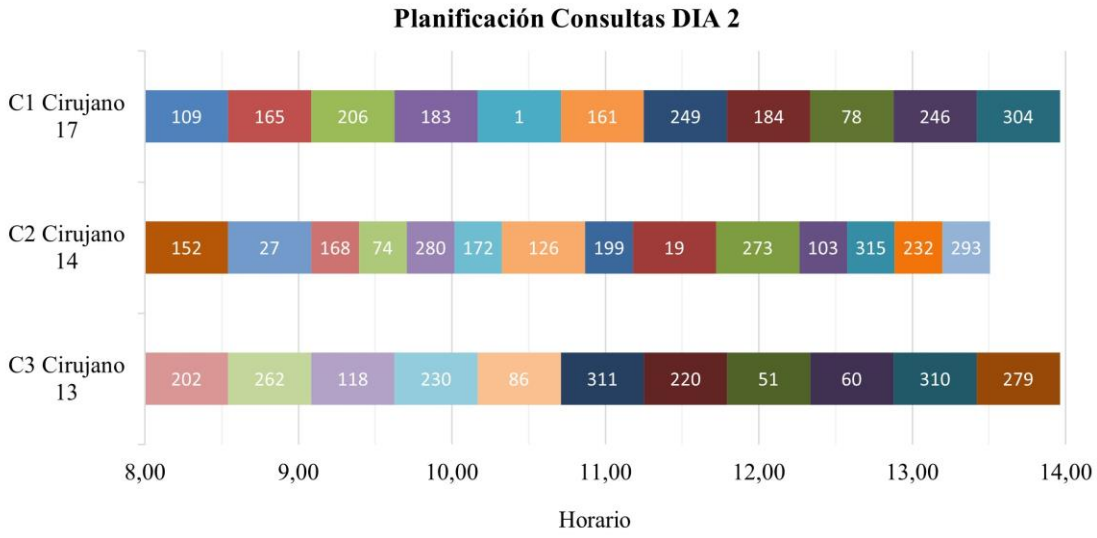


Figura 5-6*: Programación Consultas Día 2*

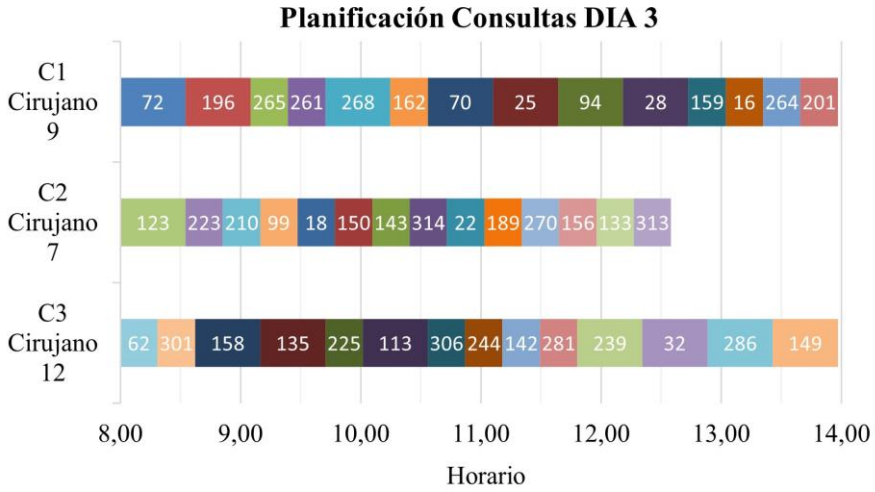


Figura 5-7: *Programación Consultas Día 3*

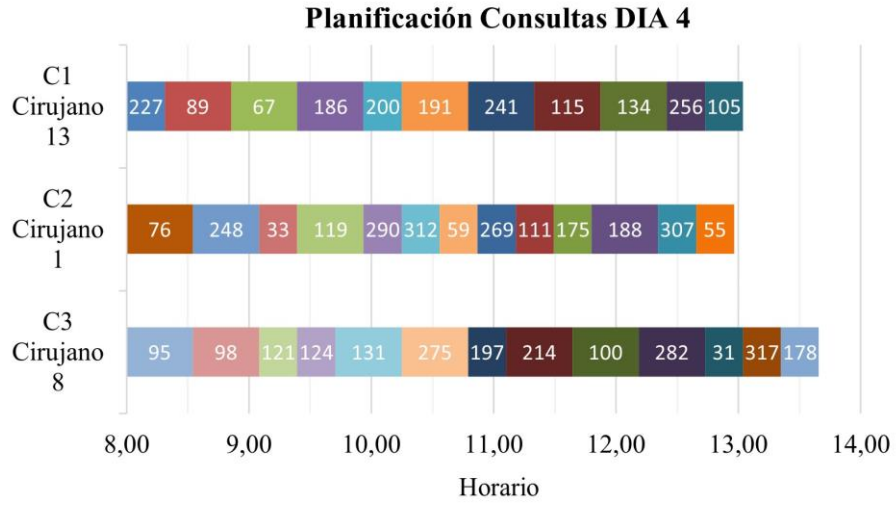


Figura 5-8: *Programación Consultas Día 4*

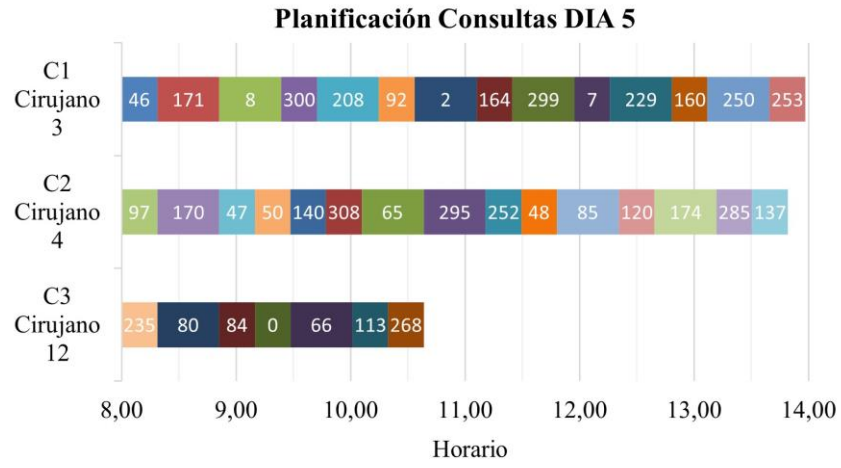


Figura 5-9*: Programación Consultas Día 5*

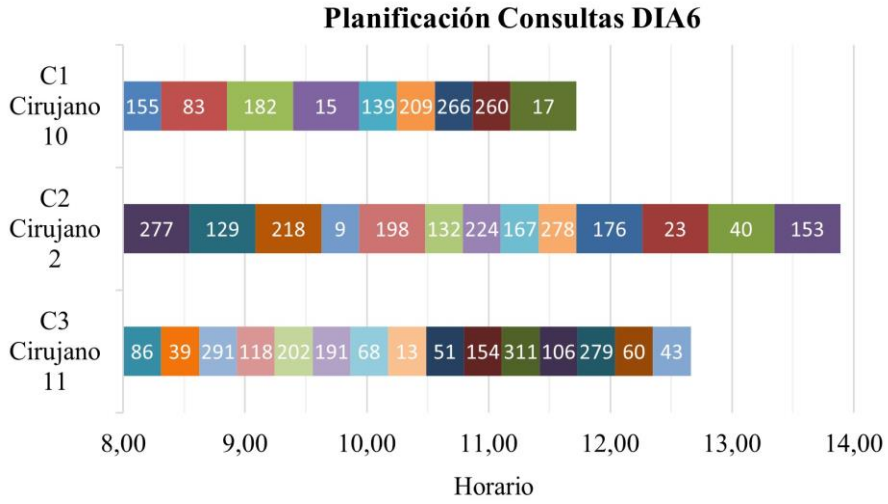


Figura 5-10: *Programación Consultas Día 6*

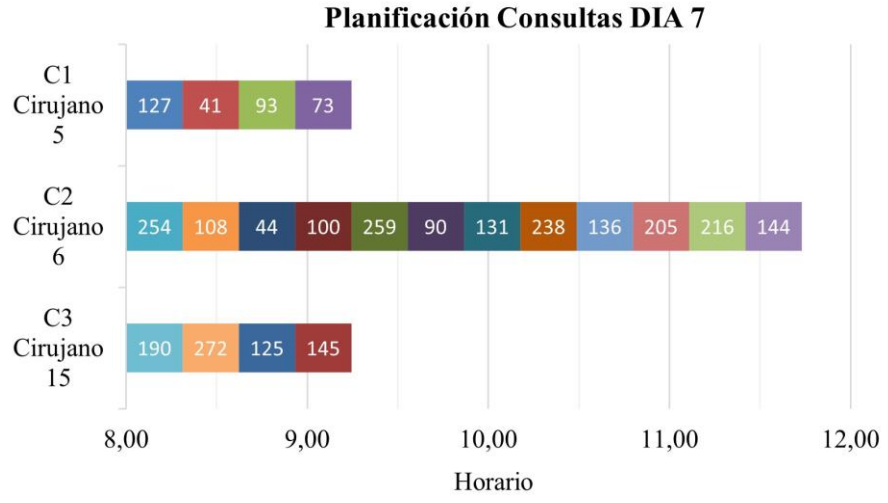


Figura 5-11*: Programación Consultas Día 7*

La programación de los quirófanos obtenida se muestra en Figura 5-12 a Figura 5-18. El horario de quirófanos se supone de 8 de la mañana hasta las 4 de la tarde.

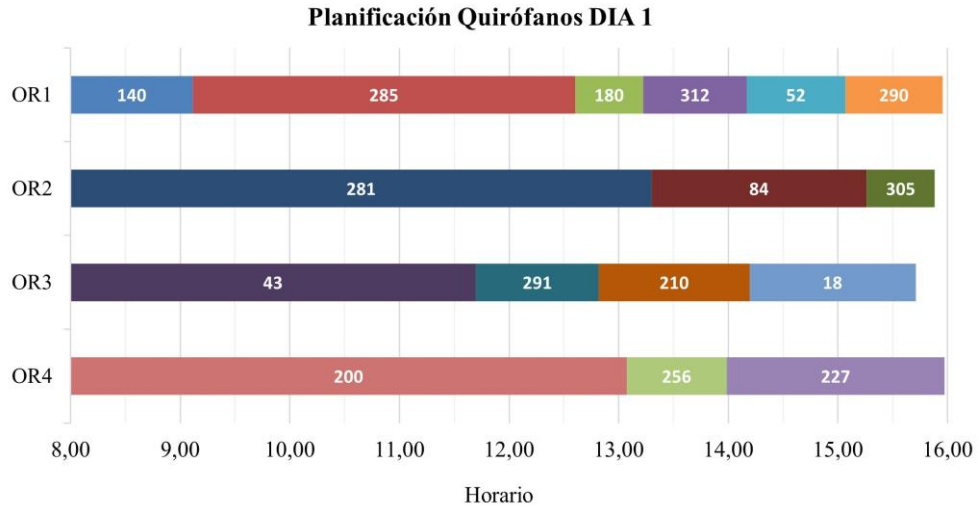


Figura 5-12: *Programación Quirófanos Día 1*

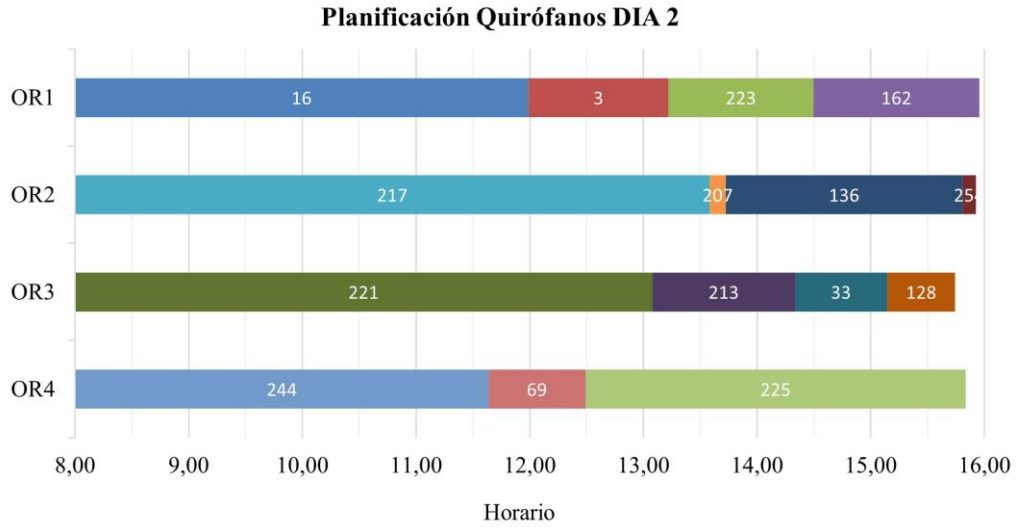


Figura 5-13: *Programación Quirófanos Día 2*

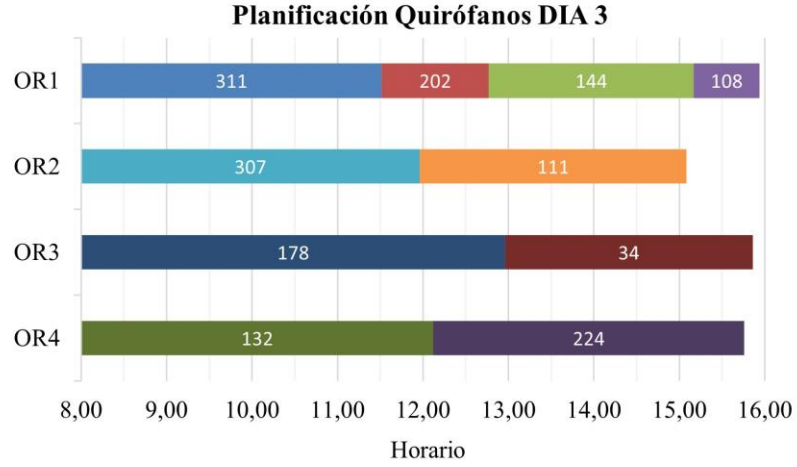


Figura 5-14: *Programación Quirófanos Día 3*

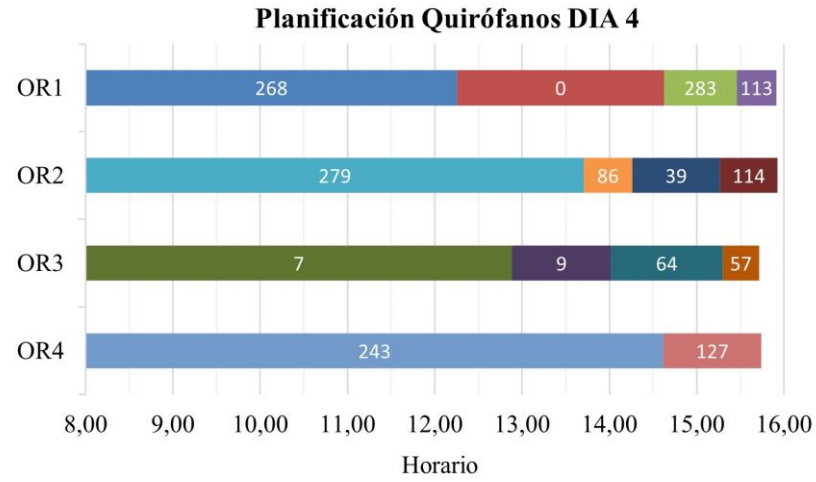


Figura 5-15*: Programación Quirófanos Día 4*

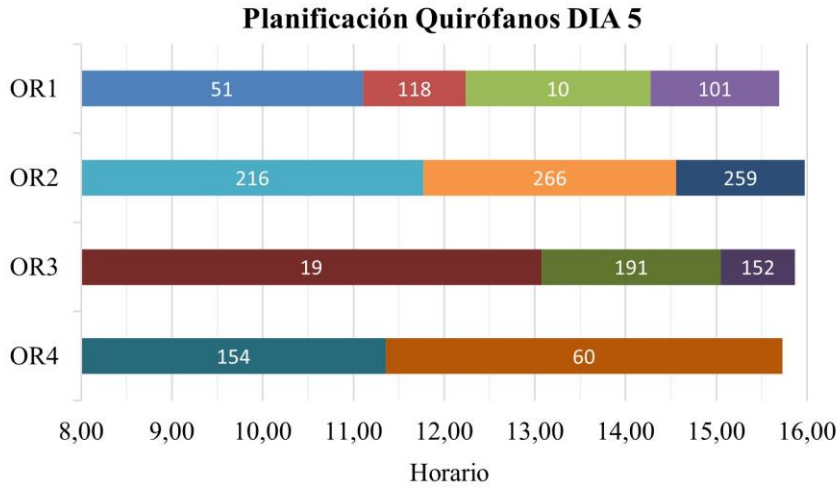


Figura 5-16*: Programación Quirófanos Día 5*

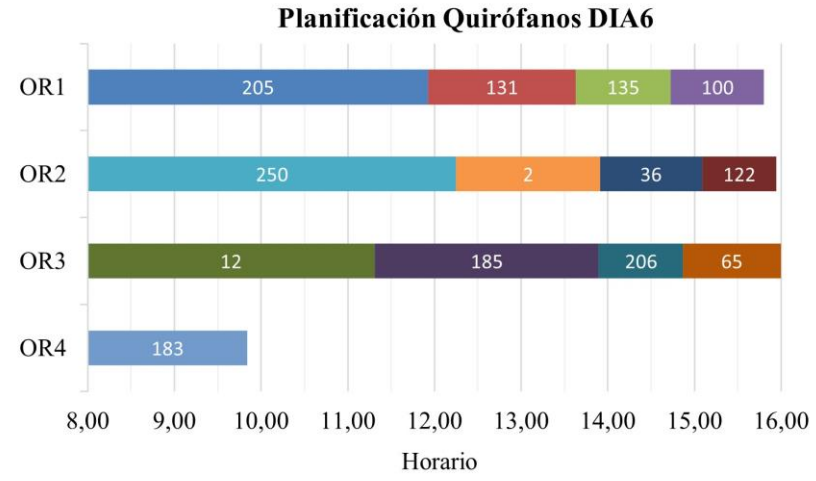


Figura 5-17*: Programación Quirófanos Día 6*

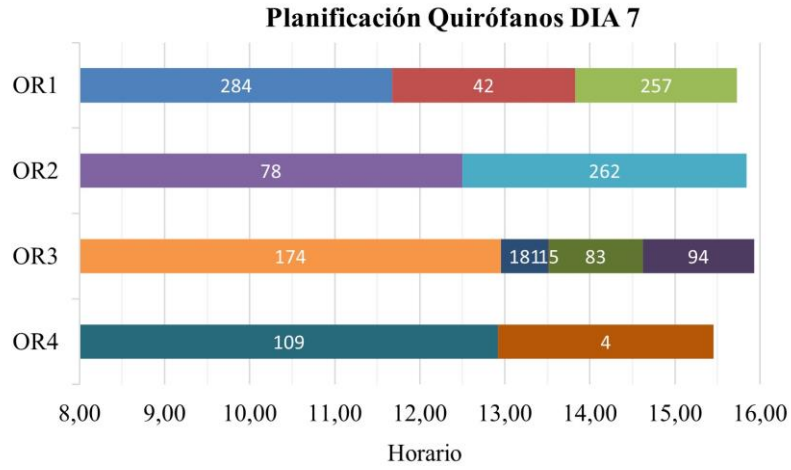


Figura 5-18 : *Programación Quirófanos Día 7*

A continuación, se procede a analizar la validez de los resultados. Siguiendo esta planificación, el número de pacientes restantes en la lista de espera tras una semana es de 176. Esto es, se ha completado el ciclo sanitario para el 45% de los pacientes en lista de espera y quedan restantes un 55%. El porcentaje de pacientes atendidos en al menos una etapa es del del 80%. 256 pacientes han sido atendidos en al menos una etapa durante esta semana de horizonte temporal. Aquellos que han avanzado dos etapas en esta semana son 50 pacientes, un 15,6% de la lista de espera total. Por último, se atiende en todas sus etapas, PAE, cirugía y consulta postoperatoria a 18 pacientes, un 5,6%. Son 64 los pacientes cuyas consultas y operaciones no se llevarán a cabo durante esta semana, un 20% de la lista de espera total. Ha de tenerse en cuenta que la lista de espera supera en un 100% la capacidad total del hospital, por ello quedan muchos pacientes sin atender.

Con respecto a la demora de los pacientes, al final de la semana habrá 21 pacientes en total que van con retraso. Ha de tenerse en cuenta que, al crear la instancia, muchos pacientes son ya establecidos como tarde, ya que se les asigna un tiempo acumulado en la lista de espera aleatorio. Se simula que dicha lista de espera proviene de unas semanas anteriores en las que también ha habido retrasos acumulados. De esta forma, el primer día del horizonte temporal, de 320 pacientes que hay en la lista de espera, 194 tienen tiempo acumulado mayor a su tiempo de respuesta, un 60%, un porcentaje muy alto. Es por ello por lo que el número de pacientes con demora es tan elevado. Este número se va compensando a lo largo de la semana hasta que en el último día de la semana el número de pacientes que va tarde es sólo 21.

Por otro lado, se analiza el uso que se hace de los recursos. Las consultas se utilizan en total un 80,1% de su capacidad, y los quirófanos un 94%. Es una buena utilización de los recursos, ya que se utilizan todas las consultas y quirófanos, y en gran parte de las ocasiones se aprovechan todas las horas disponibles. Se observa que, a pesar de haber muchos pacientes en lista de espera, no se utilizan los recursos a su máximo de capacidad, sobre todo en consultas. Esto es debido principalmente a que en las consultas sólo puede atender un cirujano por día. Muchos pacientes quedan excluidos de ser atendidos por no pertenecer a la unidad quirúrgica del cirujano pasando consulta. Por ello, si un cirujano es asignado a cierta consulta, únicamente los pacientes de su unidad podrán ser atendidos en ella. Así, si no hay muchos pacientes de la unidad pendientes de una consulta, quedará cierto tiempo de atención en consulta sin ser aprovechado. Si la lista de espera fuera más extensa, por lo general sí habría pacientes suficientes como para aprovechar todas las horas de consulta disponibles.

En cambio, los quirófanos, al poder ser utilizados por más de un equipo quirúrgico al día, son utilizados a mayor capacidad, casi completa, ya que se planifican las intervenciones por orden de la secuencia de pacientes, asignándole un hueco en el quirófano siempre que haya un grupo de cirujanos apto y libre. Los motivos por los que el paciente puede quedar sin ser operado tras una semana son, o bien porque aquellos cirujanos aptos para operar con la experiencia requerida tienen descanso y no pueden atender dicho día, bien porque los quirófanos ya han completado su capacidad y no hay hueco para un nuevo paciente, o bien porque

los posibles cirujanos aptos para la intervención se encuentran ya atendiendo en consultas y no pueden operar dicho día. En resumen, las horas en las que el quirófano se queda sin ser utilizado es principalmente debido a la falta de equipos quirúrgicos libres y aptos para la intervención, debido a los descansos o debido a que ya se encuentren asignados a consulta.

Con respecto al equilibrio de carga de trabajo entre cirujanos, la varianza de las horas acumuladas tanto en consulta como en quirófanos es de 7.54, o lo que es lo mismo, una desviación típica de 2.74 horas a lo largo de toda la semana. Este valor se iría compensando si el horizonte temporal fuera más tiempo ya que habría más margen para asignar cirujanos a consultas o quirófanos según deban equilibrar horas.

El tiempo de ejecución para el horizonte temporal de una semana ha sido de 24 horas, un tiempo muy alto debido a la cantidad de pacientes en la lista de espera, 320.

6 CONCLUSIONES

Este trabajo aborda un problema real del Hospital Puerta del Mar en Cádiz, el cual requiere la realización de un programa para la planificación del hospital. Este programa deberá, a partir de una lista de espera de pacientes, aportar como solución la planificación semanal del hospital. Esto es, una asignación de cirujanos a consultas y quirófanos, así como los pacientes que serán atendidos en cada consulta y quirófano.

Para ello, se han desarrollado seis metaheurísticas diferentes con el fin de compararlas entre ellas y escoger finalmente la que mejor soluciones aporte. Estas metaheurísticas se diferencian en las funciones que utilizan para obtener la solución, DS1 y DS2. Para evaluarlas, en primer lugar, se procesan las seis metaheurísticas para una batería de 36 instancias de diferente tamaño y un horizonte temporal de un día. Se observa que la heurística IG1-LS es aquella que en más ocasiones devuelve la mejor solución, y, además, de media, la que mejor solución ha aportado. Por tanto, esta heurística es la que se aporta como solución final para desarrollar la planificación del hospital para un horizonte temporal de una semana.

Obtenida la planificación para el horizonte temporal de una semana con la heurística IG1-LS, se obtienen los diagramas de Gantt de la semana, tanto para consultas como quirófanos, mostrado en las figuras del capítulo 5.

Una óptima planificación en los hospitales es un factor esencial para poder reducir las largas listas de espera por las que deben pasar los pacientes antes de ser atendidos. Es fundamental que se siga investigando en este campo para incrementar la eficiencia de funcionamiento de los hospitales, asegurando una calidad de atención a los ciudadanos para evitar que se vean obligados a esperar hasta un año para recibir atención médica, con el fin último de mejorar su calidad de vida.

7 LÍNEAS DE FUTURA INVESTIGACIÓN

Con vistas a futuras investigaciones sería interesante analizar la variación de resultados en caso de aplicar distintas formas de obtener una solución admisible, así como aplicar distintas políticas dentro del hospital.

En primer lugar, convendría realizar más experimentaciones con el fin de evaluar el impacto de aplicar distintos pesos a las funciones objetivo. En este proyecto se ha aplicado un peso de 1 para todas. Sin embargo, en el caso real, cada unidad quirúrgica aplicaría pesos distintos según sus prioridades. Sería interesante evaluar la aplicación de un mayor peso a la función objetivo Tardiness para comprobar en qué medida se disminuyen los retrasos, y cómo afecta eso a la utilización de los recursos.

Un posible cambio de política en el hospital sería aplicar *open-scheduling* en vez de *block-scheduling*. Como ya se explicó en el capítulo 2, el hospital se puede organizar siguiendo dos políticas. En primer lugar, y la forma más comúnmente aplicada en los hospitales, es la política *block-scheduling*. Esto es, cada cirujano únicamente puede ser asignado a una consulta al día y ningún otro cirujano puede atender en dicha consulta. Cada cirujano tiene su 'bloque' y no pueden atender pacientes fuera de éste. Esta política es ordenada, ya que evita los movimientos de cirujanos de consulta en consulta en el que pierden mucho tiempo y puede crear retraso, pero a la vez puede llevar a que un intervalo de tiempo disponible de consulta quede sin ser utilizado debido a que el cirujano no tiene más pacientes a los que atender dicho día.

Por otro lado, encontramos *open-scheduling*, en el que cada cirujano puede atender a pacientes independientemente de la consulta donde se encuentre en los distintos intervalos de tiempo. Al contrario que la política anterior, la aplicación de esta política al escenario real puede llegar a ser un tanto caótica debido al movimiento de cirujanos de consulta en consulta, pero sin embargo maximiza la utilización de los recursos, que es un gran beneficio al hablar de reducir listas de espera en el sistema sanitario.

El problema que se ha planteado en este trabajo aplica la política *block-scheduling*, por tanto, sería conveniente comparar los resultados obtenidos si se aplicara la política *open-scheduling*.

Otra posible variación de la solución sería cambiar la planificación de 'día a día' a 'semana en semana'. Es decir, actualmente la planificación se realiza día a día, y cada día se sigue el mismo proceso. El paciente es atendido, sube de etapa, y el siguiente día ya se encuentra de nuevo en la lista de espera y puede ser atendido. Sin embargo, se podría analizar cómo varía, a largo plazo, que la planificación se realizara semanalmente, sólo pudiendo ser atendido el paciente de semana en semana. Es decir, si el paciente es atendido en su consulta PAE el miércoles, hasta el lunes siguiente no podrá ser programada su intervención quirúrgica. ¿Se equilibrarían los tiempos de espera? ¿Se equilibrarían los retrasos de forma que en vez de tener un paciente que vaya muy tarde, tener a algunos que vayan con poco retraso?

Sería interesante comparar todos los resultados para finalmente aplicar en el hospital aquella política que mejores resultados aporte.

8 BIBLIOGRAFÍA

Canca Ortiz, J. D., & González Rodríguez, P. L. (2015). *Técnicas de optimización*. Iris-Copy.

Fei, H., Chu, C., & Meskens, N. (2008). Solving a tactical operating room planning problem by a column-generation-based heuristic procedure with four criteria. *Annals of Operations Research 2008 166:1*, *166*(1), 91–108. https://doi.org/10.1007/S10479-008-0413-3

Fei, H., Meskens, N., & Chu, C. (2010). A planning and scheduling problem for an operating theatre using an open scheduling strategy. *Computers & Industrial Engineering*, *58*(2), 221–230. https://doi.org/10.1016/J.CIE.2009.02.012

Framinan, J. M., Leisten, R., & García, R. R. (2014). Manufacturing scheduling systems: An integrated view on models, methods and tools. In *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools* (1st ed.). Springer-Verlag London Ltd. https://doi.org/10.1007/978-1-4471-6272-8

González de Diego, C. M., & Araúzo Araúzo, J. A. (2015). *Algoritmos constructivos para la programación de operaciones en entornos job shop flexibles* [Universidad de Valladolid]. https://uvadoc.uva.es/bitstream/handle/10324/18700/TFG-I-442.pdf;jsessionid=54B273251C05FE39CCA964C088AD888B?sequence=1

González-Busto, B., & García, R. (1999). Waiting lists in Spanish public hospitals: a system dynamics approach. *Systems Dynamics Review*. https://doi.org/10.1002/(SICI)1099-1727(199923)15:3<201::AID-SDR170>3.0.CO;2-5

Hospital Universitario Puerta del Mar | Servicio Andaluz de Salud. (2021, December). Junta de Andalucía. https://www.sspa.juntadeandalucia.es/servicioandaluzdesalud/ciudadania/derechos-y-garantias/tiempos-de-respuesta-asistencial-listas-de-espera/intervenciones-quirurgicas-junio-2021/hospital-universitario-puerta-del-mar

INFORME CCOO LISTAS DE ESPERA 2021. (n.d.). Retrieved June 13, 2022, from https://sanidad.ccoo.es/037dfb76e11046b743b4715991724bb3000057.pdf

Marí, R. P. . (2006). *De Euclides a Java: historia de los algoritmos y de los lenguajes de programación* (1st ed.). Nivola.

Melián, B., Moreno Pérez, J. A., Marcos Moreno Vega, J., Sánchez, F., & Cruz de Tenerife, S. (2003). Metaheuristics: A global view. *Revista Iberoamericana de Inteligencia Artificial. No*, *19*, 7–28. https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Metaheuristicas/Bibliography/metaheuristicas-vision-global.pdf

Molina-Pariente, J. M., Fernandez-Viagas, V., & Framinan, J. M. (2015). Integrated operating room planning and scheduling problem with assistant surgeon dependent surgery durations. *Computers and Industrial Engineering*, *82*, 8–20. https://doi.org/10.1016/J.CIE.2015.01.006

Molina-Pariente, J. M., Hans, E. W., Framinan, J. M., & Gomez-Cia, T. (2015). New heuristics for planning operating rooms. *Computers and Industrial Engineering*, *90*, 429–443. https://doi.org/10.1016/J.CIE.2015.10.002

Osman, I. H., & Kelly, J. P. (1996). Meta-Heuristics: An Overview. In *Meta-Heuristics*. Springer, Boston, MA. https://doi.org/10.1007/978-1-4613-1361-8_1

Pinedo, M. (2005). Planning and scheduling in manufacturing and services. In *Springer Series in Operations Research and Financial Engineering*. Springer Nature. https://doi.org/10.1007/B139030

¿Qué es Python? | Blog Becas Santander. (2021, April 9). https://www.becas-santander.com/es/blog/python-que-es.html

Ramírez Rojas, M. de los Á. (2020). *Integración de planificación de consultas y quirófanos en el sector sanitario* [Escuela Técnica Superior de Ingeniería]. https://idus.us.es/bitstream/handle/11441/102558/TFG-2982-RAMIREZ%20ROJAS%20.pdf?sequence=1&isAllowed=y

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033–2049. https://doi.org/10.1016/j.ejor.2005.12.009

9 ANEXO: CÓDIGO EN PYTHON

```
import pandas as pd
import numpy as np
import random
import math
import shutil
import time

def generarinstancia(numor,numc,alfa,beta):
    global descanso
    global S, P
    l=1 #num dias en horiz
    mds=4.9#por probabilidad debe ser 3/4 de la semana
    a=8 #num horas al  dia
    CT=numc*6*7+numor*8*7

    P=1
    pacientes=list(range(0,P))
    u=[[]]#unidad a la q pertenece
    estado=[[]]#estado ene lque se encuentra
    w=[[]]#peso en la lista de espera
    tiempoacum=[[]] #tiempo que lleva en ese estado
    dp=[[]] #duraciones
    dq=[[]]
    dc=[[]]
    trp=[[]] #tiempos de respuesta
    trq=[[]]
    trc=[[]]
    hp=[[]] #habilidad ciruj principal
    hs=[[]] #habilidad ciruj asistente
    cirujasoc=[[]]
    suma=0
    while len(u)<P-1 or len(u)==1:
        if P!=1:
            u.append([])#unidad a la q pertenece
            estado.append([])#estado ene lque se encuentra
            w.append([])#peso en la lista de espera
            tiempoacum.append([]) #tiempo que lleva en ese estado
            dp.append([]) #duraciones
            dq.append([])
            dc.append([])
            trp.append([]) #tiempos de respuesta
            trq.append([])
            trc.append([])
            hp.append([]) #habilidad ciruj principal
            hs.append([]) #habilidad ciruj asistente
            cirujasoc.append([])

        u[-1]=random.randint(0, 7)
        estado[-1]=random.randint(0,2)
```

```
            auxw=random.choices([random.randint(0,5),random.randint(10,15),random.randint(5,10)],
weights=(0.5,0.2,0.3),k=1)
            w[-1]=auxw[0]
            tiempoacum[-1]=random.randint(1,200)

            if len(u)==1:
               dp[-1]=random.uniform(0.30,0.60) #de media tarda 0.32 horas (8h para 25 consultas)
               dc[-1]=random.uniform(0.30,0.60)

            else:
               dp[-1]=dp[-2]
               dc[-1]=dc[-2]
            mu=random.randint(1,4)
            sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
            dq[-1]=abs(random.normalvariate(mu, sigma))
            hp[-1]=random.randint(1,20)
            hs[-1]=random.randint(0,20)

            if len(u)==1:
               trp[-1]=random.randint(28,42)
               trc[-1]=random.randint(28,42)
            else:
               trp[-1]=trp[-2]
               trc[-1]=trc[-2]

            if w[-1]>=10:
               trq[-1]=45
            elif w[-1]<=5:
               trq[-1]=360
            else:
               trq[-1]=180

            if estado[-1]==0:
               suma=suma+dp[-1]
            elif estado[-1]==1:
               suma=suma+dq[-1]
            else:
               suma=suma+dc[-1]
            durmedia=suma/len(dq)
            P=int((numc*6*beta+numor*8*beta)/durmedia) #número de pacientes admisibles

         pacientes=list(range(0,len(u)))

         #CIRUJANOS#
         #se crean los cirujanos en funcion de los pacientes que se han obtenido aleatoriamente.
         S=int(alfa*CT/mds/l/a)+1 #numero de cirujanos necesarios para cubrir todo
         cirujanos=list(range(0,S))
         exp=[[]]*S
         uc=[[]]

         sumatiempos=0
         unidad=0
         while unidad<8:
            j=0
            sumatiempos=0
            while j<len(pacientes):
               if u[j]==unidad:
                  if estado[j]==0:
```

```
                    sumatiempos=sumatiempos+dp[j] #la suma de lo que tardarian los pacientes
                elif estado[j]==1:
                    sumatiempos=sumatiempos+dq[j]
                else:
                    sumatiempos=sumatiempos+dc[j]
            j=j+1
        iteraciones=int(sumatiempos/5/8)+1
        it=0
        while it<iteraciones:
            if uc==[[]]:
                uc[0]=unidad
            else:
                uc.append([])
                uc[-1]=unidad
            it=it+1
        unidad=unidad+1
    while len(uc)<S:
        uc.append([])
        uc[-1]=random.randint(0,7)
    random.shuffle(uc)

    descanso=np.random.choice([0,1],(S,7),p=[0.7,0.3])

    #una vez tenemos la matriz de descansos, hay que modificarla para que todos los días haya mínimo un cirujano de cada unidad
    dia=0#bucle para comprobar que hay cirujanos de todas las unidades todos los días
    while dia<1:
        unidad=0
        flag=0
        while unidad<8:
            flag=0
            for c in cirujanos:
                if uc[c]==unidad and descanso[c,dia]==0:
                    flag=1 #ya hay un cirujano de esa unidad que no tiene descanso ese día
                    break
            if flag==0: #si no hay ningun ciruj de la unidad disponible, se elimina el descanso del primero que caiga
                for c in cirujanos:
                    if uc[c]==unidad:
                        descanso[c][dia]=0
                        break
            unidad=unidad+1
        dia=dia+1

    #experiencia de los cirujs#
    #comprobamos cuál es la máxima experiencia requerida por cada unidad:
    x=0
    unidad=0
    maximo=[0,0,0,0,0,0,0,0]
    while unidad<8:  #para cada unidad
        x=0
        while x<len(pacientes): #recogemos los pacientes que son de esa unidad
            if u[x]==unidad:
                if maximo[unidad]<hp[x]:
                    maximo[unidad]=hp[x]
            x=x+1
        unidad=unidad+1
```

```
#asignamos las experiencias a cada cirujano
unidad=0
while unidad<8:
   flag=0
   for x in cirujanos:
      if uc[x]==unidad:
         if flag==0:
            exp[x]=random.randint(maximo[unidad], 20)
            flag=1#se marca que se tiene ya cubierta con la maxima experiencia los pacientes de esa
unidad
         else:
            exp[x]=random.randint(1,20)
   unidad=unidad+1

k=0
#asignar cirujanos asociados a los pacientes según su unidad
while k<len(pacientes):
   delaunidad=[]
   if estado[k]==2:
      i=0
      while i<S:
         if uc[i]==u[k]:

            if delaunidad==[]:
               delaunidad=[i]
            else:
               delaunidad.append([])
               delaunidad[-1]=i
         i=i+1
      cirujasoc[k]=random.choice(delaunidad)
   else:
       cirujasoc[k]='NaN'
   k=k+1

dfpacientes = pd.DataFrame({'Número paciente':pacientes,
                            'dp':dp,
                            'trp':trp,
                            'dq':dq,
                            'trq':trq,
                            'u':u,
                            'hp':hp,
                            'hs':hs,
                            'dc':dc,
                            'trc':trc,
                            'w':w,
                            'estado':estado,
                            'cirujano':cirujasoc,
                            'tacum':tiempoacum
                            })
datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                            'G':exp,
                            'S':uc,
         })
datos = pd.DataFrame({ 'Número de quirófanos' : [numor],
                  'Número de consultas' : [numc]

   })
with pd.ExcelWriter('instancia.xlsx') as writer:
```

```
            dfpacientes.to_excel(writer, sheet_name = "Pacientes")
            datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
            datos.to_excel(writer, sheet_name = "Datos")

        def IG0():
            global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
            global listaciruj, listaasist, listapacs
            global sumaFO

            def insertar(lista,pos1,pos2):
                copia=lista.copy()
                a=copia.pop(pos1)
                copia.insert(pos2,a)
                return copia

            def mejora(v1probando,v2best):
                if v1probando[3]<v2best[3]:
                    return True
                else:
                    return False

            def nivelar(patients,surgeons):

                j=min(numc-1,len(surgeons)-1)
                while j>=0:
                    if patients[j]==[]:
                        surgeons.pop(j)
                        patients.pop(j)
                        surgeons.insert(numc-1,[])
                        patients.insert(numc-1,[])
                    j=j-1
                return surgeons

            def destruction(sequence):#la secuencia que se le pasará será secbest
                n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
                # print(n)
                a=0
                secborrar=sequence[:]
                while a<len(n):
                    secborrar.remove(n[a])
                    a=a+1
                return secborrar, n
            def construction(seqbest,n): #para secuenncia
                seqprobando=seqbest[:]
                a=0
                while a<len(n):
                    seqprobando=seqbest[:]
                    seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para luego ir tomando las que la mejoren
                    seqbest=seqprobando[:]
                    global hcc, hcor
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FObest=cajanegrasecuencia(seqbest)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()
```

```
                seqprobando.pop(0)

                b=1
                while b<len(seqbest):
                  seqprobando.insert(b, n[a])
                  hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                  hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                  FOprobando=cajanegrasecuencia(seqprobando)
                  hcc=hcccopia.copy()#para que las pruebas no alteren el real
                  hcor=hcorcopia.copy()

                  if mejora(FOprobando,FObest):
                     seqbest=seqprobando[:]
                     FObest=FOprobando[:]
                   #no se ha añadido ningún número a la secuencia y terminamos el bucle
                  seqprobando.pop(b)
                  b=b+1
                a=a+1
              return seqbest

          def leerexcel():
              global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado,
cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc
              datospacientes = pd.read_excel("data.xlsx", "Pacientes")
              pacientes = datospacientes['Número paciente'].tolist()
              dp = datospacientes['dp'].tolist()
              trp = datospacientes['trp'].tolist()
              dq = datospacientes['dq'].tolist()
              trq = datospacientes['trq'].tolist()
              u = datospacientes['u'].tolist()
              hp = datospacientes['hp'].tolist()
              hs = datospacientes['hs'].tolist()
              dc = datospacientes['dc'].tolist()
              trc = datospacientes['trc'].tolist()
              pesos = datospacientes['w'].tolist()
              estado = datospacientes['estado'].tolist()
              cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
              cirjasoc = [int(x) for x in cirjasoc1.tolist()]
              for x in range(len(cirjasoc)):
                 if cirjasoc[x]==999:
                    cirjasoc[x]='NaN'
              tiempoacum = datospacientes['tacum'].tolist()

          def cajanegrasecuencia(secuencia):
             global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
              global listaciruj, listaasist, listapacs
              def asignacion(secuencia):
                global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor,
ctconsultas, ctor, hcor, hcc
              pacientesconsultas = [[]]
              cirujanosconsultas=[[]]
              x=0
              while x<numc-1:
                 pacientesconsultas.append([])
                 cirujanosconsultas.append([])
                 x=x+1
```

```
pacientesor = [[]]
cirujanosor = [[]]
asistentesor = [[]]
x=0
while x<numor-1:
   pacientesor.append([])
   cirujanosor.append([])
   asistentesor.append([])
   x=x+1


###FUNCIONES###

#Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
      #sirve para medir el equilibrio de la carga de trabajo#
def equilibrio(hcc,hcor):
   return (hcc.var()+hcor.var())

#Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
def tardy(pacientes):
   global pesos
   x=0
   late=[]
   for x in range(len(pacientes)):
      if estado[x]==0:
         if tiempoacum[x]>trp[x]:
            late.append((tiempoacum[x]-trp[x])*pesos[x])
      if estado[x]==1:
         if tiempoacum[x]>trq[x]:
            late.append((tiempoacum[x]-trq[x])*pesos[x])
      if estado[x]==2:
         if tiempoacum[x]>trc[x]:
            late.append((tiempoacum[x]-trc[x])*pesos[x])
   return sum(late) #late indica cuántos pacientes van con retraso actualmente

#Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
def asignadodia(cirujano,consultaor):
   global cirujanosconsultas
   global cirujanosor
   global numc
   i=0
   flag = 0
   if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
      for i in range(len(cirujanosconsultas)):
         if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
            return i
            flag=1
      if flag==0:
         return 'NaN'
   elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
      for i in range(numor):
         if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
            return i
            flag=1
      if flag==0:
```

```
                return 'NaN'

        #funcion para calcular el indice de un vector
        def indice(vector,valor):
            x=0
            for x in range(len(vector)):
                if vector[x]==valor:
                    return x
                    break

        #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
        def busquedaPAE (paciente):
            global cirujanos
            global numc
            global hcc
            global cirujanosconsultas
            j=0
            cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
            while j<len(cirujanos):
                if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad
                    cirujanosdelaunidad.append(j)
                j=j+1
            #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosdelaunidad)):
                horas.append(hcc[cirujanosdelaunidad[j]])
            horasnumpy=np.array(horas)
            cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
            np.argsort(horasnumpy)
            cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
            #lista de cirujanos ordenadas según horas acumuladas en consulta
            j=0
            flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
            #para cada cirujano, miramos si está asignado a alguna consulta el dia
            while j<len(cirujanosdelaunidad):

                asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                        return [asignadoenconsulta,cirujanosdelaunidad[j]]
                        flagyatieneconsulta=1
                        break

                if flagyatieneconsulta==1:break
                j=j+1
            if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero la primera consulta libre
                # print('el cirujano no tiene consulta asignada en toda la semana, así que le buscamos la primera consulta libre')
                j=0

                while j<len(cirujanosconsultas):
```

```
                    if cirujanosconsultas[j]==[]:#si la consulta está libre
                        x=0
                        for x in cirujanosdelaunidad:
                            if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or asignadodia(x, 'consulta', dia)==j):
                                #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa misma,
                                return [j,x]
                                flagyatieneconsulta=1
                                break#cuando se le asigna, se sale del for
                        if flagyatieneconsulta==1:
                            break
                        j=j+1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


            # Función para asignar un paciente y un cirujano a consulta postop
            def busquedaPOSTOP (paciente):
                #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                flagyatieneconsulta=0

                asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                        return [asignadoenconsulta,cirjasoc[paciente]]
                        flagyatieneconsulta=1
                if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                    #se le asigna el primer hueco que haya en la semana
                    j=0

                    while j<len(cirujanosconsultas):
                        if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                            if asignadoenconsulta=='NaN' and asignadodia(cirjasoc[paciente], 'OR')=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                                return [j,cirjasoc[paciente]]
                                flagyatieneconsulta=1
                                break#cuando se le asigna, se sale del for
                        if flagyatieneconsulta==1:
                            break
                        j=j+1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana



            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                    #la experiencia y nivel de dificultad de la  operación.
                j=0
```

```
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                  if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                    cirujanosexpunidad.append(cirujanos[j])
                  j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                  horas.append(hcor[cirujanosexpunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente


                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                  if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                    asistexp.append(cirujanos[j])
                  j=j+1
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                  horas.append(hcor[asistexp[j]])
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                  j=0

                  flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                  #para cada cirujano, miramos si está asignado a algún OR el dia
                  while j<len(cirujanosexpunidad):
                    asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                    if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                      if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                        return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                        flagyatieneOR=1
                        break
                    j=j+1
                  if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                    j=0
```

```
                    while j<numor:
                        if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                            x=0
                            for x in cirujanosexpunidad:
                                if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                    return [j,cirujanosexpunidad[0],'NaN']
                                    flagyatieneOR=1
                                    break#cuando se le asigna, se sale del for
                            if flagyatieneOR==1:
                                break
                            j=j+1
                    if flagyatieneOR==0:
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                else:   #sí se necesita un asistente
                    #vamos a buscar a la vez un cirujano principal y un asistente
                    i=0
                    flagyatieneasist=0
                    for i in cirujanosexpunidad:
                        asignadoenOR = asignadodia(i,'OR')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                #buscamos un asistente
                                k=0
                                for k in asistexp:
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        return [asignadoenOR,i,k]
                                        flagyatieneasist=1
                                        break
                            if flagyatieneasist==1:break
                    if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                        #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                        i=0
                        for i in cirujanosexpunidad:
                            asignadoenconsulta = asignadodia(i,'consulta')
                            x=0
                            while x<numor:
                                if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                    if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                    #buscamos un asistente
                                        for k in asistexp:
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
                                                return [x,i,k]
                                                flagyatieneasist=1
                                                break
                                    if flagyatieneasist==1:break
```

```
                        x=x+1
                    if flagyatieneasist==1:break
                if flagyatieneasist==0:
                    return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                    #significa que no hay hueco posible para este paciente en OR esta semana


            global pacientesaborrar
            pacientesaborrar=[]
            atendidos=0
            for k in pacientes:
                if k not in secuencia:
                    tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
            for k in range(len(secuencia)): #x es el indice del paciente
                if estado[indice(pacientes,secuencia[k])]==0:
                    retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                    if retorno!='NaN':
                        cirujanosconsultas[retorno[0]]=[retorno[1]]
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                        hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                        tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1 día
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

            tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==1:
                    retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                    if retorno!='NaN':
                        cirujanosor[retorno[0]].append(retorno[1])
                        hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                        pacientesor[retorno[0]].append(secuencia[k])
                        ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                        cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        if retorno[2]!='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                            hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                        if retorno[2]=='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

            tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==2:
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal]
                    if retorno!='NaN':
                        cirujanosconsultas[retorno[0]]=[retorno[1]]
                        pacientesconsultas[retorno[0]].append(secuencia[k])
```

```
                    ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                    #habría que eliminar al paciente de la lista
                    hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                    flagcambiado=1
                    atendidos=atendidos+1
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                    pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))
            eq=equilibrio(hcc,hcor)
            tarde=tardy(pacientes)

            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
                i=i+1

            i=0
            while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
                i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return  eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


        #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
        leerexcel()
        ctconsultas = [[0]]
        x=0
        while x<numc-1:
            ctconsultas.append([0])
            x=x+1

        ctor = [[0]]
        x=0
        while x<numor-1:
            ctor.append([0])
            x=x+1
```

```
    global dia
    leerexcel()
    vectorFO=asignacion(secuencia)

    listaciruj[dia] = cirujanosconsultas+cirujanosor
    listaasist[dia] = empty+asistentesor
    listapacs[dia] = pacientesconsultas+pacientesor

    return vectorFO

def actualizarexcel(pacientesaborrar):
    #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
    j=0
    x=0
    while j < len(pacientesaborrar):
        x=0
        while x<len(pacientes):
            if pacientesaborrar[j]==pacientes[x]:
                del pacientes[x]
                del cirjasoc[x]
                del dc[x]
                del dp[x]
                del dq[x]
                del estado[x]
                del hp[x]
                del hs[x]
                del pesos[x]
                del tiempoacum[x]
                del trc[x]
                del trp[x]
                del trq[x]
                del u[x]
            else:x=x+1
        j=j+1

    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                                'dp':dp,
                                'trp':trp,
                                'dq':dq,
                                'trq':trq,
                                'u':u,
                                'hp':hp,
                                'hs':hs,
                                'dc':dc,
                                'trc':trc,
                                'w':pesos,
                                'estado':estado,
                                'cirujano':cirjasoc,
                                'tacum':tiempoacum
                                })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                                'G':exp,
                                'S':uc
        })
    with pd.ExcelWriter('data.xlsx') as writer:
        nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
        datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
```

```
        datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):
    c=0
    C=[]
    while c<numc:
        C.append('C')
        c=c+1
    o=0
    O=[]
    while o<numor:
        O.append('O')
        o=o+1
    dfsol1 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[0],
                           'ASISTENTES':listaasist[0],
                           'PACIENTES':listapacs[0]

        })
    dfsol2 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[1],
                           'ASISTENTES':listaasist[1],
                           'PACIENTES':listapacs[1]

        })
    dfsol3 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[2],
                           'ASISTENTES':listaasist[2],
                           'PACIENTES':listapacs[2]

        })
    dfsol4 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[3],
                           'ASISTENTES':listaasist[3],
                           'PACIENTES':listapacs[3]

        })
    dfsol5 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[4],
                           'ASISTENTES':listaasist[4],
                           'PACIENTES':listapacs[4]

        })
    dfsol6 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[5],
                           'ASISTENTES':listaasist[5],
                           'PACIENTES':listapacs[5]

        })
    dfsol7 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[6],
                           'ASISTENTES':listaasist[6],
                           'PACIENTES':listapacs[6]

        })

    with pd.ExcelWriter('solucion.xlsx') as writer:
        dfsol1.to_excel(writer, sheet_name = "LUNES")
        dfsol2.to_excel(writer, sheet_name = "MARTES")
```

```
        dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
        dfsol4.to_excel(writer, sheet_name = "JUEVES")
        dfsol5.to_excel(writer, sheet_name = "VIERNES")
        dfsol6.to_excel(writer, sheet_name = "SABADO")
        dfsol7.to_excel(writer, sheet_name = "DOMINGO")

#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")
numor = datos.loc[0,'Número de quirófanos']
numc = datos.loc[0,'Número de consultas']
#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
   aux.append([])
   x=x+1
x=0
listaciruj=[]
listaasist=[]
listapacs=[]
while x<7:
   listaciruj.append(aux)
   listaasist.append(aux)
   listapacs.append(aux)
   x=x+1

empty =[[]]
x=0
while x<numc-1:
   empty.append([])
   x=x+1

dia=0
while dia<1:

   #NEH#
   #1- ORDENAR LA SECUENCIA INICIAL
   sumatiempos=np.array([0.0]*len(pacientes))
   for x in range(len(pacientes)):
      sumatiempos[x]=dc[x]+dq[x]+dp[x]
   pacientesnumpy = np.array(pacientes)
   ordennumpy = np.array(sumatiempos)
   np.argsort(sumatiempos)
   secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
   secinicial1=np.flip(secinicial0)
   secinicial=secinicial1.tolist()
```

```
        #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
        #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
        pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpiinicial=cajanegrasecuencia(piinicial)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
        print('FOpiinicial antes de NEH', FOpiinicial)
        j=1
        while j<len(secinicial):

          i=0
          secprobando=pi[:]
          flagcambiado=0
          while i<=j:
            secprobando.insert(i, secinicial[j])
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegrasecuencia(secprobando)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOprobando,FOpi):
              pi=secprobando[:]
              hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
              hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
              FOpi=cajanegrasecuencia(pi)
              hcc=hcccopia.copy()#para que las pruebas no alteren el real
              hcor=hcorcopia.copy()

              flagcambiado=1
             #no se ha añadido ningún número a la secuencia y terminamos el bucle
            secprobando.pop(i)
            i=i+1
          if flagcambiado==0:
            secprobando.insert(i-1,secinicial[j])
            pi=secprobando[:]
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
          j=j+1
          #fin del NEH. Tenemos FOpi y pi

        if mejora(FOpi,FOpiinicial):
          pibest=pi.copy()
          hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
```

```
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(pibest)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
else:
    pibest=piinicial.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(piinicial)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

print('FOpibest tras NEH', FOpibest)

it=0
Temp=1
inicio=time.time()
while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
    piprima=pi.copy()
    destruida=destruction(piprima)#destruction phase
    piprima=construction(destruida[0],destruida[1]) #construction phase
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpiprima=cajanegrasecuencia(piprima)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
    if mejora(FOpiprima,FOpi):
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
        if mejora(FOpi,FOpibest):
            pibest=pi.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegrasecuencia(pibest)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
    elif random.uniform(0, 1)<=-math.exp(-(FOpiprima[1]-FOpi[1])/Temp):
        print('se acepta peor solucion')
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    Temp=0.95*Temp
    it=it+1

FOpibest=cajanegrasecuencia(pibest)

print('MEJOR FO DEFINITIVA',FOpibest)
sumaFO=sumaFO+FOpibest[3]
actualizarexcel(pacientesaborrar)
dia=dia+1
```

```
        escribirsol(listaciruj, listaasist, listapacs)

        return sumaFO

    def IG0LS():

        global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        global sumaFO

        def insertar(lista,pos1,pos2):
            copia=lista.copy()
            a=copia.pop(pos1)
            copia.insert(pos2,a)
            return copia

        def mejora(v1probando,v2best):
            if v1probando[3]<v2best[3]:
                return True
            else:
                return False

        def nivelar(patients,surgeons):

            j=min(numc-1,len(surgeons)-1)
            while j>=0:
                if patients[j]==[]:
                    surgeons.pop(j)
                    patients.pop(j)
                    surgeons.insert(numc-1,[])
                    patients.insert(numc-1,[])
                j=j-1
            return surgeons

        def destruction(sequence):#la secuencia que se le pasará será secbest
            n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
            a=0
            secborrar=sequence[:]
            while a<len(n):
                secborrar.remove(n[a])
                a=a+1
            return secborrar, n
        def construction(seqbest,n): #para secuenncia
            seqprobando=seqbest[:]
            a=0
            while a<len(n):
                seqprobando=seqbest[:]
                seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para luego ir tomando las que la mejoren
                seqbest=seqprobando[:]
                global hcc, hcor
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FObest=cajanegrasecuencia(seqbest)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
```

```
                seqprobando.pop(0)
                ###PROBANDO PARA NO INSERTAR DOS VECES EL MISMO NOMBRE###
                b=1
                while b<len(seqbest):
                  seqprobando.insert(b, n[a])
                  hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                  hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                  FOprobando=cajanegrasecuencia(seqprobando)
                  hcc=hcccopia.copy()#para que las pruebas no alteren el real
                  hcor=hcorcopia.copy()

                  if mejora(FOprobando,FObest):
                     seqbest=seqprobando[:]
                     FObest=FOprobando[:]
                   #no se ha añadido ningún número a la secuencia y terminamos el bucle
                  seqprobando.pop(b)
                  b=b+1
                a=a+1
              return seqbest

          def leerexcel():
              global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado,
cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc

                datospacientes = pd.read_excel("data.xlsx", "Pacientes")
                pacientes = datospacientes['Número paciente'].tolist()
                dp = datospacientes['dp'].tolist()
                trp = datospacientes['trp'].tolist()
                dq = datospacientes['dq'].tolist()
                trq = datospacientes['trq'].tolist()
                u = datospacientes['u'].tolist()
                hp = datospacientes['hp'].tolist()
                hs = datospacientes['hs'].tolist()
                dc = datospacientes['dc'].tolist()
                trc = datospacientes['trc'].tolist()
                pesos = datospacientes['w'].tolist()
                estado = datospacientes['estado'].tolist()
                cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
                cirjasoc = [int(x) for x in cirjasoc1.tolist()]
                for x in range(len(cirjasoc)):
                   if cirjasoc[x]==999:
                      cirjasoc[x]='NaN'

                tiempoacum = datospacientes['tacum'].tolist()

          def cajanegrasecuencia(secuencia):
             global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
             global listaciruj, listaasist, listapacs
             def asignacion(secuencia):
                global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor,
ctconsultas, ctor, hcor, hcc
                pacientesconsultas = [[]]
                cirujanosconsultas=[[]]
                x=0
                while x<numc-1:
                   pacientesconsultas.append([])
                   cirujanosconsultas.append([])
```

```
    x=x+1

pacientesor = [[]]
cirujanosor = [[]]
asistentesor = [[]]
x=0
while x<numor-1:
    pacientesor.append([])
    cirujanosor.append([])
    asistentesor.append([])
    x=x+1

###FUNCIONES###

#Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
        #sirve para medir el equilibrio de la carga de trabajo#
def equilibrio(hcc,hcor):
    return (hcc.var()+hcor.var())

#Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
def lateness(pacientes):
    global pesos
    x=0
    late=[]
    for x in range(len(pacientes)):
        if estado[x]==0:
            if tiempoacum[x]>trp[x]:
                late.append((tiempoacum[x]-trp[x])*pesos[x])
        if estado[x]==1:
            if tiempoacum[x]>trq[x]:
                late.append((tiempoacum[x]-trq[x])*pesos[x])
        if estado[x]==2:
            if tiempoacum[x]>trc[x]:
                late.append((tiempoacum[x]-trc[x])*pesos[x])
    return sum(late) #late indica cuántos pacientes van con retraso actualmente

#Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
def asignadodia(cirujano,consultaor):
    global cirujanosconsultas
    global cirujanosor
    global numc
    i=0
    flag = 0
    if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
        for i in range(len(cirujanosconsultas)):
            if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                return i
                flag=1
        if flag==0:
            return 'NaN'
    elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
        for i in range(numor):
            if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
                return i
                flag=1
```

```
            if flag==0:
                return 'NaN'

        #funcion para calcular el indice de un vector
        def indice(vector,valor):
            x=0
            for x in range(len(vector)):
                if vector[x]==valor:
                    return x
                    break

        #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
        def busquedaPAE (paciente):
            global cirujanos
            global numc
            global hcc
            j=0
            cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
            while j<len(cirujanos):
                if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad
                    cirujanosdelaunidad.append(j)
                j=j+1
            #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosdelaunidad)):
                horas.append(hcc[cirujanosdelaunidad[j]])
            horasnumpy=np.array(horas)
            cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
            np.argsort(horasnumpy)
            cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
            j=0
            flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
            #para cada cirujano, miramos si está asignado a alguna consulta el dia
            while j<len(cirujanosdelaunidad):

                asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                        return [asignadoenconsulta,cirujanosdelaunidad[j]]
                        flagyatieneconsulta=1
                        break

                if flagyatieneconsulta==1:break
                j=j+1
            if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero la primera consulta libre
                j=0

                while j<len(cirujanosconsultas):
                    if cirujanosconsultas[j]==[]:#si la consulta está libre
                        x=0
                        for x in cirujanosdelaunidad:
```

```
                        if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or asignadodia(x, 'consulta', dia)==j):
                            #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa misma,
                            return [j,x]
                            flagyatieneconsulta=1
                            break#cuando se le asigna, se sale del for
                        if flagyatieneconsulta==1:
                            break
                    j=j+1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


            # Función para asignar un paciente y un cirujano a consulta postop
            def busquedaPOSTOP (paciente):
                #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                flagyatieneconsulta=0
                asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                        return [asignadoenconsulta,cirjasoc[paciente]]
                        flagyatieneconsulta=1
                if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                    #se le asigna el primer hueco que haya en la semana
                    j=0

                    while j<len(cirujanosconsultas):
                        if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                            if asignadoenconsulta=='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                                return [j,cirjasoc[paciente]]
                                flagyatieneconsulta=1
                                break#cuando se le asigna, se sale del for
                        if flagyatieneconsulta==1:
                            break
                        j=j+1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                #la experiencia y nivel de dificultad de la  operación.
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
```

```
                    j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    horas.append(hcor[cirujanosexpunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                 #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                    j=j+1
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    horas.append(hcor[asistexp[j]])
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()

                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                        asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                                return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                                flagyatieneOR=1
                                break
                        j=j+1
                    if flagyatieneOR==0:#si no tiene OR en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                        j=0
                        while j<numor:
                            if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                                x=0
                                for x in cirujanosexpunidad:
                                    if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                        return [j,cirujanosexpunidad[0],'NaN']
                                        flagyatieneOR=1
```

```
                        break#cuando se le asigna, se sale del for
                    if flagyatieneOR==1:
                        break
                j=j+1
            if flagyatieneOR==0:
                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
        else:   #sí se necesita un asistente
            #vamos a buscar a la vez un cirujano principal y un asistente
            i=0
            flagyatieneasist=0
            for i in cirujanosexpunidad:
                asignadoenOR = asignadodia(i,'OR')
                if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                    if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                        #buscamos un asistente
                        k=0
                        for k in asistexp:
                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                #si esta condicion se cumple, se asocia este asist
                                return [asignadoenOR,i,k]
                                flagyatieneasist=1
                                break
                    if flagyatieneasist==1:break
            if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                i=0
                for i in cirujanosexpunidad:
                    asignadoenconsulta = asignadodia(i,'consulta')
                    x=0
                    while x<numor:
                        if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                            if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                            #buscamos un asistente
                                for k in asistexp:
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        return [x,i,k]
                                        flagyatieneasist=1
                                        break
                            if flagyatieneasist==1:break
                            x=x+1
                        if flagyatieneasist==1:break
                if flagyatieneasist==0:
                    return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                    #significa que no hay hueco posible para este paciente en OR esta semana

###COMIENZO CODIGO###
```

```
                global pacientesaborrar
                atendidos=0
                pacientesaborrar=[]
                for k in pacientes:
                  if k not in secuencia:
                    tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
                for k in range(len(secuencia)): #x es el indice del paciente
                  if estado[indice(pacientes,secuencia[k])]==0:
                    retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                    if retorno!='NaN':
                      cirujanosconsultas[retorno[0]]=[retorno[1]]
                      pacientesconsultas[retorno[0]].append(secuencia[k])
                      ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dp[indice(pacientes,secuencia[k])])
                      hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                      tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1
día
                      flagcambiado=1
                      atendidos=atendidos+1
                    else:
                      flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                  if estado[indice(pacientes,secuencia[k])]==1:
                    retorno=busquedaOR(indice(pacientes,secuencia[k]))  #me  devuelve
[consulta,cirujprincipal,asistente]
                    if retorno!='NaN':
                      cirujanosor[retorno[0]].append(retorno[1])
                      hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                      pacientesor[retorno[0]].append(secuencia[k])
                      ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                      cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                      tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                      if retorno[2]!='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                        hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                      if retorno[2]=='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                      flagcambiado=1
                      atendidos=atendidos+1
                    else:
                      flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                  if estado[indice(pacientes,secuencia[k])]==2:
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k]))  #me  devuelve
[consulta,cirujprincipal]
                    if retorno!='NaN':
                      cirujanosconsultas[retorno[0]]=[retorno[1]]
                      pacientesconsultas[retorno[0]].append(secuencia[k])
                      ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                      #habría que eliminar al paciente de la lista
                      hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                      flagcambiado=1
                      atendidos=atendidos+1
                      tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
```

```
                    # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el hospital#
                    pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)
            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
                i=i+1

            i=0
            while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
                i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-utilizacion

        #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente, paciente', otra para consultas y OR
            #creo las listas#
            leerexcel()
            ctconsultas = [[0]]
            x=0
            while x<numc-1:
                ctconsultas.append([0])
                x=x+1

            ctor = [[0]]
            x=0
            while x<numor-1:
                ctor.append([0])
                x=x+1

            global dia
            leerexcel()
            vectorFO=asignacion(secuencia)

            listaciruj[dia] = cirujanosconsultas+cirujanosor
            listaasist[dia] = empty+asistentesor
            listapacs[dia] = pacientesconsultas+pacientesor
```

```
    return vectorFO

def actualizarexcel(pacientesaborrar):
    #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
    j=0
    x=0
    while j < len(pacientesaborrar):
        x=0
        while x<len(pacientes):
            if pacientesaborrar[j]==pacientes[x]:
                del pacientes[x]
                del cirjasoc[x]
                del dc[x]
                del dp[x]
                del dq[x]
                del estado[x]
                del hp[x]
                del hs[x]
                del pesos[x]
                del tiempoacum[x]
                del trc[x]
                del trp[x]
                del trq[x]
                del u[x]
            else:x=x+1
        j=j+1

    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                                'dp':dp,
                                'trp':trp,
                                'dq':dq,
                                'trq':trq,
                                'u':u,
                                'hp':hp,
                                'hs':hs,
                                'dc':dc,
                                'trc':trc,
                                'w':pesos,
                                'estado':estado,
                                'cirujano':cirjasoc,
                                'tacum':tiempoacum
                                })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                                'G':exp,
                                'S':uc
        })
    with pd.ExcelWriter('data.xlsx') as writer:
        nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
        datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
        datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):

    c=0
    C=[]
    while c<numc:
        C.append('C')
```

```
   c=c+1
o=0
O=[]
while o<numor:
   O.append('O')
   o=o+1
dfsol1 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[0],
                  'ASISTENTES':listaasist[0],
                  'PACIENTES':listapacs[0]

   })
dfsol2 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[1],
                  'ASISTENTES':listaasist[1],
                  'PACIENTES':listapacs[1]

   })
dfsol3 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[2],
                  'ASISTENTES':listaasist[2],
                  'PACIENTES':listapacs[2]

   })
dfsol4 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[3],
                  'ASISTENTES':listaasist[3],
                  'PACIENTES':listapacs[3]

   })
dfsol5 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[4],
                  'ASISTENTES':listaasist[4],
                  'PACIENTES':listapacs[4]

   })
dfsol6 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[5],
                  'ASISTENTES':listaasist[5],
                  'PACIENTES':listapacs[5]

   })
dfsol7 = pd.DataFrame({'Espacio':C+O,
                  'CIRUJANOS':listaciruj[6],
                  'ASISTENTES':listaasist[6],
                  'PACIENTES':listapacs[6]

   })

with pd.ExcelWriter('solucion.xlsx') as writer:
    dfsol1.to_excel(writer, sheet_name = "LUNES")
    dfsol2.to_excel(writer, sheet_name = "MARTES")
    dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
    dfsol4.to_excel(writer, sheet_name = "JUEVES")
    dfsol5.to_excel(writer, sheet_name = "VIERNES")
    dfsol6.to_excel(writer, sheet_name = "SABADO")
    dfsol7.to_excel(writer, sheet_name = "DOMINGO")
```

```
#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")
numor = datos.loc[0,'Número de quirófanos']
numc = datos.loc[0,'Número de consultas']
#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
   aux.append([])
   x=x+1
x=0
listaciruj=[]
listaasist=[]
listapacs=[]
while x<7:
   listaciruj.append(aux)
   listaasist.append(aux)
   listapacs.append(aux)
   x=x+1

empty =[[]]
x=0
while x<numc-1:
   empty.append([])
   x=x+1

dia=0
while dia<1:

   inicio=time.time()
   #NEH#
   #1- ORDENAR LA SECUENCIA INICIAL
   sumatiempos=np.array([0.0]*len(pacientes))
   for x in range(len(pacientes)):
      sumatiempos[x]=dc[x]+dq[x]+dp[x]
   pacientesnumpy = np.array(pacientes)
   ordennumpy = np.array(sumatiempos)
   np.argsort(sumatiempos)
   secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
   secinicial1=np.flip(secinicial0)
   secinicial=secinicial1.tolist()
   #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
   #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
   pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
```

```
hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpi=cajanegrasecuencia(pi)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpiinicial=cajanegrasecuencia(piinicial)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()
print('FOpiinicial antes de NEH', FOpiinicial)

j=1
while j<len(secinicial):

   i=0
   secprobando=pi[:]
   flagcambiado=0
   while i<=j:
      secprobando.insert(i, secinicial[j])
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOprobando=cajanegrasecuencia(secprobando)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      if mejora(FOprobando,FOpi):

         pi=secprobando.copy()
         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         FOpi=cajanegrasecuencia(pi)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         flagcambiado=1
       #no se ha añadido ningún número a la secuencia y terminamos el bucle
      secprobando.pop(i)
      i=i+1
   if flagcambiado==0:
      secprobando.insert(i-1,secinicial[j])
      pi=secprobando[:]
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOpi=cajanegrasecuencia(pi)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()
   j=j+1
   #fin del NEH. Tenemos FOpi y pi
print('NEH tarda', time.time()-inicio)

if mejora(FOpi,FOpiinicial):#si el NEH ha mejorado la secuencia que teniamos inicialmente
   pibest=pi.copy()
   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
   FOpibest=cajanegrasecuencia(pibest)
```

```
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
else:
    pibest=piinicial.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(piinicial)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

print('FOpibest con la secuencia tras NEH y antes de LS', FOpibest)

it=0
Temp=1
inicio=time.time()
while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
    piprima=pi.copy()
    destruida=destruction(piprima)#destruction phase
    inicio1=time.time()
    piprima=construction(destruida[0],destruida[1]) #construction phase
    print('construccion tarda',time.time()-inicio1)
    #LOCAL SEARCH#
    iniciols=time.time()
    n=len(piprima)
    i=0
    j=0
    piprimaprima=piprima.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpiprimaprima=cajanegrasecuencia(piprimaprima)
    print('FO tras construccion', FOpiprimaprima)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
    while j<n:
        i=0
        while i<n:
            secprima=piprima.copy()
            secprima.remove(piprima[j])
            secprima.insert(i,piprima[j])
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOsecprima=cajanegrasecuencia(secprima)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOsecprima,FOpiprimaprima):
                piprimaprima=secprima.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpiprimaprima=cajanegrasecuencia(piprimaprima)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

            i=i+1
        j=j+1
    #FIN LOCAL SEARCH
    print('LS tarda',time.time()-iniciols)
    print('FO tras LS', FOpiprimaprima)
```

```
                if mejora(FOpiprimaprima,FOpi):
                    pi=piprimaprima.copy()
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOpi=cajanegrasecuencia(pi)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()
                    if mejora(FOpi,FOpibest):
                        pibest=pi.copy()
                        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                        FOpibest=cajanegrasecuencia(pibest)
                        hcc=hcccopia.copy()#para que las pruebas no alteren el real
                        hcor=hcorcopia.copy()
                elif random.uniform(0, 1)<=-math.exp(-(FOpiprimaprima[1]-FOpi[1])/Temp):
                    print('se acepta peor solucion')
                    pi=piprimaprima.copy()
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOpi=cajanegrasecuencia(pi)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()
                Temp=0.95*Temp
                it=it+1

            FOpibest=cajanegrasecuencia(pibest)

            print('MEJOR FO DEFINITIVA',FOpibest)
            sumaFO=sumaFO+FOpibest[3]
            actualizarexcel(pacientesaborrar)
            dia=dia+1
        escribirsol(listaciruj, listaasist, listapacs)
        return sumaFO

    def IG1():
        global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        global sumaFO

        def insertar(lista,pos1,pos2):
            copia=lista.copy()
            a=copia.pop(pos1)
            copia.insert(pos2,a)
            return copia

        def mejora(v1probando,v2best):
            if v1probando[3]<v2best[3]:
                return True
            else:
                return False

        def nivelar(patients,surgeons):

            j=min(numc-1,len(surgeons)-1)
            while j>=0:
                if patients[j]==[]:
```

```
                surgeons.pop(j)
                patients.pop(j)
                surgeons.insert(numc-1,[])
                patients.insert(numc-1,[])
              j=j-1
            return surgeons

        def destruction(sequence):#la secuencia que se le pasará será secbest
            n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
            a=0
            secborrar=sequence[:]
            while a<len(n):
              secborrar.remove(n[a])
              a=a+1
            return secborrar, n
        def construction(seqbest,n): #para secuenncia
            seqprobando=seqbest[:]
            a=0
            while a<len(n):
              seqprobando=seqbest[:]
              seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para
luego ir tomando las que la mejoren
              seqbest=seqprobando[:]
              global hcc, hcor
              hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
              hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
              FObest=cajanegrasecuencia(seqbest)
              hcc=hcccopia.copy()#para que las pruebas no alteren el real
              hcor=hcorcopia.copy()
              seqprobando.pop(0)
              ###PROBANDO PARA NO INSERTAR DOS VECES EL MISMO NOMBRE###
              b=1
              while b<len(seqbest):
                seqprobando.insert(b, n[a])
                # print('secuencia probando, insertamos en la posición', secprobando,i)
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOprobando=cajanegrasecuencia(seqprobando)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                if mejora(FOprobando,FObest):
                  seqbest=seqprobando[:]
                  FObest=FOprobando[:]
                 #no se ha añadido ningún número a la secuencia y terminamos el bucle
                seqprobando.pop(b)
                b=b+1
              a=a+1
            return seqbest

        def destruccion(cirujanosconsults): #para asignacion cirujanos a consultas
            n=random.sample(cirujanosconsults,4)
            a=0
            ccborrar=cirujanosconsults[:]
            while a<len(n):
              ccborrar.remove(n[a])
              a=a+1
            return ccborrar, n
```

```
        def construccion(ccmejor,n): #para asignacion cirujanos a consultas
            global hcc, hcor
            ccprobando=ccmejor[:]
            a=0
            while a<len(n):
                ccprobando=ccmejor[:]
                ccprobando.append(n[a])
                ccmejor=ccprobando[:]

                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FObest=cajanegra(pibest,ccmejor)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                ccprobando.pop(-1)
                b=0
                while b<len(ccmejor):
                    ccprobando.insert(b,n[a])

                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOprobando=cajanegra(pibest,ccprobando)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    if mejora(FOprobando,FObest):
                        ccmejor=ccprobando[:]
                        FObest=FOprobando[:]
                    ccprobando.pop(b)
                    b=b+1
                a=a+1
            return ccmejor

        def leerexcel():
            global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado,
cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc
            datospacientes = pd.read_excel("data.xlsx", "Pacientes")
            pacientes = datospacientes['Número paciente'].tolist()
            dp = datospacientes['dp'].tolist()
            trp = datospacientes['trp'].tolist()
            dq = datospacientes['dq'].tolist()
            trq = datospacientes['trq'].tolist()
            u = datospacientes['u'].tolist()
            hp = datospacientes['hp'].tolist()
            hs = datospacientes['hs'].tolist()
            dc = datospacientes['dc'].tolist()
            trc = datospacientes['trc'].tolist()
            pesos = datospacientes['w'].tolist()
            estado = datospacientes['estado'].tolist()
            cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
            cirjasoc = [int(x) for x in cirjasoc1.tolist()]
            for x in range(len(cirjasoc)):
                if cirjasoc[x]==999:
                    cirjasoc[x]='NaN'
            tiempoacum = datospacientes['tacum'].tolist()
        def cajanegra(secuencia,cc):
```

```
        global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        def asignacion(secuencia,cc):
            global pacientesconsultas, pacientesor, cirujanosor, asistentesor, ctconsultas, ctor, hcor, hcc
            pacientesconsultas = [[]]
            x=0
            while x<numc-1:
                pacientesconsultas.append([])
                x=x+1

            pacientesor = [[]]
            cirujanosor = [[]]
            asistentesor = [[]]
            x=0
            while x<numor-1:
                pacientesor.append([])
                cirujanosor.append([])
                asistentesor.append([])
                x=x+1

            global eq
            ###FUNCIONES###

            #Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en
quirófano
                #sirve para medir el equilibrio de la carga de trabajo#
            def equilibrio(hcc,hcor):
                return (hcc.var()+hcor.var())

            #Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta,
ponderado con los pesos
            def lateness(pacientes):
                global pesos
                x=0
                late=[]
                for x in range(len(pacientes)):
                    if estado[x]==0:
                        if tiempoacum[x]>trp[x]:
                            # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                            late.append((tiempoacum[x]-trp[x])*pesos[x])
                    if estado[x]==1:
                        if tiempoacum[x]>trq[x]:
                            # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                            late.append((tiempoacum[x]-trq[x])*pesos[x])
                    if estado[x]==2:
                        if tiempoacum[x]>trc[x]:
                            # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                            late.append((tiempoacum[x]-trc[x])*pesos[x])
                return sum(late) #late indica cuántos pacientes van con retraso actualmente

            #Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
            def asignadodia(cirujano,consultaor):
                # global cirujanosconsultas
                global cirujanosor
                global numc
                i=0
```

```
                flag = 0
                if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
                  for i in range(len(cc[:numc])):
                    if cirujano in cc[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                      return i
                      flag=1
                  if flag==0:
                    return 'NaN'
                elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
                  for i in range(numor):
                    if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR
j, nos devuelve ese valor
                      return i
                      flag=1
                  if flag==0:
                    return 'NaN'

              #funcion para calcular el indice de un vector
              def indice(vector,valor):
                x=0
                for x in range(len(vector)):
                  if vector[x]==valor:
                    return x
                    break

              #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea
necesario coincidir en unidades)
              def busquedaPAE (paciente):
                global cirujanos
                global numc
                global hcc
                j=0
                cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
                while j<len(cirujanos):
                  if uc[j]==u[paciente]:#si el cirujano que estamos viendo es de la unidad
                    cirujanosdelaunidad.append(j)
                  j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosdelaunidad)):
                  horas.append(hcc[cirujanosdelaunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
                np.argsort(horasnumpy)
                cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
                j=0
                flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la
semana
                #para cada cirujano, miramos si está asignado a alguna consulta el dia
                while j<len(cirujanosdelaunidad):

                  asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                  if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta
asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al
dia
                      return [asignadoenconsulta,cirujanosdelaunidad[j]]
```

```
                    flagyatieneconsulta=1
                    break
                if flagyatieneconsulta==1:break
                j=j+1
            if flagyatieneconsulta==0:
                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

        # Función para asignar un paciente y un cirujano a consulta postop
        def busquedaPOSTOP (paciente):
            #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco
            flagyatieneconsulta=0
            asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
            if asignadoenconsulta!='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                    return [asignadoenconsulta,cirjasoc[paciente]]
                    flagyatieneconsulta=1
            if flagyatieneconsulta==0:
                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

        #Función para asignar un paciente y un equipo de cirujanos a quirófanos
        def busquedaOR(paciente):
            #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                #la experiencia y nivel de dificultad de la  operación.
            global hcor
            j=0
            cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
            while j<len(cirujanos):
                if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                    cirujanosexpunidad.append(cirujanos[j])
                    # print(cirujanosexpunidad)
                j=j+1
            # print('lista de cirujanos',cirujanosexpunidad)
            #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosexpunidad)):
                # print('ciruj',cirujanosexpunidad[j])
                horas.append(hcor[cirujanosexpunidad[j]])
            # print(horas)
            horasnumpy=np.array(horas)
            cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
            np.argsort(horasnumpy)
            cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
            #  print('listaordenada',cirujanosexpunidad)#lista de cirujanos ordenadas según horas acumuladas en quirófano. Esta es la lista con la que trabajaremos para obtener los posibles cirujanos
            #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

            ###LISTA ASISTENTES###
            j=0
```

```
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                        # print(asistexp)
                    j=j+1
                # print(asistexp)
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    # print('asistente',asistexp[j] )
                    horas.append(hcor[asistexp[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenadaasist',asistexp)


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    # print('No se necesita asistente')
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                        asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                        # print(' ¿está el cirujano en algun OR el dia {} ?'.format(dia))
                        if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                            # print('si, está asignado en el OR {}'.format(asignadoenOR))
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                                # print('hay hueco en este OR')
                                return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                                flagyatieneOR=1
                                break
                        j=j+1
                    if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                        # print('entra')
                        j=0
                        while j<numor:
                            # print('los cirujanos no tienen consulta asignada con hueco en la semana, así que se coge el primer hueco libre y se ve si algun ciruj puede')
                            if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                                x=0
                                for x in cirujanosexpunidad:

                                    if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                        # print('este ciruj SI puede')
                                        return [j,cirujanosexpunidad[0],'NaN']
                                        flagyatieneOR=1
```

```
                                    break#cuando se le asigna, se sale del for
                                if flagyatieneOR==1:
                                    break
                                j=j+1
                            if flagyatieneOR==0:
                                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                        else:   #sí se necesita un asistente
                            # print('si se necesita un asistente')
                            #vamos a buscar a la vez un cirujano principal y un asistente
                            i=0
                            flagyatieneasist=0
                            for i in cirujanosexpunidad:
                                # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                                asignadoenOR = asignadodia(i,'OR')
                                # print('¿está asignado en algun OR ya?')
                                if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                                    # print('si, esta en el OR',asignadoenOR)
                                    if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                        # print('sí habría hueco en su OR')
                                        #buscamos un asistente
                                        k=0
                                        for k in asistexp:
                                            # print('probamos con asist',k)
                                            # print('asignado en consulta',asignadodia(k,'consulta'))
                                            # print('asignado en OR',asignadodia(k,'OR')=='NaN')
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
                                                # print('retornamos')
                                                return [asignadoenOR,i,k]
                                                flagyatieneasist=1
                                                break
                                    if flagyatieneasist==1:break
                            if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                                #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                                # print('no hay ciruj con OR asignado, probamos a buscar el primer hueco')
                                i=0
                                for i in cirujanosexpunidad:
                                    asignadoenconsulta = asignadodia(i,'consulta')
                                    # print('¿está asignado en consulta este ciruj ya?',asignadoenconsulta)
                                    x=0
                                    while x<numor:
                                        # print('está ya asignado a un OR?',asignadodia(i,'OR'))
                                        if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                            #print(ctor[dia][x][-1])
                                            if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                            #buscamos un asistente
                                                # print('hay hueco')
                                                for k in asistexp:
                                                    # print('probamos con el asist {}, cirujanos {}, asignado en consulta? {} asignado en OR? {}'.format(k,i,asignadodia(k,'consulta'),asignadodia(k,'OR')))
```

```
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                        # print('asistente disponible')
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia  asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        return [x,i,k]
                                        flagyatieneasist=1
                                        break
                                if flagyatieneasist==1:break
                                x=x+1
                            if flagyatieneasist==1:break
                    if flagyatieneasist==0:
                        # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                        return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                        #significa que no hay hueco posible para este paciente en OR esta semana


                ###COMIENZO CODIGO###

                global pacientesaborrar
                atendidos=0
                pacientesaborrar=[]
                for k in pacientes:
                    if k not in secuencia:
                        tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1 #los pacientes que no estén en la sec, igualmente no se van a procesar y se les añade un día a su tiempo acum
                for k in range(len(secuencia)): #x es el indice del paciente
                    # print(' se ve al paciente {}'.format(secuencia[k]))
                    if estado[indice(pacientes,secuencia[k])]==0:
                        # print('vamos a busquedaPAE ')
                        retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                        # print(retorno)
                        if retorno!='NaN':
                            pacientesconsultas[retorno[0]].append(secuencia[k])
                            ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                            hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                            tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el +1 porque el día 0 aumenta 1 día
                            # print(equilibrio(hcc,hcor))
                            # print('se asigna el paciente {} a la consulta {} el día  con el cirujano {}'.format(secuencia[x],retorno[0],retorno[1]))
                            flagcambiado=1
                            atendidos=atendidos+1
                        else:
                            flagcambiado=0

                tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        # print('vamos a OR')
                        retorno=busquedaOR(indice(pacientes,secuencia[k]))  #me devuelve [consulta,cirujprincipal,asistente]
                        # print(retorno)
                        if retorno!='NaN':
                            # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente {}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
```

```
                    cirujanosor[retorno[0]].append(retorno[1])
                    hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                    pacientesor[retorno[0]].append(secuencia[k])
                    ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                    cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    if retorno[2]!='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                        hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                    if retorno[2]=='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                    flagcambiado=1
                    atendidos=atendidos+1
                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1 #se actualiza el tiempo acumulado.
            if estado[indice(pacientes,secuencia[k])]==2:
                # print('vamos a POSTOP')
                retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal]
                # print('retorno',retorno)
                if retorno!='NaN':
                    pacientesconsultas[retorno[0]].append(secuencia[k])
                    ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dc[indice(pacientes,secuencia[k])])
                    #habría que eliminar al paciente de la lista
                    hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                    flagcambiado=1
                    atendidos=atendidos+1
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el hospital#
                    pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

            if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or estado[indice(pacientes,secuencia[k])]==1):
                estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==1:
                    mu=random.randint(1,4)
                    sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                    dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

        eq=equilibrio(hcc,hcor)
        tarde=lateness(pacientes)
        utilizacionor=0
        utilizacionc=0
        i=0

        while i<numc:
            utilizacionc=ctconsultas[i][-1]+utilizacionc
            i=i+1
```

```
            i=0
            while i<numor:
               utilizacionor=ctor[i][-1]+utilizacionor
               i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return     eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


         #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
            # #creo las listas#
            leerexcel()
             #auxiliares#
            ctconsultas = [[0]]
            x=0
            while x<numc-1:
               ctconsultas.append([0])
               x=x+1

            ctor = [[0]]
            x=0
            while x<numor-1:
               ctor.append([0])
               x=x+1

            global dia
            leerexcel()
            vectorFO=asignacion(secuencia,cc)

            listaciruj[dia] = cc+cirujanosor
            listaasist[dia] = empty+asistentesor
            listapacs[dia] = pacientesconsultas+pacientesor

            return vectorFO

         def cajanegrasecuencia(secuencia):
            global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
            global listaciruj, listaasist, listapacs
            def asignacion(secuencia):
               global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor,
ctconsultas, ctor, hcor, hcc
               pacientesconsultas = [[]]
               cirujanosconsultas=[[]]
               x=0
               while x<numc-1:
                  pacientesconsultas.append([])
                  cirujanosconsultas.append([])
                  x=x+1

               pacientesor = [[]]
               cirujanosor = [[]]
               asistentesor = [[]]
               x=0
               while x<numor-1:
                  pacientesor.append([])
```

```
        cirujanosor.append([])
        asistentesor.append([])
        x=x+1

    #Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
        #sirve para medir el equilibrio de la carga de trabajo#
    def equilibrio(hcc,hcor):
        return (hcc.var()+hcor.var())

    #Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
    def lateness(pacientes):
        global pesos
        x=0
        late=[]
        for x in range(len(pacientes)):
            if estado[x]==0:
                if tiempoacum[x]>trp[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trp[x])*pesos[x])
            if estado[x]==1:
                if tiempoacum[x]>trq[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trq[x])*pesos[x])
            if estado[x]==2:
                if tiempoacum[x]>trc[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trc[x])*pesos[x])
        # print('Número de pacientes que van tarde:')
        return sum(late) #late indica cuántos pacientes van con retraso actualmente

    #Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
    def asignadodia(cirujano,consultaor):
        global cirujanosconsultas
        global cirujanosor
        global numc
        i=0
        flag = 0
        if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
            for i in range(len(cirujanosconsultas)):
                if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                    return i
                    flag=1
            if flag==0:
                return 'NaN'
                # print("El cirujano no está asignado a ninguna consulta el dia ",dia)
        elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
            for i in range(numor):
                if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
                    return i
                    flag=1
            if flag==0:
                return 'NaN'
                # print("El cirujano no está asignado a ningún quirófano el dia ",dia)

    #funcion para calcular el indice de un vector
```

```
def indice(vector,valor):
    x=0
    for x in range(len(vector)):
        if vector[x]==valor:
            return x
            break

#Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
def busquedaPAE (paciente):
    global cirujanos
    global numc
    global hcc
    global cirujanosconsultas
    j=0
    cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
    while j<len(cirujanos):
        if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad
            cirujanosdelaunidad.append(j)
        j=j+1
    # print(cirujanosdelaunidad)
    #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
    j=0
    horas=[]
    for j in range(len(cirujanosdelaunidad)):
        horas.append(hcc[cirujanosdelaunidad[j]])
    # print(horas)
    horasnumpy=np.array(horas)
    cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
    np.argsort(horasnumpy)
    cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
    # print('lista de cirujanos posibles ordenadas',cirujanosdelaunidad)#lista de cirujanos ordenadas según horas acumuladas en consulta
    j=0
    flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
    #para cada cirujano, miramos si está asignado a alguna consulta el dia
    while j<len(cirujanosdelaunidad):

        # print('¿esta el paciente asignado el dia {} a consulta?'.format(dia))
        asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
        if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
            # print('si, está asignado el dia {} en la consulta {}'.format(dia,asignadoenconsulta))
            if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                # print('sí hay hueco en la consulta')
                return [asignadoenconsulta,cirujanosdelaunidad[j]]
                flagyatieneconsulta=1
                break

        if flagyatieneconsulta==1:break
        j=j+1
    if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero la primera consulta libre
        # print('el cirujano no tiene consulta asignada en toda la semana, así que le buscamos la primera consulta libre')
```

```
                    j=0

                    while j<len(cirujanosconsultas):
                        # print(' dia{} consulta {} '.format(dia,j))
                        # print('tiempo que quedaria',ctconsultas[dia][j][-1]+dp[paciente])
                        if cirujanosconsultas[j]==[]:#si la consulta está libre
                            x=0
                            for x in cirujanosdelaunidad:
                                # print('cirujano posible en este hueco',x)
                                # print('está asignado en OR? {} está asignado en consulta?{}'.format(asignadodia(x, 'OR', dia),asignadodia(x, 'consulta', dia)))
                                if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or asignadodia(x, 'consulta', dia)==j):
                                    #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa misma,
                                    # print('entroaqui')
                                    return [j,x]
                                    flagyatieneconsulta=1
                                    break#cuando se le asigna, se sale del for
                            if flagyatieneconsulta==1:
                                break
                        j=j+1
                    if flagyatieneconsulta==0:
                        # print('nop')
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


                # Función para asignar un paciente y un cirujano a consulta postop
                def busquedaPOSTOP (paciente):
                    #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                    flagyatieneconsulta=0
                    # print('busqueda POSTOP para paciente',paciente)

                    asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                    # print('el cirujano está asignado en consulta este dia?',asignadoenconsulta)
                    asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                    # print('el cirujano está asignado en OR este dia?',asignadoenOR)
                    if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                        if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                            # print('hya hueco en esa consulta, asi que se añade ahi')
                            return [asignadoenconsulta,cirjasoc[paciente]]
                            flagyatieneconsulta=1
                    if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                        #se le asigna el primer hueco que haya en la semana
                        # print('el cirujano no tiene consulta con hueco ninguna dia, asi que se le asigna en el primer hueco que haya en la semana')
                        j=0

                        while j<len(cirujanosconsultas):
                            # print('dia {} y consulta {}'.format(dia,j))
                            if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                                # print('hay hueco en esta consulta')
                                # print('vemos si el ciruj está asignado en algun OR ese día')
```

```
                    if asignadoenconsulta=='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                        return [j,cirjasoc[paciente]]
                        flagyatieneconsulta=1
                        break#cuando se le asigna, se sale del for
                    if flagyatieneconsulta==1:
                        break
                    j=j+1
                if flagyatieneconsulta==0:
                    # print('no hay hueco posible')
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana



            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                    #la experiencia y nivel de dificultad de la  operación.
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
                        # print(cirujanosexpunidad)
                    j=j+1
                # print('lista de cirujanos',cirujanosexpunidad)
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    # print('ciruj',cirujanosexpunidad[j])
                    horas.append(hcor[cirujanosexpunidad[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                #  print('listaordenada',cirujanosexpunidad)#lista de cirujanos ordenadas según horas acumuladas en quirófano. Esta es la lista con la que trabajaremos para obtener los posibles cirujanos
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente


                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                        # print(asistexp)
                    j=j+1
                # print(asistexp)
```

```
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                   # print('asistente',asistexp[j] )
                   horas.append(hcor[asistexp[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenadaasist',asistexp)


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    # print('No se necesita asistente')
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                       asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                       # print(' ¿está el cirujano en algun OR el dia {} ?'.format(dia))
                       if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                          # print('si, está asignado en el OR {}'.format(asignadoenOR))
                          if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                             # print('hay hueco en este OR')
                             return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                             flagyatieneOR=1
                             break
                       j=j+1
                    if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                       # print('entra')
                       j=0
                       while j<numor:
                          # print('los cirujanos no tienen consulta asignada con hueco en la semana, así que se coge el primer hueco libre y se ve si algun ciruj puede')
                          if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                             x=0
                             for x in cirujanosexpunidad:
                                # print('probamos con el ciruj',x)
                                # print('asignadodia',asignadodia(x, 'consulta'))
                                # print('asignadodOR',asignadodia(x, 'OR'))
                                if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                   # print('este ciruj SI puede')
                                   return [j,cirujanosexpunidad[0],'NaN']
                                   flagyatieneOR=1
                                   break#cuando se le asigna, se sale del for
                             if flagyatieneOR==1:
                                break
                          j=j+1
                    if flagyatieneOR==0:
                       return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
```

```
                else:   #sí se necesita un asistente
                    # print('si se necesita un asistente')
                    #vamos a buscar a la vez un cirujano principal y un asistente
                    i=0
                    flagyatieneasist=0
                    for i in cirujanosexpunidad:
                        # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                        asignadoenOR = asignadodia(i,'OR')
                        # print('¿está asignado en algun OR ya?')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                            # print('si, esta en el OR',asignadoenOR)
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                # print('sí habría hueco en su OR')
                                #buscamos un asistente
                                k=0
                                for k in asistexp:
                                    # print('probamos con asist',k)
                                    # print('asignado en consulta',asignadodia(k,'consulta'))
                                    # print('asignado en OR',asignadodia(k,'OR')=='NaN')
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        # print('retornamos')
                                        return [asignadoenOR,i,k]
                                        flagyatieneasist=1
                                        break
                            if flagyatieneasist==1:break
                        if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                            #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                            # print('no hay ciruj con OR asignado, probamos a buscar el primer hueco')
                            i=0
                            for i in cirujanosexpunidad:
                                asignadoenconsulta = asignadodia(i,'consulta')
                                # print('¿está asignado en consulta este ciruj ya?',asignadoenconsulta)
                                x=0
                                while x<numor:
                                    # print('está ya asignado a un OR?',asignadodia(i,'OR'))
                                    if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                        #print(ctor[dia][x][-1])
                                        if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                        #buscamos un asistente
                                            # print('hay hueco')
                                            for k in asistexp:
                                                # print('probamos con el asist {}, cirujanos {}, asignado en consulta? {} asignado en OR? {}'.format(k,i,asignadodia(k,'consulta'),asignadodia(k,'OR')))
                                                if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                                    # print('asistente disponible')
                                                    #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                    #si esta condicion se cumple, se asocia este asist
                                                    return [x,i,k]
                                                    flagyatieneasist=1
```

```
                                    break
                                if flagyatieneasist==1:break
                                x=x+1
                            if flagyatieneasist==1:break
                        if flagyatieneasist==0:
                            # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                            return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                            #significa que no hay hueco posible para este paciente en OR esta semana


                    ###COMIENZO CODIGO###

                    global pacientesaborrar
                    atendidos=0
                    pacientesaborrar=[]
                    for k in pacientes:
                        if k not in secuencia:
                            tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
                    for k in range(len(secuencia)): #x es el indice del paciente
                        # print(' se ve al paciente {}'.format(secuencia[k]))
                        if estado[indice(pacientes,secuencia[k])]==0:
                            # print('vamos a busquedaPAE ')
                            retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                            # print(retorno)
                            if retorno!='NaN':
                                cirujanosconsultas[retorno[0]]=[retorno[1]]
                                pacientesconsultas[retorno[0]].append(secuencia[k])
                                ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                                hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                                tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1 día
                                # print(equilibrio(hcc,hcor))
                                # print('se asigna el paciente {} a la consulta {} el día  con el cirujano {}'.format(secuencia[x],retorno[0],retorno[1]))
                                flagcambiado=1
                                atendidos=atendidos+1
                            else:
                                flagcambiado=0

                        tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                        if estado[indice(pacientes,secuencia[k])]==1:
                            # print('vamos a OR')
                            retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                            # print(retorno)
                            if retorno!='NaN':
                                # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente {}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
                                cirujanosor[retorno[0]].append(retorno[1])
                                hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                                pacientesor[retorno[0]].append(secuencia[k])
                                ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                                cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                                tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                                if retorno[2]!='NaN':
                                    asistentesor[retorno[0]].append(retorno[2])
```

```
                    hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                  if retorno[2]=='NaN':
                    asistentesor[retorno[0]].append(retorno[2])
                  flagcambiado=1
                  atendidos=atendidos+1
                else:
                  flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
              if estado[indice(pacientes,secuencia[k])]==2:
                # print('vamos a POSTOP')
                retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve
[consulta,cirujprincipal]
                # print('retorno',retorno)
                if retorno!='NaN':
                  cirujanosconsultas[retorno[0]]=[retorno[1]]
                  pacientesconsultas[retorno[0]].append(secuencia[k])
                  ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                  #habría que eliminar al paciente de la lista
                  hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                  flagcambiado=1
                  atendidos=atendidos+1
                  tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                  # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                  pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])


                else:
                  flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

              if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==1:
                  mu=random.randint(1,4)
                  sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                  dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)
            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
              utilizacionc=ctconsultas[i][-1]+utilizacionc
              i=i+1

            i=0
            while i<numor:
              utilizacionor=ctor[i][-1]+utilizacionor
              i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
```

```
        return     eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


        #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
        # #creo las listas#
        leerexcel()
        ctconsultas = [[0]]
        x=0
        while x<numc-1:
          ctconsultas.append([0])
          x=x+1

        ctor = [[0]]
        x=0
        while x<numor-1:
          ctor.append([0])
          x=x+1

        global dia
        leerexcel()
        vectorFO=asignacion(secuencia)

        listaciruj[dia] = cirujanosconsultas+cirujanosor
        listaasist[dia] = empty+asistentesor
        listapacs[dia] = pacientesconsultas+pacientesor

        return vectorFO

      def actualizarexcel(pacientesaborrar):
        #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
        j=0
        x=0
        # print('pacientes a borrar', pacientesaborrar)
        while j < len(pacientesaborrar):
          x=0
          # print('j',j)
          while x<len(pacientes):
            # print('x',x)
            if pacientesaborrar[j]==pacientes[x]:
              # print('se borra el paciente', pacientesaborrar[j])
              del pacientes[x]
              del cirjasoc[x]
              del dc[x]
              del dp[x]
              del dq[x]
              del estado[x]
              del hp[x]
              del hs[x]
              del pesos[x]
              del tiempoacum[x]
              del trc[x]
              del trp[x]
              del trq[x]
              del u[x]
            else:x=x+1
          j=j+1
```

```
    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                            'dp':dp,
                            'trp':trp,
                            'dq':dq,
                            'trq':trq,
                            'u':u,
                            'hp':hp,
                            'hs':hs,
                            'dc':dc,
                            'trc':trc,
                            'w':pesos,
                            'estado':estado,
                            'cirujano':cirjasoc,
                            'tacum':tiempoacum
                            })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                            'G':exp,
                            'S':uc
       })
    with pd.ExcelWriter('data.xlsx') as writer:
       nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
       datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
       datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):
    c=0
    C=[]
    while c<numc:
       C.append('C')
       c=c+1
    o=0
    O=[]
    while o<numor:
       O.append('O')
       o=o+1
    dfsol1 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[0],
                    'ASISTENTES':listaasist[0],
                    'PACIENTES':listapacs[0]

       })
    dfsol2 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[1],
                    'ASISTENTES':listaasist[1],
                    'PACIENTES':listapacs[1]

       })
    dfsol3 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[2],
                    'ASISTENTES':listaasist[2],
                    'PACIENTES':listapacs[2]

       })
    dfsol4 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[3],
                    'ASISTENTES':listaasist[3],
                    'PACIENTES':listapacs[3]
```

```
    })
    dfsol5 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[4],
                           'ASISTENTES':listaasist[4],
                           'PACIENTES':listapacs[4]

    })
    dfsol6 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[5],
                           'ASISTENTES':listaasist[5],
                           'PACIENTES':listapacs[5]

    })
    dfsol7 = pd.DataFrame({'Espacio':C+O,
                           'CIRUJANOS':listaciruj[6],
                           'ASISTENTES':listaasist[6],
                           'PACIENTES':listapacs[6]

    })

    with pd.ExcelWriter('solucion.xlsx') as writer:
        dfsol1.to_excel(writer, sheet_name = "LUNES")
        dfsol2.to_excel(writer, sheet_name = "MARTES")
        dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
        dfsol4.to_excel(writer, sheet_name = "JUEVES")
        dfsol5.to_excel(writer, sheet_name = "VIERNES")
        dfsol6.to_excel(writer, sheet_name = "SABADO")
        dfsol7.to_excel(writer, sheet_name = "DOMINGO")



#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")

#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
    aux.append([])
    x=x+1
x=0
listaciruj=[]
listaasist=[]
listapacs=[]
while x<7:
```

```
            listaciruj.append(aux)
            listaasist.append(aux)
            listapacs.append(aux)
            x=x+1

        empty =[[]]
        x=0
        while x<numc-1:
            empty.append([])
            x=x+1

        dia=0
        while dia<1:

            #NEH#
            #1- ORDENAR LA SECUENCIA INICIAL
            sumatiempos=np.array([0.0]*len(pacientes))
            for x in range(len(pacientes)):
                # print(x, dc[x], dp[x], dq[x])
                sumatiempos[x]=dc[x]+dq[x]+dp[x]
            pacientesnumpy = np.array(pacientes)
            ordennumpy = np.array(sumatiempos)
            np.argsort(sumatiempos)
            secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
            secinicial1=np.flip(secinicial0)
            secinicial=secinicial1.tolist()
            #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
            #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
            # print(secinicial)
            pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()


            piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpiinicial=cajanegrasecuencia(piinicial)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
            print('FOpiinicial antes de NEH', FOpiinicial)
            inicio=time.time()
            j=1
            while j<len(secinicial):

                i=0
                secprobando=pi[:]
                # print('secuencia que estamos probando', secprobando)
                flagcambiado=0
                while i<=j:
                    secprobando.insert(i, secinicial[j])
                    # print('secuencia probando, insertamos en la posición', secprobando,i)
```

```
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOprobando=cajanegrasecuencia(secprobando)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOprobando,FOpi):
            pi=secprobando[:]
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            flagcambiado=1
         #no se ha añadido ningún número a la secuencia y terminamos el bucle
        secprobando.pop(i)
        i=i+1
    if flagcambiado==0:
        secprobando.insert(i-1,secinicial[j])
        pi=secprobando[:]
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    j=j+1


    #fin del NEH. Tenemos FOpi y pi

fin=time.time()
print('El NEH tarda:', fin-inicio)
if mejora(FOpi,FOpiinicial):
    pibest=pi.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(pibest)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
else:
    pibest=piinicial.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(piinicial)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

print('FOpibest tras NEH', FOpibest)

it=0
Temp=1
principio=time.time()
while time.time()-principio<len(pacientes)*(numor+numc)*0.025:
    piprima=pi.copy()
    destruida=destruction(piprima)#destruction phase
    inicio=time.time()
    piprima=construction(destruida[0],destruida[1]) #construction phase
```

```
    fin=time.time()
    print('construccion tarda', fin-inicio)
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpiprima=cajanegrasecuencia(piprima)
    print('FO tras construccion', FOpiprima)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
    if mejora(FOpiprima,FOpi):
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
        if mejora(FOpi,FOpibest):
            pibest=pi.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegrasecuencia(pibest)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
    elif random.uniform(0, 1)<=-math.exp(-(FOpiprima[1]-FOpi[1])/Temp):
        print('se acepta peor solucion')
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    Temp=0.95*Temp
    it=it+1

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpibest=cajanegrasecuencia(pibest)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()
print('FOpibest tras destruc+construc', FOpibest)
#aquí ya tendríamos IG para la secuencia
#aplicamos ahora el IG para la asignación de cirujanos

ccsec=cirujanosconsultas.copy()#Se parte de esta asignación para realizar e NEH

ccinicial=ccsec[:]
for x in cirujanos:
    if [x] not in ccinicial and descanso[x][dia]==0:
        ccinicial.append([x])

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOcc=cajanegra(pibest,ccsec)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccsec[:numc])
```

```
FOcc=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()


cc=[ccinicial[0]]
j=1
inicio=time.time()
#NEH#
while j<len(ccinicial):
   i=0
   ccprobando=cc[:]
   flagcambiado=0
   while i<=j:
      ccprobando.insert(i,ccinicial[j])
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOprobando=cajanegra(pibest,ccprobando[:numc])
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      s=nivelar(pacientesconsultas,ccprobando[:numc])
      FOprobando=cajanegra(pibest,s)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      if mejora(FOprobando,FOcc):
         cc=ccprobando[:]
         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         FOcc=cajanegra(pibest,cc[:numc])
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         s=nivelar(pacientesconsultas,cc[:numc])
         FOcc=cajanegra(pibest,s)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         flagcambiado=1
      ccprobando.pop(i)
      i=i+1
   if flagcambiado==0:
      ccprobando.insert(i-1,ccinicial[j])
      cc=ccprobando[:]

      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOcc=cajanegra(pibest,cc[:numc])
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
```

```
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

        j=j+1
#FIN NEH#
    fin=time.time()
    print('NEH para cirujs tarda', fin-inicio)
    FOccbest=FOcc[:]
    ccbest=cc.copy()

    print('Foccbest tras NEH', FOccbest)
    it=0
    Temp=1
    inicio1=time.time()
    while time.time()-inicio1<len(pacientes)*(numor+numc)*0.025:

        # ccprima=cc.copy()
        # FOccprima=FOcc[:]
        #destruccion
        ccdestruida=destruccion(ccbest)
        inicio2=time.time()
        ccprima=construccion(ccdestruida[0],ccdestruida[1])
        fin=time.time()
        print('construccion paara cirujs tarda', fin-inicio2)

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccprima=cajanegra(pibest,ccprima[:numc])
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccprima[:numc])
        FOccprima=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOccprima,FOcc):
            cc=ccprima.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOcc,FOccbest):
                ccbest=cc.copy()
```

```
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOccbest=cajanegra(pibest,ccbest[:numc])
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    s=nivelar(pacientesconsultas,ccbest[:numc])
                    FOccbest=cajanegra(pibest,s)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                elif random.uniform(0, 1)<=-math.exp(-(FOccprima[1]-FOcc[1])/Temp):
                    print('se acepta peor solucion')
                    cc=ccprima.copy()
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOcc=cajanegra(pibest,cc)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()
                Temp=0.95*Temp
                it=it+1

            #aquí ya tendríamos FOccbest y ccbest, que serían las mejores soluciones tras el IG

                print('FOccbest tras destruc+construc', FOccbest)
                secuencia=pibest.copy()
                cirujanosconsultas=ccbest.copy()

                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FO=cajanegra(secuencia, cirujanosconsultas[:numc]) #FO DEFINITIVA#
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                s=nivelar(pacientesconsultas,cirujanosconsultas[:numc])
                FO=cajanegra(secuencia,s)
                print('MEJOR FO DEFINITIVA',FO)
                sumaFO=sumaFO+FO[3]

                # print(secuencia)
                actualizarexcel(pacientesaborrar)
                dia=dia+1

            escribirsol(listaciruj, listaasist, listapacs)

            return sumaFO

        def IG1LS():

            global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc,
cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor,
tiempoacum
            global listaciruj, listaasist, listapacs
            global sumaFO

            def insertar(lista,pos1,pos2):
```

```
    copia=lista.copy()
    a=copia.pop(pos1)
    copia.insert(pos2,a)
    return copia

def mejora(v1probando,v2best):
    if v1probando[3]<v2best[3]:
        return True
    else:
        return False

def nivelar(patients,surgeons):

    j=min(numc-1,len(surgeons)-1)
    while j>=0:
        if patients[j]==[]:
            surgeons.pop(j)
            patients.pop(j)
            surgeons.insert(numc-1,[])
            patients.insert(numc-1,[])
        j=j-1
    return surgeons

def destruction(sequence):#la secuencia que se le pasará será secbest
    n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
    # print(n)
    a=0
    secborrar=sequence[:]
    while a<len(n):
        secborrar.remove(n[a])
        a=a+1
    return secborrar, n
def construction(seqbest,n): #para secuenncia
    seqprobando=seqbest[:]
    a=0
    while a<len(n):
        seqprobando=seqbest[:]
        seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para
luego ir tomando las que la mejoren
        seqbest=seqprobando[:]
        global hcc, hcor
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FObest=cajanegrasecuencia(seqbest)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
        # print('la primera secuencia es', secbest, FObest[1])
        seqprobando.pop(0)
        ###PROBANDO PARA NO INSERTAR DOS VECES EL MISMO NOMBRE###
        b=1
        while b<len(seqbest):
          # print('i',i)
          seqprobando.insert(b, n[a])
          # print('secuencia probando, insertamos en la posición', secprobando,i)
          hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
          hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
          FOprobando=cajanegrasecuencia(seqprobando)
          hcc=hcccopia.copy()#para que las pruebas no alteren el real
```

```
            hcor=hcorcopia.copy()

            # print('FO=',FOprobando[1])
            if mejora(FOprobando,FObest):
               # print('mejora')
               seqbest=seqprobando[:]
               FObest=FOprobando[:]
             #no se ha añadido ningún número a la secuencia y terminamos el bucle
            seqprobando.pop(b)
            b=b+1
         a=a+1
      return seqbest

def destruccion(cirujanosconsults): #para asignacion cirujanos a consultas
      n=random.sample(cirujanosconsults,4)
      a=0
      ccborrar=cirujanosconsults[:]
      while a<len(n):
         ccborrar.remove(n[a])
         a=a+1
      return ccborrar, n

def construccion(ccmejor,n): #para asignacion cirujanos a consultas
      global hcc, hcor
      ccprobando=ccmejor[:]
      a=0
      while a<len(n):
         ccprobando=ccmejor[:]
         ccprobando.append(n[a])
         ccmejor=ccprobando[:]

         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         FObest=cajanegra(pibest,ccmejor)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         ccprobando.pop(-1)
         b=0
         while b<len(ccmejor):
            ccprobando.insert(b,n[a])

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegra(pibest,ccprobando)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOprobando,FObest):
               # print('mejora')
               ccmejor=ccprobando[:]
               FObest=FOprobando[:]
            ccprobando.pop(b)
            b=b+1
         a=a+1
      return ccmejor
```

```
        def leerexcel():
            global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado, cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc

            datospacientes = pd.read_excel("data.xlsx", "Pacientes")

            pacientes = datospacientes['Número paciente'].tolist()

            dp = datospacientes['dp'].tolist()

            trp = datospacientes['trp'].tolist()

            dq = datospacientes['dq'].tolist()

            trq = datospacientes['trq'].tolist()

            u = datospacientes['u'].tolist()

            hp = datospacientes['hp'].tolist()

            hs = datospacientes['hs'].tolist()

            dc = datospacientes['dc'].tolist()

            trc = datospacientes['trc'].tolist()

            pesos = datospacientes['w'].tolist()

            estado = datospacientes['estado'].tolist()

            cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
            cirjasoc = [int(x) for x in cirjasoc1.tolist()]
            for x in range(len(cirjasoc)):
               if cirjasoc[x]==999:
                  cirjasoc[x]='NaN'

            tiempoacum = datospacientes['tacum'].tolist()
        def cajanegra(secuencia,cc):

          global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
            global listaciruj, listaasist, listapacs
            def asignacion(secuencia,cc):
               global pacientesconsultas, pacientesor, cirujanosor, asistentesor, ctconsultas, ctor, hcor, hcc
               pacientesconsultas = [[]]
               x=0
               while x<numc-1:
                  pacientesconsultas.append([])
                  x=x+1

               pacientesor = [[]]
               cirujanosor = [[]]
               asistentesor = [[]]
               x=0
               while x<numor-1:
                  pacientesor.append([])
                  cirujanosor.append([])
                  asistentesor.append([])
```

```
        x=x+1

    global eq
    ###FUNCIONES###

    #Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
        #sirve para medir el equilibrio de la carga de trabajo#
    def equilibrio(hcc,hcor):
        return (hcc.var()+hcor.var())

    #Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta, ponderado con los pesos
    def lateness(pacientes):
        global pesos
        x=0
        late=[]
        for x in range(len(pacientes)):
            if estado[x]==0:
                if tiempoacum[x]>trp[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trp[x])*pesos[x])
            if estado[x]==1:
                if tiempoacum[x]>trq[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trq[x])*pesos[x])
            if estado[x]==2:
                if tiempoacum[x]>trc[x]:
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trc[x])*pesos[x])
        # print('Número de pacientes que van tarde:')
        return sum(late)  #late indica cuántos pacientes van con retraso actualmente

    #Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
    def asignadodia(cirujano,consultaor):
        # global cirujanosconsultas
        global cirujanosor
        global numc
        i=0
        flag = 0
        if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
            for i in range(len(cc[:numc])):
                if cirujano in cc[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                    return i
                    flag=1
            if flag==0:
                return 'NaN'
                # print(''El cirujano no está asignado a ninguna consulta el dia '',dia)
        elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
            for i in range(numor):
                if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
                    return i
                    flag=1
            if flag==0:
                return 'NaN'

    #funcion para calcular el indice de un vector
```

```
def indice(vector,valor):
    x=0
    for x in range(len(vector)):
        if vector[x]==valor:
            return x
            break

#Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
def busquedaPAE (paciente):
    global cirujanos
    global numc
    global hcc
    j=0
    cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
    while j<len(cirujanos):
        if uc[j]==u[paciente]:#si el cirujano que estamos viendo es de la unidad
            cirujanosdelaunidad.append(j)
        j=j+1
    #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
    j=0
    horas=[]
    for j in range(len(cirujanosdelaunidad)):
        horas.append(hcc[cirujanosdelaunidad[j]])
    horasnumpy=np.array(horas)
    cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
    np.argsort(horasnumpy)
    cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
    j=0
    flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
    #para cada cirujano, miramos si está asignado a alguna consulta el dia
    while j<len(cirujanosdelaunidad):

        asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
        if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
            if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                return [asignadoenconsulta,cirujanosdelaunidad[j]]
                flagyatieneconsulta=1
                break
        if flagyatieneconsulta==1:break
        j=j+1
    if flagyatieneconsulta==0:
        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

# Función para asignar un paciente y un cirujano a consulta postop
def busquedaPOSTOP (paciente):
    #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

    flagyatieneconsulta=0
    asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
    asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
    if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
```

```
                    if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                        return [asignadoenconsulta,cirjasoc[paciente]]
                        flagyatieneconsulta=1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                    #la experiencia y nivel de dificultad de la  operación.
                global hcor
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
                    j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    # print('ciruj',cirujanosexpunidad[j])
                    horas.append(hcor[cirujanosexpunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                        # print(asistexp)
                    j=j+1
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    # print('asistente',asistexp[j] )
                    horas.append(hcor[asistexp[j]])
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()

                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    j=0
```

```
                flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                #para cada cirujano, miramos si está asignado a algún OR el dia
                while j<len(cirujanosexpunidad):
                   asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                   if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                      if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                         return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                         flagyatieneOR=1
                         break
                   j=j+1
                if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                   j=0
                   while j<numor:
                      if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                         x=0
                         for x in cirujanosexpunidad:
                            if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                               # print('este ciruj SI puede')
                               return [j,cirujanosexpunidad[0],'NaN']
                               flagyatieneOR=1
                               break#cuando se le asigna, se sale del for
                         if flagyatieneOR==1:
                            break
                      j=j+1
                if flagyatieneOR==0:
                   return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
             else:   #sí se necesita un asistente
                #vamos a buscar a la vez un cirujano principal y un asistente
                i=0
                flagyatieneasist=0
                for i in cirujanosexpunidad:
                   # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                   asignadoenOR = asignadodia(i,'OR')
                   # print('¿está asignado en algun OR ya?')
                   if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                      # print('si, esta en el OR',asignadoenOR)
                      if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                         # print('sí habría hueco en su OR')
                         #buscamos un asistente
                         k=0
                         for k in asistexp:
                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                               #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                               #si esta condicion se cumple, se asocia este asist
                               # print('retornamos')
                               return [asignadoenOR,i,k]
                               flagyatieneasist=1
                               break
                   if flagyatieneasist==1:break
                if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
```

```
                    #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                    i=0
                    for i in cirujanosexpunidad:
                        asignadoenconsulta = asignadodia(i,'consulta')
                        x=0
                        while x<numor:
                            if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                #buscamos un asistente
                                    for k in asistexp:
                                        if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                            return [x,i,k]
                                            flagyatieneasist=1
                                            break
                                if flagyatieneasist==1:break
                                x=x+1
                            if flagyatieneasist==1:break
                    if flagyatieneasist==0:
                        # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                        return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                        #significa que no hay hueco posible para este paciente en OR esta semana

                ###COMIENZO CODIGO###

                global pacientesaborrar
                atendidos=0
                pacientesaborrar=[]
                for k in pacientes:
                    if k not in secuencia:
                        tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1 #los pacientes que no estén en la sec, igualmente no se van a procesar y se les añade un día a su tiempo acum
                for k in range(len(secuencia)): #x es el indice del paciente
                    # print(' se ve al paciente {}'.format(secuencia[k]))
                    if estado[indice(pacientes,secuencia[k])]==0:
                        retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                        if retorno!='NaN':
                            pacientesconsultas[retorno[0]].append(secuencia[k])
                            ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                            hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                            tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el +1 porque el día 0 aumenta 1 día
                            flagcambiado=1
                            atendidos=atendidos+1
                        else:
                            flagcambiado=0

                tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                        if retorno!='NaN':
                            cirujanosor[retorno[0]].append(retorno[1])
                            hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
```

```
                    pacientesor[retorno[0]].append(secuencia[k])
                    ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                    cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    if retorno[2]!='NaN':
                      asistentesor[retorno[0]].append(retorno[2])
                      hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                    if retorno[2]=='NaN':
                      asistentesor[retorno[0]].append(retorno[2])
                    flagcambiado=1
                    atendidos=atendidos+1
                  else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1 #se actualiza el tiempo acumulado.
                if estado[indice(pacientes,secuencia[k])]==2:
                  retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal]
                  if retorno!='NaN':
                    pacientesconsultas[retorno[0]].append(secuencia[k])
                    ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dc[indice(pacientes,secuencia[k])])
                    #habría que eliminar al paciente de la lista
                    hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                    flagcambiado=1
                    atendidos=atendidos+1
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el hospital#
                    pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                  else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or estado[indice(pacientes,secuencia[k])]==1):
                  estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                  if estado[indice(pacientes,secuencia[k])]==1:
                    mu=random.randint(1,4)
                    sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                    dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

              eq=equilibrio(hcc,hcor)
              tarde=lateness(pacientes)
              utilizacionor=0
              utilizacionc=0
              i=0

              while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
                i=i+1

              i=0
              while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
```

```
            i=i+1
        utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
        return     eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion

    #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
    # #creo las listas#
    leerexcel()
     #auxiliares#
    ctconsultas = [[0]]
    x=0
    while x<numc-1:
        ctconsultas.append([0])
        x=x+1

    ctor = [[0]]
    x=0
    while x<numor-1:
        ctor.append([0])
        x=x+1

    global dia
    leerexcel()
    vectorFO=asignacion(secuencia,cc)

    listaciruj[dia] = cc[:numc] + cirujanosor
    listaasist[dia] = empty+asistentesor
    listapacs[dia] = pacientesconsultas+pacientesor

    return vectorFO

def cajanegrasecuencia(secuencia):
    global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
    global listaciruj, listaasist, listapacs
    def asignacion(secuencia):
        global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor,
ctconsultas, ctor, hcor, hcc
        pacientesconsultas = [[]]
        cirujanosconsultas=[[]]
        x=0
        while x<numc-1:
            pacientesconsultas.append([])
            cirujanosconsultas.append([])
            x=x+1

        pacientesor = [[]]
        cirujanosor = [[]]
        asistentesor = [[]]
        x=0
        while x<numor-1:
            pacientesor.append([])
            cirujanosor.append([])
            asistentesor.append([])
            x=x+1

        ###FUNCIONES###
```

```
#Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
    #sirve para medir el equilibrio de la carga de trabajo#
def equilibrio(hcc,hcor):
  return (hcc.var()+hcor.var())

#Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
def lateness(pacientes):
  global pesos
  x=0
  late=[]
  for x in range(len(pacientes)):
    if estado[x]==0:
      if tiempoacum[x]>trp[x]:
        late.append((tiempoacum[x]-trp[x])*pesos[x])
    if estado[x]==1:
      if tiempoacum[x]>trq[x]:
        late.append((tiempoacum[x]-trq[x])*pesos[x])
    if estado[x]==2:
      if tiempoacum[x]>trc[x]:
        late.append((tiempoacum[x]-trc[x])*pesos[x])
  return sum(late) #late indica cuántos pacientes van con retraso actualmente

#Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
def asignadodia(cirujano,consultaor):
  global cirujanosconsultas
  global cirujanosor
  global numc
  i=0
  flag = 0
  if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
    for i in range(len(cirujanosconsultas)):
      if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
        return i
        flag=1
    if flag==0:
      return 'NaN'
  elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
    for i in range(numor):
      if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
        return i
        flag=1
    if flag==0:
      return 'NaN'

#funcion para calcular el indice de un vector
def indice(vector,valor):
  x=0
  for x in range(len(vector)):
    if vector[x]==valor:
      return x
      break

#Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
def busquedaPAE (paciente):
```

```
                global cirujanos
                global numc
                global hcc
                j=0
                cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
                while j<len(cirujanos):
                  if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la
unidad
                    cirujanosdelaunidad.append(j)
                  j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosdelaunidad)):
                  horas.append(hcc[cirujanosdelaunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
                np.argsort(horasnumpy)
                cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
                j=0
                flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la
semana
                #para cada cirujano, miramos si está asignado a alguna consulta el dia
                while j<len(cirujanosdelaunidad):

                  asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                  if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta
asignadoenconsulta
                    # print('si, está asignado el dia {} en la consulta {}'.format(dia,asignadoenconsulta))
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al
dia
                      # print('sí hay hueco en la consulta')
                      return [asignadoenconsulta,cirujanosdelaunidad[j]]
                      flagyatieneconsulta=1
                      break
                  if flagyatieneconsulta==1:break
                  j=j+1
                if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le
asigna al primero la primera consulta libre
                  j=0

                  while j<len(cirujanosconsultas):
                    if cirujanosconsultas[j]==[]:#si la consulta está libre
                      x=0
                      for x in cirujanosdelaunidad:
                        if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or
asignadodia(x, 'consulta', dia)==j):
                          #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa
misma,
                          # print('entroaqui')
                          return [j,x]
                          flagyatieneconsulta=1
                          break#cuando se le asigna, se sale del for
                      if flagyatieneconsulta==1:
                        break
                    j=j+1
                if flagyatieneconsulta==0:
```

```
            return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

        # Función para asignar un paciente y un cirujano a consulta postop
        def busquedaPOSTOP (paciente):
            #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

            flagyatieneconsulta=0

            asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
            asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
            if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                    return [asignadoenconsulta,cirjasoc[paciente]]
                    flagyatieneconsulta=1
            if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                #se le asigna el primer hueco que haya en la semana
                j=0

                while j<len(cirujanosconsultas):
                    if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                        if asignadoenconsulta=='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                            return [j,cirjasoc[paciente]]
                            flagyatieneconsulta=1
                            break#cuando se le asigna, se sale del for
                    if flagyatieneconsulta==1:
                        break
                    j=j+1
            if flagyatieneconsulta==0:
                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

        #Función para asignar un paciente y un equipo de cirujanos a quirófanos
        def busquedaOR(paciente):
            #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                #la experiencia y nivel de dificultad de la  operación.
            j=0
            cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
            while j<len(cirujanos):
                if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                    cirujanosexpunidad.append(cirujanos[j])
                j=j+1
            #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosexpunidad)):
                horas.append(hcor[cirujanosexpunidad[j]])
            horasnumpy=np.array(horas)
            cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
            np.argsort(horasnumpy)
            cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
```

```
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                  if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                    asistexp.append(cirujanos[j])
                    # print(asistexp)
                  j=j+1
                # print(asistexp)
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                  # print('asistente',asistexp[j] )
                  horas.append(hcor[asistexp[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenadaasist',asistexp)

                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                  # print('No se necesita asistente')
                  j=0

                  flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                  #para cada cirujano, miramos si está asignado a algún OR el dia
                  while j<len(cirujanosexpunidad):
                    asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                    # print(' ¿está el cirujano en algun OR el dia {} ?'.format(dia))
                    if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                      # print('si, está asignado en el OR {}'.format(asignadoenOR))
                      if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                        # print('hay hueco en este OR')
                        return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                        flagyatieneOR=1
                        break
                    j=j+1
                  if flagyatieneOR==0:#si no tiene OR en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                    # print('entra')
                    j=0
                    while j<numor:
                      # print('los cirujanos no tienen consulta asignada con hueco en la semana, así que se coge el primer hueco libre y se ve si algun ciruj puede')
                      if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                        x=0
                        for x in cirujanosexpunidad:
                          # print('probamos con el ciruj',x)
                          # print('asignadodia',asignadodia(x, 'consulta'))
```

```
                                # print('asignadodOR',asignadodia(x, 'OR'))
                                if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                    # print('este ciruj SI puede')
                                    return [j,cirujanosexpunidad[0],'NaN']
                                    flagyatieneOR=1
                                    break#cuando se le asigna, se sale del for
                                if flagyatieneOR==1:
                                    break
                                j=j+1
                            if flagyatieneOR==0:
                                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                        else:   #sí se necesita un asistente
                            # print('si se necesita un asistente')
                            #vamos a buscar a la vez un cirujano principal y un asistente
                            i=0
                            flagyatieneasist=0
                            for i in cirujanosexpunidad:
                                # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                                asignadoenOR = asignadodia(i,'OR')
                                # print('¿está asignado en algun OR ya?')
                                if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                                    # print('si, esta en el OR',asignadoenOR)
                                    if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                        # print('sí habría hueco en su OR')
                                        #buscamos un asistente
                                        k=0
                                        for k in asistexp:
                                            # print('probamos con asist',k)
                                            # print('asignado en consulta',asignadodia(k,'consulta'))
                                            # print('asignado en OR',asignadodia(k,'OR')=='NaN')
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
                                                # print('retornamos')
                                                return [asignadoenOR,i,k]
                                                flagyatieneasist=1
                                                break
                                    if flagyatieneasist==1:break
                            if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                                #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                                # print('no hay ciruj con OR asignado, probamos a buscar el primer hueco')
                                i=0
                                for i in cirujanosexpunidad:
                                    asignadoenconsulta = asignadodia(i,'consulta')
                                    # print('¿está asignado en consulta este ciruj ya?',asignadoenconsulta)
                                    x=0
                                    while x<numor:
                                        # print('está ya asignado a un OR?',asignadodia(i,'OR'))
                                        if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                            #print(ctor[dia][x][-1])
                                            if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
```

```
                                #buscamos un asistente
                                   # print('hay hueco')
                                   for k in asistexp:
                                      # print('probamos con el asist {}, cirujanos {}, asignado en consulta? {} asignado en OR? {}'.format(k,i,asignadodia(k,'consulta'),asignadodia(k,'OR')))
                                      if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                         # print('asistente disponible')
                                         #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                         #si esta condicion se cumple, se asocia este asist
                                         return [x,i,k]
                                         flagyatieneasist=1
                                         break
                                   if flagyatieneasist==1:break
                                   x=x+1
                                if flagyatieneasist==1:break
                             if flagyatieneasist==0:
                                # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                                return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                                #significa que no hay hueco posible para este paciente en OR esta semana
                          ###COMIENZO CODIGO###

                          global pacientesaborrar
                          atendidos=0
                          pacientesaborrar=[]
                          for k in pacientes:
                             if k not in secuencia:
                                tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
                          for k in range(len(secuencia)): #x es el indice del paciente
                             # print(' se ve al paciente {}'.format(secuencia[k]))
                             if estado[indice(pacientes,secuencia[k])]==0:
                                # print('vamos a busquedaPAE ')
                                retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                                # print(retorno)
                                if retorno!='NaN':
                                   cirujanosconsultas[retorno[0]]=[retorno[1]]
                                   pacientesconsultas[retorno[0]].append(secuencia[k])
                                   ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                                   hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                                   tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1 día
                                   # print(equilibrio(hcc,hcor))
                                   # print('se asigna el paciente {} a la consulta {} el día  con el cirujano {}'.format(secuencia[x],retorno[0],retorno[1]))
                                   flagcambiado=1
                                   atendidos=atendidos+1
                                else:
                                   flagcambiado=0

                          tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                          if estado[indice(pacientes,secuencia[k])]==1:
                             # print('vamos a OR')
                             retorno=busquedaOR(indice(pacientes,secuencia[k]))  #me devuelve [consulta,cirujprincipal,asistente]
                             # print(retorno)
```

```
                    if retorno!='NaN':
                        # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente
{}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
                        cirujanosor[retorno[0]].append(retorno[1])
                        hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                        pacientesor[retorno[0]].append(secuencia[k])
                        ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                        cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        if retorno[2]!='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                            hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                        if retorno[2]=='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==2:
                    # print('vamos a POSTOP')
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k]))          #me devuelve
[consulta,cirujprincipal]
                    # print('retorno',retorno)
                    if retorno!='NaN':
                        cirujanosconsultas[retorno[0]]=[retorno[1]]
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                        #habría que eliminar al paciente de la lista
                        hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                        flagcambiado=1
                        atendidos=atendidos+1
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                        pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)
            utilizacionor=0
            utilizacionc=0
            i=0
```

```
            while i<numc:
               utilizacionc=ctconsultas[i][-1]+utilizacionc
               i=i+1

            i=0
            while i<numor:
               utilizacionor=ctor[i][-1]+utilizacionor
               i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return     eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion

         #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
            # #creo las listas#
            leerexcel()
            ctconsultas = [[0]]
            x=0
            while x<numc-1:
               ctconsultas.append([0])
               x=x+1

            ctor = [[0]]
            x=0
            while x<numor-1:
               ctor.append([0])
               x=x+1

            global dia
            leerexcel()
            vectorFO=asignacion(secuencia)

            listaciruj[dia] = cirujanosconsultas+cirujanosor
            listaasist[dia] = empty+asistentesor
            listapacs[dia] = pacientesconsultas+pacientesor

            return vectorFO

         def actualizarexcel(pacientesaborrar):
            #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
            j=0
            x=0
            # print('pacientes a borrar', pacientesaborrar)
            while j < len(pacientesaborrar):
               x=0
               # print('j',j)
               while x<len(pacientes):
                  # print('x',x)
                  if pacientesaborrar[j]==pacientes[x]:
                     # print('se borra el paciente', pacientesaborrar[j])
                     del pacientes[x]
                     del cirjasoc[x]
                     del dc[x]
                     del dp[x]
                     del dq[x]
                     del estado[x]
                     del hp[x]
                     del hs[x]
```

```
            del pesos[x]
            del tiempoacum[x]
            del trc[x]
            del trp[x]
            del trq[x]
            del u[x]
          else:x=x+1
        j=j+1

    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                            'dp':dp,
                            'trp':trp,
                            'dq':dq,
                            'trq':trq,
                            'u':u,
                            'hp':hp,
                            'hs':hs,
                            'dc':dc,
                            'trc':trc,
                            'w':pesos,
                            'estado':estado,
                            'cirujano':cirjasoc,
                            'tacum':tiempoacum
                            })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                            'G':exp,
                            'S':uc
        })
    with pd.ExcelWriter('data.xlsx') as writer:
        nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
        datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
        datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):

    c=0
    C=[]
    while c<numc:
        C.append('C')
        c=c+1
    o=0
    O=[]
    while o<numor:
        O.append('O')
        o=o+1
    dfsol1 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[0],
                    'ASISTENTES':listaasist[0],
                    'PACIENTES':listapacs[0]

        })
    dfsol2 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[1],
                    'ASISTENTES':listaasist[1],
                    'PACIENTES':listapacs[1]

        })
    dfsol3 = pd.DataFrame({'Espacio':C+O,
```

```
                'CIRUJANOS':listaciruj[2],
                'ASISTENTES':listaasist[2],
                'PACIENTES':listapacs[2]

    })
dfsol4 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[3],
                'ASISTENTES':listaasist[3],
                'PACIENTES':listapacs[3]

    })
dfsol5 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[4],
                'ASISTENTES':listaasist[4],
                'PACIENTES':listapacs[4]

    })
dfsol6 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[5],
                'ASISTENTES':listaasist[5],
                'PACIENTES':listapacs[5]

    })
dfsol7 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[6],
                'ASISTENTES':listaasist[6],
                'PACIENTES':listapacs[6]

    })

with pd.ExcelWriter('solucion.xlsx') as writer:
    dfsol1.to_excel(writer, sheet_name = "LUNES")
    dfsol2.to_excel(writer, sheet_name = "MARTES")
    dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
    dfsol4.to_excel(writer, sheet_name = "JUEVES")
    dfsol5.to_excel(writer, sheet_name = "VIERNES")
    dfsol6.to_excel(writer, sheet_name = "SABADO")
    dfsol7.to_excel(writer, sheet_name = "DOMINGO")



#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")
numor = datos.loc[0,'Número de quirófanos']
# print('Número de quirófanos en el hospital', numor)
numc = datos.loc[0,'Número de consultas']
# print('Número de consultas en el hospital', numc)
#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
```

```
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
   aux.append([])
   x=x+1
x=0
listaciruj=[]
listaasist=[]
listapacs=[]
while x<7:
   listaciruj.append(aux)
   listaasist.append(aux)
   listapacs.append(aux)
   x=x+1

empty =[[]]
x=0
while x<numc-1:
   empty.append([])
   x=x+1

dia=0
while dia<1:

   inicio=time.time()
   #NEH#
   #1- ORDENAR LA SECUENCIA INICIAL
   sumatiempos=np.array([0.0]*len(pacientes))
   for x in range(len(pacientes)):
      # print(x, dc[x], dp[x], dq[x])
      sumatiempos[x]=dc[x]+dq[x]+dp[x]
   pacientesnumpy = np.array(pacientes)
   ordennumpy = np.array(sumatiempos)
   np.argsort(sumatiempos)
   secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
   secinicial1=np.flip(secinicial0)
   secinicial=secinicial1.tolist()
   #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
   #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
   # print(secinicial)
   pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
   FOpi=cajanegrasecuencia(pi)
   hcc=hcccopia.copy()#para que las pruebas no alteren el real
   hcor=hcorcopia.copy()

   piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
   FOpiinicial=cajanegrasecuencia(piinicial)
   hcc=hcccopia.copy()#para que las pruebas no alteren el real
   hcor=hcorcopia.copy()
   print('FOpiinicial antes de NEH', FOpiinicial)
```

```
j=1
while j<len(secinicial):

    i=0
    secprobando=pi[:]
    # print('secuencia que estamos probando', secprobando)
    flagcambiado=0
    while i<=j:
        secprobando.insert(i, secinicial[j])
        # print('secuencia probando, insertamos en la posición', secprobando,i)

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOprobando=cajanegrasecuencia(secprobando)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOprobando,FOpi):

            pi=secprobando.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()


            flagcambiado=1
         #no se ha añadido ningún número a la secuencia y terminamos el bucle
        secprobando.pop(i)
        i=i+1
    if flagcambiado==0: #si no ha mejorado, se inserta el paciente en última posición
        secprobando.insert(i-1,secinicial[j])
        pi=secprobando[:]
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    j=j+1
    #fin del NEH. Tenemos FOpi y pi
print('NEH tarda', time.time()-inicio)

if mejora(FOpi,FOpiinicial):#si el NEH ha mejorado la secuencia que teniamos inicialmente
    pibest=pi.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(pibest)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
else:
    pibest=piinicial.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpibest=cajanegrasecuencia(piinicial)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
```

```
        print('FOpibest con la secuencia tras NEH y antes de LS', FOpibest)


        Temp=1
        inicio=time.time()
        while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:#tiempo de parada según TFG MARIANGELES
            piprima=pi.copy()
            destruida=destruction(piprima)#destruction phase
            inicio1=time.time()
            piprima=construction(destruida[0],destruida[1]) #construction phase
            print('construccion tarda',time.time()-inicio1)

            #LOCAL SEARCH#
            iniciols=time.time()
            n=len(piprima)
            i=0
            j=0
            piprimaprima=piprima.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpiprimaprima=cajanegrasecuencia(piprimaprima)
            print('FO tras construccion', FOpiprimaprima)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
            while j<n:
                i=0
                while i<n:
                    secprima=piprima.copy()
                    secprima.remove(piprima[j])
                    secprima.insert(i,piprima[j])
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOsecprima=cajanegrasecuencia(secprima)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    if mejora(FOsecprima,FOpiprimaprima):
                        piprimaprima=secprima.copy()
                        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                        FOpiprimaprima=cajanegrasecuencia(piprimaprima)
                        hcc=hcccopia.copy()#para que las pruebas no alteren el real
                        hcor=hcorcopia.copy()

                    i=i+1
                j=j+1
            #FIN LOCAL SEARCH
            print('LS tarda',time.time()-iniciols)
            if mejora(FOpiprimaprima,FOpi):
                pi=piprimaprima.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpi=cajanegrasecuencia(pi)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
                if mejora(FOpi,FOpibest):
```

```
                pibest=pi.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpibest=cajanegrasecuencia(pibest)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
            elif random.uniform(0, 1)<=-math.exp(-(FOpiprimaprima[1]-FOpi[1])/Temp):
                print('se acepta peor solucion')
                pi=piprimaprima.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpi=cajanegrasecuencia(pi)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
            Temp=0.99*Temp

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpibest=cajanegrasecuencia(pibest)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
        print('FOpibest tras LS', FOpibest)

        #aquí ya tendríamos IG para la secuencia
        #aplicamos ahora el IG para la asignación de cirujanos
        ccsec=cirujanosconsultas.copy()#Se parte de esta asignación para realizar e NEH


        ccinicial=ccsec[:]
        for x in cirujanos:
            if [x] not in ccinicial and descanso[x][dia]==0:
                ccinicial.append([x])

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOcc=cajanegra(pibest,ccsec)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        #SE NIVELA PARA QUE LOS CIRUJANOS QUE HAN SIDO ASIGNADOS A CONSULTAS Y AL FINAL SE QUEDAN SIN PACIENTES NO SE TENGAN EN CUENTA Y PUEDAN SER ASIGNADOS A OR
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccsec[:numc])
        FOcc=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()


        cc=[ccinicial[0]]
        j=1
        #NEH#
        while j<len(ccinicial):
            i=0
            ccprobando=cc[:]
            flagcambiado=0
            while i<=j:
```

```
            ccprobando.insert(i,ccinicial[j])
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegra(pibest,ccprobando[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,ccprobando[:numc])
            FOprobando=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOprobando,FOcc):
                cc=ccprobando[:]
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOcc=cajanegra(pibest,cc[:numc])
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                s=nivelar(pacientesconsultas,cc[:numc])
                FOcc=cajanegra(pibest,s)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                flagcambiado=1
            ccprobando.pop(i)
            i=i+1
        if flagcambiado==0:
            ccprobando.insert(i-1,ccinicial[j])
            cc=ccprobando[:]

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

        j=j+1
        #FIN NEH#

    FOccbest=FOcc[:]
    ccbest=cc.copy()

    print('Foccbest con el que se entra a LS', FOccbest)
    it=0
    Temp=1
```

```
inicio=time.time()
while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
    # ccprima=cc.copy()
    # FOccprima=FOcc[:]
    #destruccion
    ccdestruida=destruccion(ccbest)
    ccprima=construccion(ccdestruida[0],ccdestruida[1])
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOccprima=cajanegra(pibest,ccprima[:numc])
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    s=nivelar(pacientesconsultas,ccprima[:numc])
    FOccprima=cajanegra(pibest,s)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()


    #LOCAL SEARCH#

    n=len(ccprima)
    i=0
    j=0
    ccprimaprima=ccprima.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOccprimaprima=cajanegra(pibest,ccprimaprima[:numc]) #es FOccprima
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    s=nivelar(pacientesconsultas,ccprimaprima[:numc])
    FOccprimaprima=cajanegra(pibest,s)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    while j<n:
        i=0
        while i<n:
            cirujanosprima=ccprima.copy()
            cirujanosprima.remove(ccprima[j])
            cirujanosprima.insert(i,ccprima[j])
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcirujanosprima=cajanegra(pibest,cirujanosprima[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cirujanosprima[:numc])
            FOcirujanosprima=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
```

```
        if mejora(FOcirujanosprima,FOccprimaprima):
            ccprimaprima=ccprima.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOccprimaprima=cajanegra(pibest,ccprimaprima[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,ccprimaprima[:numc])
            FOccprimaprima=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
        i=i+1
        if i>numc:#cuandobya el cirujano es asignado a consultas inexistentes, break
            break
    j=j+1

#FIN LOCAL SEARCH

if mejora(FOccprimaprima,FOcc):
    cc=ccprimaprima.copy()

    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOcc=cajanegra(pibest,cc[:numc])
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    s=nivelar(pacientesconsultas,cc[:numc])
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOcc=cajanegra(pibest,s)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    if mejora(FOcc,FOccbest):
        ccbest=cc.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccbest=cajanegra(pibest,ccbest[:numc])
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        s=nivelar(pacientesconsultas,ccbest[:numc])

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccbest=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

elif random.uniform(0, 1)<=-math.exp(-(FOccprima[1]-FOcc[1])/Temp):
    print('se acepta peor solucion')
    cc=ccprima.copy()
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
```

```
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOcc=cajanegra(pibest,cc)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
            Temp=0.99*Temp
            it=it+1

        #aquí ya tendríamos FOccbest y ccbest, que serían las mejores soluciones tras el IG

            print('FO con los cirujs tras LS', FOccbest)
            secuencia=pibest.copy()
            cirujanosconsultas=ccbest.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FO=cajanegra(secuencia, cirujanosconsultas[:numc]) #FO DEFINITIVA#
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            s=nivelar(pacientesconsultas,cirujanosconsultas[:numc])
            FO=cajanegra(secuencia,s)
            print('MEJOR FO DEFINITIVA',FO)
            print('ctor',ctor)
            print('ctconsultas', ctconsultas)
            sumaFO=sumaFO+FO[3]
            actualizarexcel(pacientesaborrar)
            dia=dia+1

        escribirsol(listaciruj, listaasist, listapacs)
        return sumaFO

    def IG2():
        global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        global sumaFO

        def insertar(lista,pos1,pos2):
            copia=lista.copy()
            a=copia.pop(pos1)
            copia.insert(pos2,a)
            return copia

        def mejora(v1probando,v2best):
            if v1probando[3]<v2best[3]:
                return True
            else:
                return False

        def nivelar(patients,surgeons):

            j=min(numc-1,len(surgeons)-1)
            while j>=0:
                if patients[j]==[]:
                    surgeons.pop(j)
                    patients.pop(j)
                    surgeons.insert(numc-1,[])
                    patients.insert(numc-1,[])
```

```
                j=j-1
            return surgeons

        def destruction(sequence):#la secuencia que se le pasará será secbest
            n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
            # print(n)
            a=0
            secborrar=sequence[:]
            while a<len(n):
                secborrar.remove(n[a])
                a=a+1
            return secborrar, n
        def construction(seqbest, ccfija, n): #para secuenncia
            seqprobando=seqbest[:]
            a=0
            while a<len(n):
                seqprobando=seqbest[:]
                seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para luego ir tomando las que la mejoren
                seqbest=seqprobando[:]
                global hcc, hcor
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FObest=cajanegra(seqbest,ccfija)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                s=nivelar(pacientesconsultas,ccfija[:numc])
                FObest=cajanegra(seqbest,s)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                # print('la primera secuencia es', secbest, FObest[1])
                seqprobando.pop(0)
                ###PROBANDO PARA NO INSERTAR DOS VECES EL MISMO NOMBRE###
                b=1
                while b<len(seqbest):
                    # print('i',i)
                    seqprobando.insert(b, n[a])
                    # print('secuencia probando, insertamos en la posición', secprobando,i)
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOprobando=cajanegra(seqprobando, ccfija)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    s=nivelar(pacientesconsultas,ccfija[:numc])
                    FOprobando=cajanegra(seqprobando,s)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    # print('FO=',FOprobando[1])
                    if mejora(FOprobando,FObest):
                        # print('mejora')
```

```
                    seqbest=seqprobando[:]
                    FObest=FOprobando[:]
                  #no se ha añadido ningún número a la secuencia y terminamos el bucle
                  seqprobando.pop(b)
                  b=b+1
               a=a+1
            return seqbest

         def destruccion(cirujanosconsults): #para asignacion cirujanos a consultas
            n=random.sample(cirujanosconsults,4)
            a=0
            ccborrar=cirujanosconsults[:]
            while a<len(n):
               ccborrar.remove(n[a])
               a=a+1
            return ccborrar, n

         def construccion(ccmejor,n): #para asignacion cirujanos a consultas
            global hcc, hcor
            ccprobando=ccmejor[:]
            a=0
            while a<len(n):
               ccprobando=ccmejor[:]
               ccprobando.append(n[a])
               ccmejor=ccprobando[:]

               hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
               hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
               FObest=cajanegra(pibest,ccmejor)
               hcc=hcccopia.copy()#para que las pruebas no alteren el real
               hcor=hcorcopia.copy()

               ccprobando.pop(-1)
               b=0
               while b<len(ccmejor):
                  ccprobando.insert(b,n[a])

                  hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                  hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                  FOprobando=cajanegra(pibest,ccprobando)
                  hcc=hcccopia.copy()#para que las pruebas no alteren el real
                  hcor=hcorcopia.copy()

                  if mejora(FOprobando,FObest):
                     # print('mejora')
                     ccmejor=ccprobando[:]
                     FObest=FOprobando[:]
                  ccprobando.pop(b)
                  b=b+1
               a=a+1
            return ccmejor


         def leerexcel():
            global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado, cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc

            datospacientes = pd.read_excel("data.xlsx", "Pacientes")
```

```
        pacientes = datospacientes['Número paciente'].tolist()

        dp = datospacientes['dp'].tolist()

        trp = datospacientes['trp'].tolist()

        dq = datospacientes['dq'].tolist()

        trq = datospacientes['trq'].tolist()

        u = datospacientes['u'].tolist()

        hp = datospacientes['hp'].tolist()

        hs = datospacientes['hs'].tolist()

        dc = datospacientes['dc'].tolist()

        trc = datospacientes['trc'].tolist()

        pesos = datospacientes['w'].tolist()

        estado = datospacientes['estado'].tolist()

        cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
        cirjasoc = [int(x) for x in cirjasoc1.tolist()]
        for x in range(len(cirjasoc)):
            if cirjasoc[x]==999:
                cirjasoc[x]='NaN'

        tiempoacum = datospacientes['tacum'].tolist()
    def cajanegra(secuencia,cc):

        global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        def asignacion(secuencia,cc):
            global pacientesconsultas, pacientesor, cirujanosor, asistentesor, ctconsultas, ctor, hcor, hcc
            pacientesconsultas = [[]]
            x=0
            while x<numc-1:
                pacientesconsultas.append([])
                x=x+1

            pacientesor = [[]]
            cirujanosor = [[]]
            asistentesor = [[]]
            x=0
            while x<numor-1:
                pacientesor.append([])
                cirujanosor.append([])
                asistentesor.append([])
                x=x+1

            global eq
            ###FUNCIONES###
```

```
#Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
    #sirve para medir el equilibrio de la carga de trabajo#
def equilibrio(hcc,hcor):
  return (hcc.var()+hcor.var())

#Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta, ponderado con los pesos
def lateness(pacientes):
  global pesos
  x=0
  late=[]
  for x in range(len(pacientes)):
    if estado[x]==0:
      if tiempoacum[x]>trp[x]:
        # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
        late.append((tiempoacum[x]-trp[x])*pesos[x])
    if estado[x]==1:
      if tiempoacum[x]>trq[x]:
        # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
        late.append((tiempoacum[x]-trq[x])*pesos[x])
    if estado[x]==2:
      if tiempoacum[x]>trc[x]:
        # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
        late.append((tiempoacum[x]-trc[x])*pesos[x])
  # print('Número de pacientes que van tarde:')
  return sum(late) #late indica cuántos pacientes van con retraso actualmente

#Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
def asignadodia(cirujano,consultaor):
  # global cirujanosconsultas
  global cirujanosor
  global numc
  i=0
  flag = 0
  if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
    for i in range(len(cc[:numc])):
      if cirujano in cc[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
        return i
        flag=1
    if flag==0:
      return 'NaN'
      # print("El cirujano no está asignado a ninguna consulta el dia ",dia)
  elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
    for i in range(numor):
      if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
        return i
        flag=1
    if flag==0:
      return 'NaN'
      # print("El cirujano no está asignado a ningún quirófano el dia ",dia)

#funcion para calcular el indice de un vector
def indice(vector,valor):
  x=0
  for x in range(len(vector)):
    if vector[x]==valor:
```

```
                    return x
                    break

        #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
        def busquedaPAE (paciente):
            global cirujanos
            global numc
            global hcc
            j=0
            cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
            while j<len(cirujanos):
                if uc[j]==u[paciente]:#si el cirujano que estamos viendo es de la unidad
                    cirujanosdelaunidad.append(j)
                j=j+1
            # print(cirujanosdelaunidad)
            #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosdelaunidad)):
                horas.append(hcc[cirujanosdelaunidad[j]])
            # print(horas)
            horasnumpy=np.array(horas)
            cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
            np.argsort(horasnumpy)
            cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
            # print('lista de cirujanos posibles ordenadas',cirujanosdelaunidad)#lista de cirujanos ordenadas según horas acumuladas en consulta
            j=0
            flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
            #para cada cirujano, miramos si está asignado a alguna consulta el dia
            while j<len(cirujanosdelaunidad):

                # print('¿esta el paciente asignado el dia {} a consulta?'.format(dia))
                asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
                    # print('si, está asignado el dia {} en la consulta {}'.format(dia,asignadoenconsulta))
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                        # print('sí hay hueco en la consulta')
                        return [asignadoenconsulta,cirujanosdelaunidad[j]]
                        flagyatieneconsulta=1
                        break

                if flagyatieneconsulta==1:break
                j=j+1
            if flagyatieneconsulta==0:
                # print('nop')
                return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


        # Función para asignar un paciente y un cirujano a consulta postop
        def busquedaPOSTOP (paciente):
            #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco
```

```
                flagyatieneconsulta=0
                # print('busqueda POSTOP para paciente',paciente)

                asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                # print('el cirujano está asignado en consulta este dia?',asignadoenconsulta)
                asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                # print('el cirujano está asignado en OR este dia?',asignadoenOR)
                if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                        # print('hay hueco en esa consulta, así que se añade ahi')
                        return [asignadoenconsulta,cirjasoc[paciente]]
                        flagyatieneconsulta=1
                if flagyatieneconsulta==0:
                    # print('no hay hueco posible')
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                #la experiencia y nivel de dificultad de la  operación.
                global hcor
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
                        # print(cirujanosexpunidad)
                    j=j+1
                # print('lista de cirujanos',cirujanosexpunidad)
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    # print('ciruj',cirujanosexpunidad[j])
                    horas.append(hcor[cirujanosexpunidad[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenada',cirujanosexpunidad)#lista de cirujanos ordenadas según horas acumuladas en quirófano. Esta es la lista con la que trabajaremos para obtener los posibles cirujanos
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
```

```
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                        # print(asistexp)
                    j=j+1
                # print(asistexp)
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    # print('asistente',asistexp[j] )
                    horas.append(hcor[asistexp[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenadaasist',asistexp)


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    # print('No se necesita asistente')
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                        asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                        # print(' ¿está el cirujano en algun OR el dia {} ?'.format(dia))
                        if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                            # print('si, está asignado en el OR {}'.format(asignadoenOR))
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                                # print('hay hueco en este OR')
                                return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                                flagyatieneOR=1
                                break
                        j=j+1
                    if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                        # print('entra')
                        j=0
                        while j<numor:
                            # print('los cirujanos no tienen consulta asignada con hueco en la semana, así que se coge el primer hueco libre y se ve si algun ciruj puede')
                            if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                                x=0
                                for x in cirujanosexpunidad:
                                    # print('probamos con el ciruj',x)
                                    # print('asignadodia',asignadodia(x, 'consulta'))
                                    # print('asignadodOR',asignadodia(x, 'OR'))
                                    if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                        # print('este ciruj SI puede')
                                        return [j,cirujanosexpunidad[0],'NaN']
                                        flagyatieneOR=1
                                        break#cuando se le asigna, se sale del for
```

```
                        if flagyatieneOR==1:
                            break
                        j=j+1
                    if flagyatieneOR==0:
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                else:   #sí se necesita un asistente
                    # print('si se necesita un asistente')
                    #vamos a buscar a la vez un cirujano principal y un asistente
                    i=0
                    flagyatieneasist=0
                    for i in cirujanosexpunidad:
                        # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                        asignadoenOR = asignadodia(i,'OR')
                        # print('¿está asignado en algun OR ya?')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                            # print('si, esta en el OR',asignadoenOR)
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                # print('sí habría hueco en su OR')
                                #buscamos un asistente
                                k=0
                                for k in asistexp:
                                    # print('probamos con asist',k)
                                    # print('asignado en consulta',asignadodia(k,'consulta'))
                                    # print('asignado en OR',asignadodia(k,'OR')=='NaN')
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        # print('retornamos')
                                        return [asignadoenOR,i,k]
                                        flagyatieneasist=1
                                        break
                        if flagyatieneasist==1:break
                    if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                        #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                        # print('no hay ciruj con OR asignado, probamos a buscar el primer hueco')
                        i=0
                        for i in cirujanosexpunidad:
                            asignadoenconsulta = asignadodia(i,'consulta')
                            # print('¿está asignado en consulta este ciruj ya?',asignadoenconsulta)
                            x=0
                            while x<numor:
                                # print('está ya asignado a un OR?',asignadodia(i,'OR'))
                                if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                    #print(ctor[dia][x][-1])
                                    if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                    #buscamos un asistente
                                        # print('hay hueco')
                                        for k in asistexp:
                                            # print('probamos con el asist {}, cirujanos {}, asignado en consulta? {} asignado en OR? {}'.format(k,i,asignadodia(k,'consulta'),asignadodia(k,'OR')))
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
```

```
                                    # print('asistente disponible')
                                    #asiste!=cirujasoc  asist no puede estar en consultas ese dia     asist no puede estar asignado a otro OR que no sea este ese dia
                                    #si esta condicion se cumple, se asocia este asist
                                    return [x,i,k]
                                    flagyatieneasist=1
                                    break
                                if flagyatieneasist==1:break
                                x=x+1
                            if flagyatieneasist==1:break
                        if flagyatieneasist==0:
                            # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                            return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                            #significa que no hay hueco posible para este paciente en OR esta semana


                    ###COMIENZO CODIGO###

                    global pacientesaborrar
                    atendidos=0
                    pacientesaborrar=[]
                    for k in pacientes:
                        if k not in secuencia:
                            tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1 #los pacientes que no estén en la sec, igualmente no se van a procesar y se les añade un día a su tiempo acum
                    for k in range(len(secuencia)): #x es el indice del paciente
                        # print(' se ve al paciente {}'.format(secuencia[k]))
                        if estado[indice(pacientes,secuencia[k])]==0:
                            # print('vamos a busquedaPAE ')
                            retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                            # print(retorno)
                            if retorno!='NaN':
                                pacientesconsultas[retorno[0]].append(secuencia[k])
                                ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                                hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                                tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el +1 porque el día 0 aumenta 1 día
                                # print(equilibrio(hcc,hcor))
                                # print('se asigna el paciente {} a la consulta {} el día  con el cirujano {}'.format(secuencia[x],retorno[0],retorno[1]))
                                flagcambiado=1
                                atendidos=atendidos+1
                            else:
                                flagcambiado=0

                        tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                        if estado[indice(pacientes,secuencia[k])]==1:
                            # print('vamos a OR')
                            retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                            # print(retorno)
                            if retorno!='NaN':
                                # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente {}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
                                cirujanosor[retorno[0]].append(retorno[1])
                                hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
```

```
                    pacientesor[retorno[0]].append(secuencia[k])
                    ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                    cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    if retorno[2]!='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                        hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                    if retorno[2]=='NaN':
                        asistentesor[retorno[0]].append(retorno[2])
                    flagcambiado=1
                    atendidos=atendidos+1
                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1 #se actualiza el tiempo acumulado.
            if estado[indice(pacientes,secuencia[k])]==2:
                # print('vamos a POSTOP')
                retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal]
                # print('retorno',retorno)
                if retorno!='NaN':
                    pacientesconsultas[retorno[0]].append(secuencia[k])
                    ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dc[indice(pacientes,secuencia[k])])
                    #habría que eliminar al paciente de la lista
                    hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                    flagcambiado=1
                    atendidos=atendidos+1
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                    # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el hospital#
                    pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                else:
                    flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

            if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or estado[indice(pacientes,secuencia[k])]==1):
                estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==1:
                    mu=random.randint(1,4)
                    sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                    dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

        eq=equilibrio(hcc,hcor)
        tarde=lateness(pacientes)
        utilizacionor=0
        utilizacionc=0
        i=0

        while i<numc:
            utilizacionc=ctconsultas[i][-1]+utilizacionc
            i=i+1

        i=0
```

```
            while i<numor:
               utilizacionor=ctor[i][-1]+utilizacionor
               i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return     eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


         #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
         # #creo las listas#
         leerexcel()
          #auxiliares#
         ctconsultas = [[0]]
         x=0
         while x<numc-1:
            ctconsultas.append([0])
            x=x+1

         ctor = [[0]]
         x=0
         while x<numor-1:
            ctor.append([0])
            x=x+1

         global dia
         leerexcel()
         vectorFO=asignacion(secuencia,cc)

         listaciruj[dia] = cc+cirujanosor
         listaasist[dia] = empty+asistentesor
         listapacs[dia] = pacientesconsultas+pacientesor

         return vectorFO

      def cajanegrasecuencia(secuencia):
         global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
         global listaciruj, listaasist, listapacs
         def asignacion(secuencia):
            global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor,
ctconsultas, ctor, hcor, hcc
            pacientesconsultas = [[]]
            cirujanosconsultas=[[]]
            x=0
            while x<numc-1:
               pacientesconsultas.append([])
               cirujanosconsultas.append([])
               x=x+1

            pacientesor = [[]]
            cirujanosor = [[]]
            asistentesor = [[]]
            x=0
            while x<numor-1:
               pacientesor.append([])
               cirujanosor.append([])
               asistentesor.append([])
```

```
    x=x+1


###FUNCIONES###

#Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
        #sirve para medir el equilibrio de la carga de trabajo#
def equilibrio(hcc,hcor):
    return (hcc.var()+hcor.var())

#Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
def lateness(pacientes):
    global pesos
    x=0
    late=[]
    for x in range(len(pacientes)):
        if estado[x]==0:
            if tiempoacum[x]>trp[x]:
                # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                late.append((tiempoacum[x]-trp[x])*pesos[x])
        if estado[x]==1:
            if tiempoacum[x]>trq[x]:
                # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                late.append((tiempoacum[x]-trq[x])*pesos[x])
        if estado[x]==2:
            if tiempoacum[x]>trc[x]:
                # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                late.append((tiempoacum[x]-trc[x])*pesos[x])
    # print('Número de pacientes que van tarde:')
    return sum(late) #late indica cuántos pacientes van con retraso actualmente

#Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
def asignadodia(cirujano,consultaor):
    global cirujanosconsultas
    global cirujanosor
    global numc
    i=0
    flag = 0
    if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
        for i in range(len(cirujanosconsultas)):
            if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                return i
                flag=1
        if flag==0:
            return 'NaN'
            # print("El cirujano no está asignado a ninguna consulta el dia ",dia)
    elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
        for i in range(numor):
            if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR j, nos devuelve ese valor
                return i
                flag=1
        if flag==0:
            return 'NaN'
            # print("El cirujano no está asignado a ningún quirófano el dia ",dia)
```

```
#funcion para calcular el indice de un vector
def indice(vector,valor):
  x=0
  for x in range(len(vector)):
    if vector[x]==valor:
      return x
      break

#Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea necesario coincidir en unidades)
def busquedaPAE (paciente):
  global cirujanos
  global numc
  global hcc
  global cirujanosconsultas
  j=0
  cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
  while j<len(cirujanos):
    if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad
      cirujanosdelaunidad.append(j)
    j=j+1
  # print(cirujanosdelaunidad)
  #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
  j=0
  horas=[]
  for j in range(len(cirujanosdelaunidad)):
    horas.append(hcc[cirujanosdelaunidad[j]])
  # print(horas)
  horasnumpy=np.array(horas)
  cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
  np.argsort(horasnumpy)
  cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
  # print('lista de cirujanos posibles ordenadas',cirujanosdelaunidad)#lista de cirujanos ordenadas según horas acumuladas en consulta
  j=0
  flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
  #para cada cirujano, miramos si está asignado a alguna consulta el dia
  while j<len(cirujanosdelaunidad):

    # print('¿esta el paciente asignado el dia {} a consulta?'.format(dia))
    asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
    if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
      # print('si, está asignado el dia {} en la consulta {}'.format(dia,asignadoenconsulta))
      if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
        # print('sí hay hueco en la consulta')
        return [asignadoenconsulta,cirujanosdelaunidad[j]]
        flagyatieneconsulta=1
        break

    if flagyatieneconsulta==1:break
    j=j+1
  if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero la primera consulta libre
```

```
                    # print('el cirujano no tiene consulta asignada en toda la semana, así que le buscamos la primera consulta libre')
                    j=0

                    while j<len(cirujanosconsultas):
                        # print(' dia{} consulta {} '.format(dia,j))
                        # print('tiempo que quedaria',ctconsultas[dia][j][-1]+dp[paciente])
                        if cirujanosconsultas[j]==[]:#si la consulta está libre
                            x=0
                            for x in cirujanosdelaunidad:
                                # print('cirujano posible en este hueco',x)
                                # print('está asignado en OR? {} está asignado en consulta?{}'.format(asignadodia(x, 'OR', dia),asignadodia(x, 'consulta', dia)))
                                if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or asignadodia(x, 'consulta', dia)==j):
                                    #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa misma,
                                    # print('entroaqui')
                                    return [j,x]
                                    flagyatieneconsulta=1
                                    break#cuando se le asigna, se sale del for
                            if flagyatieneconsulta==1:
                                break
                        j=j+1
                    if flagyatieneconsulta==0:
                        # print('nop')
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


                # Función para asignar un paciente y un cirujano a consulta postop
                def busquedaPOSTOP (paciente):
                    #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                    flagyatieneconsulta=0
                    # print('busqueda POSTOP para paciente',paciente)

                    asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                    # print('el cirujano está asignado en consulta este dia?',asignadoenconsulta)
                    asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                    # print('el cirujano está asignado en OR este dia?',asignadoenOR)
                    if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                        if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                            # print('hya hueco en esa consulta, asi que se añade ahi')
                            return [asignadoenconsulta,cirjasoc[paciente]]
                            flagyatieneconsulta=1
                    if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                        #se le asigna el primer hueco que haya en la semana
                        # print('el cirujano no tiene consulta con hueco ninguna dia, asi que se le asigna en el primer hueco que haya en la semana')
                        j=0

                        while j<len(cirujanosconsultas):
                            # print('dia {} y consulta {}'.format(dia,j))
                            if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                                # print('hay hueco en esta consulta')
```

```
                        # print('vemos si el ciruj está asignado en algun OR ese día')
                        if asignadoenconsulta=='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                            return [j,cirjasoc[paciente]]
                            flagyatieneconsulta=1
                            break#cuando se le asigna, se sale del for
                        if flagyatieneconsulta==1:
                            break
                        j=j+1
                    if flagyatieneconsulta==0:
                        # print('no hay hueco posible')
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana



                #Función para asignar un paciente y un equipo de cirujanos a quirófanos
                def busquedaOR(paciente):
                    #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                        #la experiencia y nivel de dificultad de la  operación.
                    j=0
                    cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                    while j<len(cirujanos):
                        if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                            cirujanosexpunidad.append(cirujanos[j])
                            # print(cirujanosexpunidad)
                        j=j+1
                    # print('lista de cirujanos',cirujanosexpunidad)
                    #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                    j=0
                    horas=[]
                    for j in range(len(cirujanosexpunidad)):
                        # print('ciruj',cirujanosexpunidad[j])
                        horas.append(hcor[cirujanosexpunidad[j]])
                    # print(horas)
                    horasnumpy=np.array(horas)
                    cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                    np.argsort(horasnumpy)
                    cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                    # print('listaordenada',cirujanosexpunidad)#lista de cirujanos ordenadas según horas acumuladas en quirófano. Esta es la lista con la que trabajaremos para obtener los posibles cirujanos
                    #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente


                    ###LISTA ASISTENTES###
                    j=0
                    asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                    while j<len(cirujanos):
                        if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                            asistexp.append(cirujanos[j])
                            # print(asistexp)
                        j=j+1
```

```
                # print(asistexp)
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                   # print('asistente',asistexp[j] )
                   horas.append(hcor[asistexp[j]])
                # print(horas)
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()
                # print('listaordenadaasist',asistexp)


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                   # print('No se necesita asistente')
                   j=0

                   flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                   #para cada cirujano, miramos si está asignado a algún OR el dia
                   while j<len(cirujanosexpunidad):
                      asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                      # print(' ¿está el cirujano en algun OR el dia {} ?'.format(dia))
                      if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                         # print('si, está asignado en el OR {}'.format(asignadoenOR))
                         if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                            # print('hay hueco en este OR')
                            return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                            flagyatieneOR=1
                            break
                      j=j+1
                   if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                      # print('entra')
                      j=0
                      while j<numor:
                         # print('los cirujanos no tienen consulta asignada con hueco en la semana, así que se coge el primer hueco libre y se ve si algun ciruj puede')
                         if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                            x=0
                            for x in cirujanosexpunidad:
                               # print('probamos con el ciruj',x)
                               # print('asignadodia',asignadodia(x, 'consulta'))
                               # print('asignadodOR',asignadodia(x, 'OR'))
                               if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                  # print('este ciruj SI puede')
                                  return [j,cirujanosexpunidad[0],'NaN']
                                  flagyatieneOR=1
                                  break#cuando se le asigna, se sale del for
                            if flagyatieneOR==1:
                               break
                         j=j+1
                      if flagyatieneOR==0:
```

```
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                else:   #sí se necesita un asistente
                    # print('si se necesita un asistente')
                    #vamos a buscar a la vez un cirujano principal y un asistente
                    i=0
                    flagyatieneasist=0
                    for i in cirujanosexpunidad:
                        # print('probamos con el ciruj {} el dia {}'.format(i,dia))
                        asignadoenOR = asignadodia(i,'OR')
                        # print('¿está asignado en algun OR ya?')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                            # print('si, esta en el OR',asignadoenOR)
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                # print('sí habría hueco en su OR')
                                #buscamos un asistente
                                k=0
                                for k in asistexp:
                                    # print('probamos con asist',k)
                                    # print('asignado en consulta',asignadodia(k,'consulta'))
                                    # print('asignado en OR',asignadodia(k,'OR')=='NaN')
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        # print('retornamos')
                                        return [asignadoenOR,i,k]
                                        flagyatieneasist=1
                                        break
                            if flagyatieneasist==1:break
                    if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                        #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                        # print('no hay ciruj con OR asignado, probamos a buscar el primer hueco')
                        i=0
                        for i in cirujanosexpunidad:
                            asignadoenconsulta = asignadodia(i,'consulta')
                            # print('¿está asignado en consulta este ciruj ya?',asignadoenconsulta)
                            x=0
                            while x<numor:
                                # print('está ya asignado a un OR?',asignadodia(i,'OR'))
                                if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                    #print(ctor[dia][x][-1])
                                    if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                    #buscamos un asistente
                                        # print('hay hueco')
                                        for k in asistexp:
                                            # print('probamos con el asist {}, cirujanos {}, asignado en consulta? {} asignado en OR? {}'.format(k,i,asignadodia(k,'consulta'),asignadodia(k,'OR')))
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                                # print('asistente disponible')
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
```

```
                                return [x,i,k]
                                flagyatieneasist=1
                                break
                        if flagyatieneasist==1:break
                        x=x+1
                    if flagyatieneasist==1:break
            if flagyatieneasist==0:
                # print('NO HAY POSIBLE COMBINACION CIRUJANO ASISTENTE')
                return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                #significa que no hay hueco posible para este paciente en OR esta semana


        ###COMIENZO CODIGO###

        global pacientesaborrar
        atendidos=0
        pacientesaborrar=[]
        for k in pacientes:
            if k not in secuencia:
                tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
        for k in range(len(secuencia)): #x es el indice del paciente
            # print(' se ve al paciente {}'.format(secuencia[k]))
            if estado[indice(pacientes,secuencia[k])]==0:
                # print('vamos a busquedaPAE ')
                retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                # print(retorno)
                if retorno!='NaN':
                    cirujanosconsultas[retorno[0]]=[retorno[1]]
                    pacientesconsultas[retorno[0]].append(secuencia[k])
                    ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                    hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                    tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1 día
                    # print(equilibrio(hcc,hcor))
                    # print('se asigna el paciente {} a la consulta {} el día  con el cirujano {}'.format(secuencia[x],retorno[0],retorno[1]))
                    flagcambiado=1
                    atendidos=atendidos+1
                else:
                    flagcambiado=0

            tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
            if estado[indice(pacientes,secuencia[k])]==1:
                # print('vamos a OR')
                retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                # print(retorno)
                if retorno!='NaN':
                    # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente {}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
                    cirujanosor[retorno[0]].append(retorno[1])
                    hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                    pacientesor[retorno[0]].append(secuencia[k])
                    ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                    cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                    tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
```

```
                        if retorno[2]!='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                            hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                        if retorno[2]=='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==2:
                    # print('vamos a POSTOP')
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k]))      #me devuelve
[consulta,cirujprincipal]
                    # print('retorno',retorno)
                    if retorno!='NaN':
                        cirujanosconsultas[retorno[0]]=[retorno[1]]
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                        #habría que eliminar al paciente de la lista
                        hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                        flagcambiado=1
                        atendidos=atendidos+1
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                        pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])


                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)
            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
                i=i+1

            i=0
            while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
                i=i+1
```

```
        utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
        return  eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


    #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
    # #creo las listas#
    leerexcel()
    ctconsultas = [[0]]
    x=0
    while x<numc-1:
      ctconsultas.append([0])
      x=x+1

    ctor = [[0]]
    x=0
    while x<numor-1:
      ctor.append([0])
      x=x+1

    global dia
    leerexcel()
    vectorFO=asignacion(secuencia)

    listaciruj[dia] = cirujanosconsultas+cirujanosor
    listaasist[dia] = empty+asistentesor
    listapacs[dia] = pacientesconsultas+pacientesor

    return vectorFO

  def actualizarexcel(pacientesaborrar):
    #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
    j=0
    x=0
    # print('pacientes a borrar', pacientesaborrar)
    while j < len(pacientesaborrar):
      x=0
      # print('j',j)
      while x<len(pacientes):
        # print('x',x)
        if pacientesaborrar[j]==pacientes[x]:
          # print('se borra el paciente', pacientesaborrar[j])
          del pacientes[x]
          del cirjasoc[x]
          del dc[x]
          del dp[x]
          del dq[x]
          del estado[x]
          del hp[x]
          del hs[x]
          del pesos[x]
          del tiempoacum[x]
          del trc[x]
          del trp[x]
          del trq[x]
          del u[x]
        else:x=x+1
```

```
        j=j+1

    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                            'dp':dp,
                            'trp':trp,
                            'dq':dq,
                            'trq':trq,
                            'u':u,
                            'hp':hp,
                            'hs':hs,
                            'dc':dc,
                            'trc':trc,
                            'w':pesos,
                            'estado':estado,
                            'cirujano':cirjasoc,
                            'tacum':tiempoacum
                            })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                            'G':exp,
                            'S':uc
        })
    with pd.ExcelWriter('data.xlsx') as writer:
        nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
        datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
        datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):
    c=0
    C=[]
    while c<numc:
        C.append('C')
        c=c+1
    o=0
    O=[]
    while o<numor:
        O.append('O')
        o=o+1
    dfsol1 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[0],
                    'ASISTENTES':listaasist[0],
                    'PACIENTES':listapacs[0]

        })
    dfsol2 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[1],
                    'ASISTENTES':listaasist[1],
                    'PACIENTES':listapacs[1]

        })
    dfsol3 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[2],
                    'ASISTENTES':listaasist[2],
                    'PACIENTES':listapacs[2]

        })
    dfsol4 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[3],
                    'ASISTENTES':listaasist[3],
```

```
                'PACIENTES':listapacs[3]

        })
    dfsol5 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[4],
                'ASISTENTES':listaasist[4],
                'PACIENTES':listapacs[4]

        })
    dfsol6 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[5],
                'ASISTENTES':listaasist[5],
                'PACIENTES':listapacs[5]

        })
    dfsol7 = pd.DataFrame({'Espacio':C+O,
                'CIRUJANOS':listaciruj[6],
                'ASISTENTES':listaasist[6],
                'PACIENTES':listapacs[6]

        })

    with pd.ExcelWriter('solucion.xlsx') as writer:
        dfsol1.to_excel(writer, sheet_name = "LUNES")
        dfsol2.to_excel(writer, sheet_name = "MARTES")
        dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
        dfsol4.to_excel(writer, sheet_name = "JUEVES")
        dfsol5.to_excel(writer, sheet_name = "VIERNES")
        dfsol6.to_excel(writer, sheet_name = "SABADO")
        dfsol7.to_excel(writer, sheet_name = "DOMINGO")



#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")

#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
    aux.append([])
    x=x+1
x=0
listaciruj=[]
listaasist=[]
listapacs=[]
```

```
        while x<7:
            listaciruj.append(aux)
            listaasist.append(aux)
            listapacs.append(aux)
            x=x+1

        empty =[[]]
        x=0
        while x<numc-1:
            empty.append([])
            x=x+1

        dia=0
        while dia<1:

            #NEH#
            #1- ORDENAR LA SECUENCIA INICIAL
            sumatiempos=np.array([0.0]*len(pacientes))
            for x in range(len(pacientes)):
                # print(x, dc[x], dp[x], dq[x])
                sumatiempos[x]=dc[x]+dq[x]+dp[x]
            pacientesnumpy = np.array(pacientes)
            ordennumpy = np.array(sumatiempos)
            np.argsort(sumatiempos)
            secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
            secinicial1=np.flip(secinicial0)
            secinicial=secinicial1.tolist()
            #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
            #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
            # print(secinicial)
            pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()


            piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpiinicial=cajanegrasecuencia(piinicial)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
            print('FOpiinicial antes de NEH', FOpiinicial)
            j=1
            while j<len(secinicial):

                i=0
                secprobando=pi[:]
                # print('secuencia que estamos probando', secprobando)
                flagcambiado=0
                while i<=j:
                    secprobando.insert(i, secinicial[j])
                    # print('secuencia probando, insertamos en la posición', secprobando,i)
```

```
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOprobando=cajanegrasecuencia(secprobando)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                if mejora(FOprobando,FOpi):
                    pi=secprobando[:]
                    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                    FOpi=cajanegrasecuencia(pi)
                    hcc=hcccopia.copy()#para que las pruebas no alteren el real
                    hcor=hcorcopia.copy()

                    flagcambiado=1
                 #no se ha añadido ningún número a la secuencia y terminamos el bucle
                secprobando.pop(i)
                i=i+1
            if flagcambiado==0:
                secprobando.insert(i-1,secinicial[j])
                pi=secprobando[:]
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpi=cajanegrasecuencia(pi)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
            j=j+1
            #fin del NEH. Tenemos FOpi y pi. Tras el NEH hemos obtenido una asignación de cirujanos a
consultas
            #▼Ahora se fija esta asignación y se busca la mejor secuencia que se adapte a ella

        if mejora(FOpi,FOpiinicial):
            pibest=pi.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegrasecuencia(pibest)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
        else:
            pibest=piinicial.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegrasecuencia(piinicial)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

        print('FOpibest tras NEH', FOpibest)

        #Tras el NEH hemos obtenido una asignación de cirujanos a consultas
        #Ahora se fija esta asignación y se busca la mejor secuencia que se adapte a ella
        ccfija=cirujanosconsultas.copy()
        it=0
        Temp=1
        inicio=time.time()
        while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
            piprima=pi.copy()
            destruida=destruction(piprima)#destruction phase
            piprima=construction(destruida[0], ccfija, destruida[1]) #construction phase
```

```
    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FOpiprima=cajanegra(piprima, ccfija)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()

    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    s=nivelar(pacientesconsultas,ccfija[:numc])
    FOpiprima=cajanegra(piprima,s)
    print('FO tras construccion', FOpiprima)
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()


    if mejora(FOpiprima,FOpi):
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegra(pi, ccfija)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccfija[:numc])
        FOpi=cajanegra(pi,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOpi,FOpibest):
            pibest=pi.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegra(pibest, ccfija)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,ccfija[:numc])
            FOpibest=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
    elif random.uniform(0, 1)<=-math.exp(-(FOpiprima[1]-FOpi[1])/Temp):
        print('se acepta peor solucion')
        pi=piprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    Temp=0.95*Temp
    it=it+1

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpibest=cajanegra(pibest, ccfija)
```

```
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccfija[:numc])
FOpibest=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

print('FOpibest tras destruc+construc', FOpibest)
#aquí ya tendríamos IG para la secuencia
#aplicamos ahora el IG para la asignación de cirujanos

ccsec=ccfija.copy()#Se parte de esta asignación para realizar e NEH

ccinicial=ccsec[:]
for x in cirujanos:
   if [x] not in ccinicial and descanso[x][dia]==0:
      ccinicial.append([x])

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOcc=cajanegra(pibest,ccsec)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccsec[:numc])
FOcc=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()


cc=[ccinicial[0]]
j=1
#NEH#
while j<len(ccinicial):
   i=0
   ccprobando=cc[:]
   flagcambiado=0
   while i<=j:
      ccprobando.insert(i,ccinicial[j])
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOprobando=cajanegra(pibest,ccprobando[:numc])
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      s=nivelar(pacientesconsultas,ccprobando[:numc])
      FOprobando=cajanegra(pibest,s)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      if mejora(FOprobando,FOcc):
```

```
            cc=ccprobando[:]
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            flagcambiado=1
          ccprobando.pop(i)
          i=i+1
        if flagcambiado==0:
          ccprobando.insert(i-1,ccinicial[j])
          cc=ccprobando[:]

          hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
          hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
          FOcc=cajanegra(pibest,cc[:numc])
          hcc=hcccopia.copy()#para que las pruebas no alteren el real
          hcor=hcorcopia.copy()

          hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
          hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
          s=nivelar(pacientesconsultas,cc[:numc])
          FOcc=cajanegra(pibest,s)
          hcc=hcccopia.copy()#para que las pruebas no alteren el real
          hcor=hcorcopia.copy()

        j=j+1
#FIN NEH#
    FOccbest=FOcc[:]
    ccbest=cc.copy()

    print('Foccbest tras NEH', FOccbest)
    it=0
    Temp=1
    inicio=time.time()
    while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:

        #destruccion
        ccdestruida=destruccion(ccbest)
        ccprima=construccion(ccdestruida[0],ccdestruida[1])

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccprima=cajanegra(pibest,ccprima[:numc])
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccprima[:numc])
```

```
        FOccprima=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOccprima,FOcc):
            cc=ccprima.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOcc,FOccbest):
                ccbest=cc.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOccbest=cajanegra(pibest,ccbest[:numc])
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                s=nivelar(pacientesconsultas,ccbest[:numc])
                FOccbest=cajanegra(pibest,s)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()

        elif random.uniform(0, 1)<=-math.exp(-(FOccprima[1]-FOcc[1])/Temp):
            print('se acepta peor solucion')
            cc=ccprima.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
        Temp=0.95*Temp
        it=it+1

#aquí ya tendríamos FOccbest y ccbest, que serían las mejores soluciones tras el IG

    print('FOccbest tras destruc+construc', FOccbest)
    secuencia=pibest.copy()
    cirujanosconsultas=ccbest.copy()

    hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
    hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
    FO=cajanegra(secuencia, cirujanosconsultas[:numc]) #FO DEFINITIVA#
    hcc=hcccopia.copy()#para que las pruebas no alteren el real
    hcor=hcorcopia.copy()
```

```
            s=nivelar(pacientesconsultas,cirujanosconsultas[:numc])
            FO=cajanegra(secuencia,s)
            print('MEJOR FO DEFINITIVA',FO)
            sumaFO=sumaFO+FO[3]


            # print(secuencia)
            actualizarexcel(pacientesaborrar)
            dia=dia+1

        escribirsol(listaciruj, listaasist, listapacs)

        return sumaFO

    def IG2LS():

        global descanso, dia, hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosconsultas, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        global sumaFO

        def insertar(lista,pos1,pos2):
            copia=lista.copy()
            a=copia.pop(pos1)
            copia.insert(pos2,a)
            return copia

        def mejora(v1probando,v2best):
            if v1probando[3]<v2best[3]:
                return True
            else:
                return False

        def nivelar(patients,surgeons):

            j=min(numc-1,len(surgeons)-1)
            while j>=0:
                if patients[j]==[]:
                    surgeons.pop(j)
                    patients.pop(j)
                    surgeons.insert(numc-1,[])
                    patients.insert(numc-1,[])
                j=j-1
            return surgeons

        def destruction(sequence):#la secuencia que se le pasará será secbest
            n=random.sample(sequence,4)#número de elementos que destruiremos de nuestra secuencia
            # print(n)
            a=0
            secborrar=sequence[:]
            while a<len(n):
                secborrar.remove(n[a])
                a=a+1
            return secborrar, n
        def construction(seqbest, ccfija, n): #para secuenncia
            seqprobando=seqbest[:]
            a=0
```

```
        while a<len(n):
          seqprobando=seqbest[:]
          seqprobando.insert(0,n[a])#proponemos que la mejor secuencia de la fase es la primera, para luego ir tomando las que la mejoren
          seqbest=seqprobando[:]
          global hcc, hcor
          hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
          hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
          FObest=cajanegrasecuencia(seqbest)
          hcc=hcccopia.copy()#para que las pruebas no alteren el real
          hcor=hcorcopia.copy()

          hcorcopia=hcor.copy()#horas cirujanos en OR a la semana
          hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
          s=nivelar(pacientesconsultas,ccfija[:numc])
          FObest=cajanegra(seqbest,s)
          hcc=hcccopia.copy()#para que las pruebas no alteren el real
          hcor=hcorcopia.copy()

          # print('la primera secuencia es', secbest, FObest[1])
          seqprobando.pop(0)
          ###PROBANDO PARA NO INSERTAR DOS VECES EL MISMO NOMBRE###
          b=1
          while b<len(seqbest):
            # print('i',i)
            seqprobando.insert(b, n[a])
            # print('secuencia probando, insertamos en la posición', secprobando,i)
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegrasecuencia(seqprobando)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,ccfija[:numc])
            FOprobando=cajanegra(seqprobando,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            # print('FO=',FOprobando[1])
            if mejora(FOprobando,FObest):
              # print('mejora')
              seqbest=seqprobando[:]
              FObest=FOprobando[:]
             #no se ha añadido ningún número a la secuencia y terminamos el bucle
            seqprobando.pop(b)
            b=b+1
          a=a+1
        return seqbest

      def destruccion(cirujanosconsults): #para asignacion cirujanos a consultas
        n=random.sample(cirujanosconsults,4)
        a=0
        ccborrar=cirujanosconsults[:]
        while a<len(n):
          ccborrar.remove(n[a])
          a=a+1
```

```
    return ccborrar, n

def construccion(ccmejor,n): #para asignacion cirujanos a consultas
    global hcc, hcor
    ccprobando=ccmejor[:]
    a=0
    while a<len(n):
        ccprobando=ccmejor[:]
        ccprobando.append(n[a])
        ccmejor=ccprobando[:]

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FObest=cajanegra(pibest,ccmejor)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        ccprobando.pop(-1)
        b=0
        while b<len(ccmejor):
            ccprobando.insert(b,n[a])

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegra(pibest,ccprobando)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOprobando,FObest):
                # print('mejora')
                ccmejor=ccprobando[:]
                FObest=FOprobando[:]
            ccprobando.pop(b)
            b=b+1
        a=a+1
    return ccmejor


def leerexcel():
    global datospacientes, pacientes, cirjasoc, dp, trp, dq, trq, u, hp, hs, dc, trc, pesos, estado, cirjasoc, tiempoacum, datoscirujanos, cirujanos, exp, uc

    datospacientes = pd.read_excel("data.xlsx", "Pacientes")

    pacientes = datospacientes['Número paciente'].tolist()

    dp = datospacientes['dp'].tolist()

    trp = datospacientes['trp'].tolist()

    dq = datospacientes['dq'].tolist()

    trq = datospacientes['trq'].tolist()

    u = datospacientes['u'].tolist()

    hp = datospacientes['hp'].tolist()
```

```
            hs = datospacientes['hs'].tolist()

            dc = datospacientes['dc'].tolist()

            trc = datospacientes['trc'].tolist()

            pesos = datospacientes['w'].tolist()

            estado = datospacientes['estado'].tolist()

            cirjasoc1 = datospacientes['cirujano'].fillna(999)#cambiamos cuando no hay valores por 0
            cirjasoc = [int(x) for x in cirjasoc1.tolist()]
            for x in range(len(cirjasoc)):
               if cirjasoc[x]==999:
                  cirjasoc[x]='NaN'

            tiempoacum = datospacientes['tacum'].tolist()
         def cajanegra(secuencia,cc):

            global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor,
ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
            global listaciruj, listaasist, listapacs
            def asignacion(secuencia,cc):
               global pacientesconsultas, pacientesor, cirujanosor, asistentesor, ctconsultas, ctor, hcor, hcc
               pacientesconsultas = [[]]
               x=0
               while x<numc-1:
                  pacientesconsultas.append([])
                  x=x+1

               pacientesor = [[]]
               cirujanosor = [[]]
               asistentesor = [[]]
               x=0
               while x<numor-1:
                  pacientesor.append([])
                  cirujanosor.append([])
                  asistentesor.append([])
                  x=x+1

               global eq
               ###FUNCIONES###

               #Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en
quirófano
                  #sirve para medir el equilibrio de la carga de trabajo#
               def equilibrio(hcc,hcor):
                  return (hcc.var()+hcor.var())

               #Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta,
ponderado con los pesos
               def lateness(pacientes):
                  global pesos
                  x=0
                  late=[]
                  for x in range(len(pacientes)):
                     if estado[x]==0:
                        if tiempoacum[x]>trp[x]:
```

```
                    # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                    late.append((tiempoacum[x]-trp[x])*pesos[x])
                if estado[x]==1:
                    if tiempoacum[x]>trq[x]:
                        # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                        late.append((tiempoacum[x]-trq[x])*pesos[x])
                if estado[x]==2:
                    if tiempoacum[x]>trc[x]:
                        # print('el paciente {} va tarde a {}'.format(pacientes[x],estado[x]))
                        late.append((tiempoacum[x]-trc[x])*pesos[x])
            # print('Número de pacientes que van tarde:')
            return sum(late) #late indica cuántos pacientes van con retraso actualmente

        #Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
        def asignadodia(cirujano,consultaor):
            # global cirujanosconsultas
            global cirujanosor
            global numc
            i=0
            flag = 0
            if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
                for i in range(len(cc[:numc])):
                    if cirujano in cc[i]:#si el cirujano está en la consulta i, nos devuelve ese valor
                        return i
                        flag=1
                if flag==0:
                    return 'NaN'
                    # print("El cirujano no está asignado a ninguna consulta el dia ",dia)
            elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
                for i in range(numor):
                    if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR
j, nos devuelve ese valor
                        return i
                        flag=1
                if flag==0:
                    return 'NaN'
                    # print("El cirujano no está asignado a ningún quirófano el dia ",dia)

        #funcion para calcular el indice de un vector
        def indice(vector,valor):
            x=0
            for x in range(len(vector)):
                if vector[x]==valor:
                    return x
                    break

        #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea
necesario coincidir en unidades)
        def busquedaPAE (paciente):
            global cirujanos
            global numc
            global hcc
            j=0
            cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
            while j<len(cirujanos):
                if uc[j]==u[paciente]:#si el cirujano que estamos viendo es de la unidad
                    cirujanosdelaunidad.append(j)
                j=j+1
```

```
                #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosdelaunidad)):
                  horas.append(hcc[cirujanosdelaunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
                np.argsort(horasnumpy)
                cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
                j=0
                flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                #para cada cirujano, miramos si está asignado a alguna consulta el dia
                while j<len(cirujanosdelaunidad):

                  asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                  if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                      return [asignadoenconsulta,cirujanosdelaunidad[j]]
                      flagyatieneconsulta=1
                      break

                  if flagyatieneconsulta==1:break
                  j=j+1
                if flagyatieneconsulta==0:
                  return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

              # Función para asignar un paciente y un cirujano a consulta postop
              def busquedaPOSTOP (paciente):
                #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                flagyatieneconsulta=0

                asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                  if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                    return [asignadoenconsulta,cirjasoc[paciente]]
                    flagyatieneconsulta=1
                if flagyatieneconsulta==0:
                  return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana


              #Función para asignar un paciente y un equipo de cirujanos a quirófanos
              def busquedaOR(paciente):
                  #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                  #la experiencia y nivel de dificultad de la  operación.
                global hcor
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
```

```
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
                    j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    horas.append(hcor[cirujanosexpunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                    j=j+1
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    horas.append(hcor[asistexp[j]])
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                        asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                                return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                                flagyatieneOR=1
                                break
                        j=j+1
                    if flagyatieneOR==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                        j=0
                        while j<numor:
                            if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                                x=0
```

```
                    for x in cirujanosexpunidad:

                        if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                            return [j,cirujanosexpunidad[0],'NaN']
                            flagyatieneOR=1
                            break#cuando se le asigna, se sale del for
                        if flagyatieneOR==1:
                            break
                        j=j+1
                    if flagyatieneOR==0:
                        return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                else:   #sí se necesita un asistente
                    #vamos a buscar a la vez un cirujano principal y un asistente
                    i=0
                    flagyatieneasist=0
                    for i in cirujanosexpunidad:
                        asignadoenOR = asignadodia(i,'OR')
                        if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                            if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                                # print('sí habría hueco en su OR')
                                #buscamos un asistente
                                k=0
                                for k in asistexp:
                                    if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                        #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                        #si esta condicion se cumple, se asocia este asist
                                        return [asignadoenOR,i,k]
                                        flagyatieneasist=1
                                        break
                            if flagyatieneasist==1:break
                    if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                        #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                        i=0
                        for i in cirujanosexpunidad:
                            asignadoenconsulta = asignadodia(i,'consulta')
                            x=0
                            while x<numor:
                                if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                    if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                    #buscamos un asistente
                                        for k in asistexp:
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
                                                return [x,i,k]
                                                flagyatieneasist=1
                                                break
                                    if flagyatieneasist==1:break
                                    x=x+1
```

```
                    if flagyatieneasist==1:break
                if flagyatieneasist==0:
                    return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                    #significa que no hay hueco posible para este paciente en OR esta semana

            ###COMIENZO CODIGO###

            global pacientesaborrar
            atendidos=0
            pacientesaborrar=[]
            for k in pacientes:
                if k not in secuencia:
                    tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1 #los pacientes que no estén en la sec, igualmente no se van a procesar y se les añade un día a su tiempo acum
            for k in range(len(secuencia)): #x es el indice del paciente
                if estado[indice(pacientes,secuencia[k])]==0:
                    retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                    if retorno!='NaN':
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                        hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                        tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el +1 porque el día 0 aumenta 1 día
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

                tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==1:
                    retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
                    if retorno!='NaN':
                        # print('se asigna el paciente {} al OR {} el día con el cirujano {} y con asistente {}'.format(secuencia[x],retorno[0],retorno[1],retorno[2]))
                        cirujanosor[retorno[0]].append(retorno[1])
                        hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                        pacientesor[retorno[0]].append(secuencia[k])
                        ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                        cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        if retorno[2]!='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                            hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                        if retorno[2]=='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

                tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1 #se actualiza el tiempo acumulado.
                if estado[indice(pacientes,secuencia[k])]==2:
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal]
```

```
                    if retorno!='NaN':
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                        #habría que eliminar al paciente de la lista
                        hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                        flagcambiado=1
                        atendidos=atendidos+1
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                        pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)
            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
                i=i+1

            i=0
            while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
                i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return  eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion


        #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
        # #creo las listas#
        leerexcel()
         #auxiliares#
        ctconsultas = [[0]]
        x=0
        while x<numc-1:
            ctconsultas.append([0])
            x=x+1

        ctor = [[0]]
        x=0
```

```
        while x<numor-1:
            ctor.append([0])
            x=x+1

        global dia
        leerexcel()
        vectorFO=asignacion(secuencia,cc)

        listaciruj[dia] = cc[:numc] + cirujanosor
        listaasist[dia] = empty+asistentesor
        listapacs[dia] = pacientesconsultas+pacientesor

        return vectorFO

    def cajanegrasecuencia(secuencia):
        global hcor, hcc, numc, numor, ctor, ctcc, cirujanos, asistentesor, cirjasoc, cirujanosor, ctconsultas, ctor, pacientes, pacientesaborrar, pacientesconsultas, pacientesor, tiempoacum
        global listaciruj, listaasist, listapacs
        def asignacion(secuencia):
            global pacientesconsultas, cirujanosconsultas, pacientesor, cirujanosor, asistentesor, ctconsultas, ctor, hcor, hcc
            pacientesconsultas = [[]]
            cirujanosconsultas=[[]]
            x=0
            while x<numc-1:
                pacientesconsultas.append([])
                cirujanosconsultas.append([])
                x=x+1

            pacientesor = [[]]
            cirujanosor = [[]]
            asistentesor = [[]]
            x=0
            while x<numor-1:
                pacientesor.append([])
                cirujanosor.append([])
                asistentesor.append([])
                x=x+1


            ###FUNCIONES###

            #Función que aporta la varianza de lo que los cirujanos llevan acumulados en consulta y en quirófano
                #sirve para medir el equilibrio de la carga de trabajo#
            def equilibrio(hcc,hcor):
                return (hcc.var()+hcor.var())

            #Función para ver cuántos pacientes sobrepasan actualmente su tiempo de respuesta
            def lateness(pacientes):
                global pesos
                x=0
                late=[]
                for x in range(len(pacientes)):
                    if estado[x]==0:
                        if tiempoacum[x]>trp[x]:
                            late.append((tiempoacum[x]-trp[x])*pesos[x])
                    if estado[x]==1:
```

```
                    if tiempoacum[x]>trq[x]:
                        late.append((tiempoacum[x]-trq[x])*pesos[x])
                if estado[x]==2:
                    if tiempoacum[x]>trc[x]:
                        late.append((tiempoacum[x]-trc[x])*pesos[x])
            return sum(late) #late indica cuántos pacientes van con retraso actualmente

        #Función para comprobar si el cirujano está  asignado a algun OR o CONSULTA ese día
        def asignadodia(cirujano,consultaor):
            global cirujanosconsultas
            global cirujanosor
            global numc
            i=0
            flag = 0
            if consultaor=='consulta':#comprobamos si el cirujano está asignado a alguna CONSULTA
                for i in range(len(cirujanosconsultas)):
                    if cirujano in cirujanosconsultas[i]:#si el cirujano está en la consulta i, nos devuelve ese
valor
                        return i
                        flag=1
                if flag==0:
                    return 'NaN'
            elif consultaor == 'OR': #comprobamos si el cirujano está asignado a alguún QUIRÓFANO
                for i in range(numor):
                    if (cirujano in cirujanosor[i]) or (cirujano in asistentesor[i]):#si el cirujano está en el OR
j, nos devuelve ese valor
                        return i
                        flag=1
                if flag==0:
                    return 'NaN'

        #funcion para calcular el indice de un vector
        def indice(vector,valor):
            x=0
            for x in range(len(vector)):
                if vector[x]==valor:
                    return x
                    break

        #Función para asignar un paciente y un cirujano a PAE (teniendo en cuenta que en la PAE sea
necesario coincidir en unidades)
        def busquedaPAE (paciente):
            global cirujanos
            global numc
            global hcc
            j=0
            cirujanosdelaunidad = []#lista de cirujanos que pertenecen a la unidad del paciente
            while j<len(cirujanos):
                if uc[j]==u[paciente] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la
unidad
                    cirujanosdelaunidad.append(j)
                j=j+1
            #ordenamos los cirujanosdelaunidad según las horas de consulta que lleven
            j=0
            horas=[]
            for j in range(len(cirujanosdelaunidad)):
                horas.append(hcc[cirujanosdelaunidad[j]])
            horasnumpy=np.array(horas)
```

```
                cirujanosdelaunidadnumpy = np.array(cirujanosdelaunidad)
                np.argsort(horasnumpy)
                cirujanosdelaunidad = cirujanosdelaunidadnumpy[horasnumpy.argsort()].tolist()
                j=0
                flagyatieneconsulta=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                #para cada cirujano, miramos si está asignado a alguna consulta el dia
                while j<len(cirujanosdelaunidad):

                  asignadoenconsulta = asignadodia(cirujanosdelaunidad[j],'consulta')
                  if asignadoenconsulta!='NaN':#el cirujano sí está asignado a la consulta asignadoenconsulta
                    if ctconsultas[asignadoenconsulta][-1]+dp[paciente]<=6:#ponemos 6h de consulta al dia
                      return [asignadoenconsulta,cirujanosdelaunidad[j]]
                      flagyatieneconsulta=1
                      break

                  if flagyatieneconsulta==1:break
                  j=j+1
                if flagyatieneconsulta==0:#si no tiene consulta en toda la semana ningún cirujano, se le asigna al primero la primera consulta libre
                  j=0

                  while j<len(cirujanosconsultas):
                    if cirujanosconsultas[j]==[]:#si la consulta está libre
                      x=0
                      for x in cirujanosdelaunidad:
                        if asignadodia(x, 'OR')=='NaN' :#and (asignadodia(x, 'consulta', dia)=='NaN' or asignadodia(x, 'consulta', dia)==j):
                          #si el cirujano no tiene OR asignado ese día, y si tiene consultas que sean esa misma,
                          return [j,x]
                          flagyatieneconsulta=1
                          break#cuando se le asigna, se sale del for
                      if flagyatieneconsulta==1:
                        break
                    j=j+1
                if flagyatieneconsulta==0:
                  return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

              # Función para asignar un paciente y un cirujano a consulta postop
              def busquedaPOSTOP (paciente):
                #se va pasando por cada día por cada consulta, viendo dónde tiene el cirujasoc consulta asignada y si hay hueco

                flagyatieneconsulta=0

                asignadoenconsulta=asignadodia(cirjasoc[paciente],'consulta')
                asignadoenOR = asignadodia(cirjasoc[paciente],'OR')
                if asignadoenconsulta!='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#el cirujano sí está asignado a la consulta asignadoenconsulta
                  if ctconsultas[asignadoenconsulta][-1]+dc[paciente]<=6:#ponemos 6h de consulta al dia
                    return [asignadoenconsulta,cirjasoc[paciente]]
                    flagyatieneconsulta=1
                if flagyatieneconsulta==0:#e ciruj asoc no está en ninguna consulta ningún día
                  #se le asigna el primer hueco que haya en la semana
```

```
                j=0

                while j<len(cirujanosconsultas):
                    if  cirujanosconsultas[j]==[]:#este es la consulta donde se asignará el cirujano
                        if asignadoenconsulta=='NaN' and asignadoenOR=='NaN' and descanso[cirjasoc[paciente]][dia]==0:#si el cirujano no tiene OR asignado ese día, puede asignarse a consultas
                            return [j,cirjasoc[paciente]]
                            flagyatieneconsulta=1
                            break#cuando se le asigna, se sale del for
                    if flagyatieneconsulta==1:
                        break
                    j=j+1
                if flagyatieneconsulta==0:
                    return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana

            #Función para asignar un paciente y un equipo de cirujanos a quirófanos
            def busquedaOR(paciente):
                #se hará igual que para la consulta PAE, solo que ahora deberá haber cirujano principal y asistente. SSe tendrán en cuenta
                    #la experiencia y nivel de dificultad de la  operación.
                j=0
                cirujanosexpunidad = []#lista de cirujanos que pertenecen a la unidad del paciente y que cuentan con experiencia suficiente
                while j<len(cirujanos):
                    if uc[j]==u[paciente] and hp[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo es de la unidad y tiene experiencia suficiente
                        cirujanosexpunidad.append(cirujanos[j])
                        # print(cirujanosexpunidad)
                    j=j+1
                #ordenamos los cirujanosdelaunidad según las horas de OR que lleven
                j=0
                horas=[]
                for j in range(len(cirujanosexpunidad)):
                    horas.append(hcor[cirujanosexpunidad[j]])
                horasnumpy=np.array(horas)
                cirujanosexpunidadnumpy = np.array(cirujanosexpunidad)
                np.argsort(horasnumpy)
                cirujanosexpunidad = cirujanosexpunidadnumpy[horasnumpy.argsort()].tolist()
                #esta lista es para asignar un CIRUJANO PRINCIPAL, tras asignar un cirujano principal, se asignará el asistente

                ###LISTA ASISTENTES###
                j=0
                asistexp = []#lista de cirujanos que cuentan con experiencia suficiente, para los asists nose tiene en cuenta la unidad
                while j<len(cirujanos):
                    if hs[paciente]<=exp[j] and descanso[j][dia]==0:#si el cirujano que estamos viendo  tiene experiencia suficiente
                        asistexp.append(cirujanos[j])
                    j=j+1
                #ordenamos los asistentes según las horas de consulta que lleven
                j=0
                horas=[]
                for j in range(len(asistexp)):
                    horas.append(hcor[asistexp[j]])
                horasnumpy=np.array(horas)
                asistexpnumpy = np.array(asistexp)
```

```
                np.argsort(horasnumpy)
                asistexp = asistexpnumpy[horasnumpy.argsort()].tolist()


                if hs[paciente]==0:#si no necesitamos asistente, se haría con el mismo algoritmo de buscar consulta:
                    j=0

                    flagyatieneOR=0#flag para ver si pasa por los bucles y no tiene consulta en toda la semana
                    #para cada cirujano, miramos si está asignado a algún OR el dia
                    while j<len(cirujanosexpunidad):
                       asignadoenOR = asignadodia(cirujanosexpunidad[j],'OR')
                       if asignadoenOR!='NaN':#el cirujano sí está asignado a la OR asignadoenOR
                          if ctor[asignadoenOR][-1]+dq[paciente]<=8:#ponemos 8h de OR al dia
                             return [asignadoenOR,cirujanosexpunidad[j],'NaN']
                             flagyatieneOR=1
                             break
                       j=j+1
                    if flagyatieneOR==0:#si no tiene OR en toda la semana ningún cirujano, se le asigna al primero el primer hueco libre
                       j=0
                       while j<numor:
                          if ctor[j][-1]+dq[paciente]<=8:#este es el hueco donde se asignará el paciente
                             x=0
                             for x in cirujanosexpunidad:
                                if asignadodia(x,'consulta')=='NaN' and (asignadodia(x,'OR')=='NaN' or asignadodia(x,'OR')==j ):#si el cirujano no tiene consulta asignado ese día, puede asignarse a consultas
                                   return [j,cirujanosexpunidad[0],'NaN']
                                   flagyatieneOR=1
                                   break#cuando se le asigna, se sale del for
                             if flagyatieneOR==1:
                                break
                          j=j+1
                       if flagyatieneOR==0:
                          return 'NaN' #si el flag sigue, se ve que no hay hueco para la consulta PAE para este paciente en toda la semana
                else:  #sí se necesita un asistente
                   #vamos a buscar a la vez un cirujano principal y un asistente
                   i=0
                   flagyatieneasist=0
                   for i in cirujanosexpunidad:
                      asignadoenOR = asignadodia(i,'OR')
                      if asignadoenOR!='NaN':#el cirujano sí está asignado ese día en un quirófano
                         if ctor[asignadoenOR][-1]+dq[paciente]<=8:#le asignamos este hueco en la consulta
                            #buscamos un asistente
                            k=0
                            for k in asistexp:
                               if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==asignadoenOR ):
                                  #asiste!=cirujasoc  asist no puede estar en consultas ese dia    asist no puede estar asignado a otro OR que no sea este ese dia
                                  #si esta condicion se cumple, se asocia este asist
                                  # print('retornamos')
                                  return [asignadoenOR,i,k]
                                  flagyatieneasist=1
                                  break
                      if flagyatieneasist==1:break
```

```
                    if flagyatieneasist==0: #hemos probado todos los cirujanos pero no hay ninguno ya asignado a un OR durante la semana. Hay que buscarle nuevo hueco
                        #Vamos buscando un hueco para el primer cirujano, y vemos si tiene asistente posible. Si no hay asistente, se pasa al sig hueco
                        i=0
                        for i in cirujanosexpunidad:
                            asignadoenconsulta = asignadodia(i,'consulta')
                            x=0
                            while x<numor:
                                if asignadoenconsulta=='NaN' and (asignadodia(i,'OR')=='NaN' or asignadodia(i,'OR')==x ):#comprobamos que el ciruj no está en ninguna consulta ese día
                                    if ctor[x][-1]+dq[paciente]<=8:#le asignamos este hueco en OR
                                    #buscamos un asistente
                                        for k in asistexp:
                                            if k!=i and asignadodia(k,'consulta')=='NaN' and (asignadodia(k,'OR')=='NaN' or asignadodia(k,'OR')==x ):
                                                #asiste!=cirujasoc  asist no puede estar en consultas ese dia     asist no puede estar asignado a otro OR que no sea este ese dia
                                                #si esta condicion se cumple, se asocia este asist
                                                return [x,i,k]
                                                flagyatieneasist=1
                                                break
                                        if flagyatieneasist==1:break
                                    x=x+1
                                if flagyatieneasist==1:break
                        if flagyatieneasist==0:
                            return'NaN' #si tras buscar todos los cirujanos posibles con todos los asistentes en todos los huecos el flag sigue a 0,
                            #significa que no hay hueco posible para este paciente en OR esta semana

                ###COMIENZO CODIGO###

                global pacientesaborrar
                atendidos=0
                pacientesaborrar=[]
                for k in pacientes:
                    if k not in secuencia:
                        tiempoacum[indice(pacientes,k)]=tiempoacum[indice(pacientes,k)]+1
                for k in range(len(secuencia)): #x es el indice del paciente
                    if estado[indice(pacientes,secuencia[k])]==0:
                        retorno=busquedaPAE(indice(pacientes,secuencia[k])) #[consulta,cirujprincipal]
                        if retorno!='NaN':
                            cirujanosconsultas[retorno[0]]=[retorno[1]]
                            pacientesconsultas[retorno[0]].append(secuencia[k])
                            ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-1]+dp[indice(pacientes,secuencia[k])])
                            hcc[retorno[1]]=hcc[retorno[1]]+dp[indice(pacientes,secuencia[k])]
                            tiempoacum[indice(pacientes,secuencia[k])]=0#se pone el -1 porque el día 0 aumenta 1 día
                            flagcambiado=1
                            atendidos=atendidos+1
                        else:
                            flagcambiado=0

                tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        retorno=busquedaOR(indice(pacientes,secuencia[k])) #me devuelve [consulta,cirujprincipal,asistente]
```

```
                    if retorno!='NaN':
                        cirujanosor[retorno[0]].append(retorno[1])
                        hcor[retorno[1]]=hcc[retorno[1]]+dq[indice(pacientes,secuencia[k])]
                        pacientesor[retorno[0]].append(secuencia[k])
                        ctor[retorno[0]].append(ctor[retorno[0]][-1]+dq[indice(pacientes,secuencia[k])])
                        cirjasoc[indice(pacientes,secuencia[k])]=retorno[1]
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        if retorno[2]!='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                            hcor[retorno[2]]=hcor[retorno[2]]+dq[indice(pacientes,secuencia[k])]
                        if retorno[2]=='NaN':
                            asistentesor[retorno[0]].append(retorno[2])
                        flagcambiado=1
                        atendidos=atendidos+1
                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1
                if estado[indice(pacientes,secuencia[k])]==2:
                    retorno=busquedaPOSTOP(indice(pacientes,secuencia[k])) #me devuelve
[consulta,cirujprincipal]
                    # print('retorno',retorno)
                    if retorno!='NaN':
                        cirujanosconsultas[retorno[0]]=[retorno[1]]
                        pacientesconsultas[retorno[0]].append(secuencia[k])
                        ctconsultas[retorno[0]].append(ctconsultas[retorno[0]][-
1]+dc[indice(pacientes,secuencia[k])])
                        #habría que eliminar al paciente de la lista
                        hcc[retorno[1]]=hcc[retorno[1]]+dc[indice(pacientes,secuencia[k])]
                        flagcambiado=1
                        atendidos=atendidos+1
                        tiempoacum[indice(pacientes,secuencia[k])]=0 #se actualiza el tiempo acumulado.
                        # borramos todo lo del paciente en cuestión, ya que ya ha terminado su ciclo en el
hospital#
                        pacientesaborrar.append(pacientes[indice(pacientes,secuencia[k])])

                    else:
                        flagcambiado=0

tiempoacum[indice(pacientes,secuencia[k])]=tiempoacum[indice(pacientes,secuencia[k])]+1

                if flagcambiado==1 and (estado[indice(pacientes,secuencia[k])]==0 or
estado[indice(pacientes,secuencia[k])]==1):
                    estado[indice(pacientes,secuencia[k])]=estado[indice(pacientes,secuencia[k])]+1
                    if estado[indice(pacientes,secuencia[k])]==1:
                        mu=random.randint(1,4)
                        sigma=random.choice([0.1*mu,0.2*mu,0.3*mu,0.4*mu,0.5*mu])
                        dq[indice(pacientes,secuencia[k])]=abs(random.normalvariate(mu, sigma))

            eq=equilibrio(hcc,hcor)
            tarde=lateness(pacientes)

            utilizacionor=0
            utilizacionc=0
            i=0

            while i<numc:
                utilizacionc=ctconsultas[i][-1]+utilizacionc
```

```
                i=i+1

            i=0
            while i<numor:
                utilizacionor=ctor[i][-1]+utilizacionor
                i=i+1
            utilizacion=(utilizacionor+utilizacionc)/(8*numor+6*numc)
            return      eq/25,tarde/(332*15*len(pacientes)),utilizacion,eq/25+tarde/(332*15*len(pacientes))-
utilizacion

        #vamos a tener un libro excel, con una hoja cada día. Habrá una columna de 'cirujanos, asistente,
paciente', otra para consultas y OR
        # #creo las listas#
        leerexcel()
        ctconsultas = [[0]]
        x=0
        while x<numc-1:
            ctconsultas.append([0])
            x=x+1

        ctor = [[0]]
        x=0
        while x<numor-1:
            ctor.append([0])
            x=x+1

        global dia
        leerexcel()
        vectorFO=asignacion(secuencia)

        listaciruj[dia] = cirujanosconsultas+cirujanosor
        listaasist[dia] = empty+asistentesor
        listapacs[dia] = pacientesconsultas+pacientesor

        return vectorFO

    def actualizarexcel(pacientesaborrar):
        #borro los pacientesaborrar para que no los coja en la siguiente ronda y actualizo el excel
        j=0
        x=0
        while j < len(pacientesaborrar):
            x=0
            while x<len(pacientes):
                if pacientesaborrar[j]==pacientes[x]:
                    del pacientes[x]
                    del cirjasoc[x]
                    del dc[x]
                    del dp[x]
                    del dq[x]
                    del estado[x]
                    del hp[x]
                    del hs[x]
                    del pesos[x]
                    del tiempoacum[x]
                    del trc[x]
                    del trp[x]
                    del trq[x]
                    del u[x]
```

```
            else:x=x+1
        j=j+1

    #actualización del excel#
    nuevodataframepacientes = pd.DataFrame({'Número paciente':pacientes,
                                'dp':dp,
                                'trp':trp,
                                'dq':dq,
                                'trq':trq,
                                'u':u,
                                'hp':hp,
                                'hs':hs,
                                'dc':dc,
                                'trc':trc,
                                'w':pesos,
                                'estado':estado,
                                'cirujano':cirjasoc,
                                'tacum':tiempoacum
                                })
    datoscirujanos = pd.DataFrame({'Número Cirujano': cirujanos,
                                'G':exp,
                                'S':uc
        })
    with pd.ExcelWriter('data.xlsx') as writer:
        nuevodataframepacientes.to_excel(writer, sheet_name = "Pacientes")
        datoscirujanos.to_excel(writer, sheet_name = "Cirujanos")
        datos.to_excel(writer, sheet_name = "Datos")
def escribirsol(listaciruj,listaasist,listapacs):

    c=0
    C=[]
    while c<numc:
        C.append('C')
        c=c+1
    o=0
    O=[]
    while o<numor:
        O.append('O')
        o=o+1
    dfsol1 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[0],
                    'ASISTENTES':listaasist[0],
                    'PACIENTES':listapacs[0]

        })
    dfsol2 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[1],
                    'ASISTENTES':listaasist[1],
                    'PACIENTES':listapacs[1]

        })
    dfsol3 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[2],
                    'ASISTENTES':listaasist[2],
                    'PACIENTES':listapacs[2]

        })
    dfsol4 = pd.DataFrame({'Espacio':C+O,
```

```
                    'CIRUJANOS':listaciruj[3],
                    'ASISTENTES':listaasist[3],
                    'PACIENTES':listapacs[3]

        })
    dfsol5 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[4],
                    'ASISTENTES':listaasist[4],
                    'PACIENTES':listapacs[4]

        })
    dfsol6 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[5],
                    'ASISTENTES':listaasist[5],
                    'PACIENTES':listapacs[5]

        })
    dfsol7 = pd.DataFrame({'Espacio':C+O,
                    'CIRUJANOS':listaciruj[6],
                    'ASISTENTES':listaasist[6],
                    'PACIENTES':listapacs[6]

        })

    with pd.ExcelWriter('solucion.xlsx') as writer:
        dfsol1.to_excel(writer, sheet_name = "LUNES")
        dfsol2.to_excel(writer, sheet_name = "MARTES")
        dfsol3.to_excel(writer, sheet_name = "MIERCOLES")
        dfsol4.to_excel(writer, sheet_name = "JUEVES")
        dfsol5.to_excel(writer, sheet_name = "VIERNES")
        dfsol6.to_excel(writer, sheet_name = "SABADO")
        dfsol7.to_excel(writer, sheet_name = "DOMINGO")

#LECTURA DE EXCEL#

datos=pd.read_excel("data.xlsx", "Datos")
numor = datos.loc[0,'Número de quirófanos']
numc = datos.loc[0,'Número de consultas']
#cirujanos#
datoscirujanos = pd.read_excel("data.xlsx", "Cirujanos")
cirujanos = datoscirujanos['Número Cirujano'].tolist()
exp = datoscirujanos['G'].tolist()
uc = datoscirujanos['S'].tolist()

leerexcel()

#arrays que deben empezar siendo vacíos, y luego se irán rellenando#
hcor = np.array([0.0]*len(cirujanos)) #horas cirujanos en OR a la semana
hcc = np.array([0.0]*len(cirujanos)) #horas cada cirujano en consulta a la semana
aux=[[]]

x=0
while x<(numc+numor)-1:
    aux.append([])
    x=x+1
x=0
listaciruj=[]
listaasist=[]
```

```
            listapacs=[]
            while x<7:
               listaciruj.append(aux)
               listaasist.append(aux)
               listapacs.append(aux)
               x=x+1

            empty =[[]]
            x=0
            while x<numc-1:
               empty.append([])
               x=x+1

            dia=0
            while dia<1:

               #NEH#
               #1- ORDENAR LA SECUENCIA INICIAL
               sumatiempos=np.array([0.0]*len(pacientes))
               for x in range(len(pacientes)):
                  sumatiempos[x]=dc[x]+dq[x]+dp[x]
               pacientesnumpy = np.array(pacientes)
               ordennumpy = np.array(sumatiempos)
               np.argsort(sumatiempos)
               secinicial0=pacientesnumpy[ordennumpy.argsort()].tolist()#ordena de menor a mayor
               secinicial1=np.flip(secinicial0)
               secinicial=secinicial1.tolist()
               #en este punto ya tenemos la secuencia inicial, ordenada de mayor a menor por la suma de sus tiempos de proceso
               #para las secuencias que vamos viendo, se probará con unos cirujanos que asigne el método a cada consulta.
               pi=[secinicial[0]]#el primer elemento de la secuencia es el primero de la inicial
               hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
               hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
               FOpi=cajanegrasecuencia(pi)
               hcc=hcccopia.copy()#para que las pruebas no alteren el real
               hcor=hcorcopia.copy()

               piinicial=secinicial.copy()#el primer elemento de la secuencia es el primero de la inicial
               hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
               hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
               FOpiinicial=cajanegrasecuencia(piinicial)
               hcc=hcccopia.copy()#para que las pruebas no alteren el real
               hcor=hcorcopia.copy()
               print('FOpiinicial antes de NEH', FOpiinicial)

               j=1
               while j<len(secinicial):

                  i=0
                  secprobando=pi[:]
                  # print('secuencia que estamos probando', secprobando)
                  flagcambiado=0
                  while i<=j:
                     secprobando.insert(i, secinicial[j])
                     # print('secuencia probando, insertamos en la posición', secprobando,i)

                     hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
```

```
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOprobando=cajanegrasecuencia(secprobando)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOprobando,FOpi):

                pi=secprobando.copy()
                hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
                hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
                FOpi=cajanegrasecuencia(pi)
                hcc=hcccopia.copy()#para que las pruebas no alteren el real
                hcor=hcorcopia.copy()
                flagcambiado=1
             #no se ha añadido ningún número a la secuencia y terminamos el bucle
            secprobando.pop(i)
            i=i+1
        if flagcambiado==0:
            secprobando.insert(i-1,secinicial[j])
            pi=secprobando[:]
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpi=cajanegrasecuencia(pi)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
        j=j+1
        #fin del NEH. Tenemos FOpi y pi

    if mejora(FOpi,FOpiinicial):#si el NEH ha mejorado la secuencia que teniamos inicialmente
        pibest=pi.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpibest=cajanegrasecuencia(pibest)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    else:
        pibest=piinicial.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpibest=cajanegrasecuencia(piinicial)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

    print('FOpibest con la secuencia tras NEH y antes de LS', FOpibest)
    #Tras el NEH hemos obtenido una asignación de cirujanos a consultas
    #Ahora se fija esta asignación y se busca la mejor secuencia que se adapte a ella
    ccfija=cirujanosconsultas.copy()

    it=0
    Temp=1
    inicio=time.time()
    while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
        piprima=pi.copy()
        destruida=destruction(piprima)#destruction phase
        piprima=construction(destruida[0], ccfija, destruida[1]) #construction phase

        #LOCAL SEARCH#
        n=len(piprima)
```

```
i=0
j=0
piprimaprima=piprima.copy()
hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpiprimaprima=cajanegra(piprimaprima, ccfija)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

hcorcopia=hcor.copy()#horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccfija[:numc])
FOpiprimaprima=cajanegra(piprimaprima,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

while j<n:
   i=0
   while i<n:
      secprima=piprima.copy()
      secprima.remove(piprima[j])
      secprima.insert(i,piprima[j])
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOsecprima=cajanegra(secprima, ccfija)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      hcorcopia=hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      s=nivelar(pacientesconsultas,ccfija[:numc])
      FOsecprima=cajanegra(secprima,s)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      if mejora(FOsecprima,FOpiprimaprima):
         piprimaprima=secprima.copy()
         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         FOpiprimaprima=cajanegra(piprimaprima, ccfija)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         s=nivelar(pacientesconsultas,ccfija[:numc])
         FOpiprimaprima=cajanegra(piprimaprima,s)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

      i=i+1
   j=j+1
#FIN LOCAL SEARCH

if mejora(FOpiprimaprima,FOpi):
   pi=piprimaprima.copy()
   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
```

```
        FOpi=cajanegra(pi, ccfija)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccfija[:numc])
        FOpi=cajanegra(pi,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOpi,FOpibest):
            pibest=pi.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOpibest=cajanegra(pibest, ccfija)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia= hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,ccfija[:numc])
            FOpibest=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

    elif random.uniform(0, 1)<=-math.exp(-(FOpiprimaprima[1]-FOpi[1])/Temp):
        print('se acepta peor solucion')
        pi=piprimaprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOpi=cajanegrasecuencia(pi)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()
    Temp=0.95*Temp
    it=it+1

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOpibest=cajanegra(pibest, ccfija)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()
print('FOpibest tras LS', FOpibest)

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccfija[:numc])
FOpibest=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

#aquí ya tendríamos IG para la secuencia
#aplicamos ahora el IG para la asignación de cirujanos

ccsec=ccfija.copy() #Se parte de esta asignación para realizar e NEH

ccinicial=ccsec[:]
for x in cirujanos:
```

```
    if [x] not in ccinicial and descanso[x][dia]==0:
        ccinicial.append([x])

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
FOcc=cajanegra(pibest,ccsec)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
s=nivelar(pacientesconsultas,ccsec[:numc])
FOcc=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()


cc=[ccinicial[0]]
j=1
#NEH#
while j<len(ccinicial):
    i=0
    ccprobando=cc[:]
    flagcambiado=0
    while i<=j:
        ccprobando.insert(i,ccinicial[j])
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOprobando=cajanegra(pibest,ccprobando[:numc])
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccprobando[:numc])
        FOprobando=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        if mejora(FOprobando,FOcc):
            cc=ccprobando[:]
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            flagcambiado=1
        ccprobando.pop(i)
        i=i+1
    if flagcambiado==0:
```

```
            ccprobando.insert(i-1,ccinicial[j])
            cc=ccprobando[:]

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc[:numc])
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            s=nivelar(pacientesconsultas,cc[:numc])
            FOcc=cajanegra(pibest,s)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

        j=j+1
#FIN NEH#

    FOccbest=FOcc[:]
    ccbest=cc.copy()

    print('Foccbest con el que se entra a LS', FOccbest)
    it=0
    Temp=1
    inicio=time.time()
    while time.time()-inicio<len(pacientes)*(numor+numc)*0.025:
        #destruccion
        ccdestruida=destruccion(ccbest)
        ccprima=construccion(ccdestruida[0],ccdestruida[1])
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccprima=cajanegra(pibest,ccprima[:numc])
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        s=nivelar(pacientesconsultas,ccprima[:numc])
        FOccprima=cajanegra(pibest,s)
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()


        #LOCAL SEARCH#

        n=len(ccprima)
        i=0
        j=0
        ccprimaprima=ccprima.copy()
        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
        FOccprimaprima=cajanegra(pibest,ccprimaprima[:numc]) #es FOccprima
        hcc=hcccopia.copy()#para que las pruebas no alteren el real
        hcor=hcorcopia.copy()

        hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
        hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
```

```
s=nivelar(pacientesconsultas,ccprimaprima[:numc])
FOccprimaprima=cajanegra(pibest,s)
hcc=hcccopia.copy()#para que las pruebas no alteren el real
hcor=hcorcopia.copy()

while j<n:
   i=0
   while i<n:
      cirujanosprima=ccprima.copy()
      cirujanosprima.remove(ccprima[j])
      cirujanosprima.insert(i,ccprima[j])
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FOcirujanosprima=cajanegra(pibest,cirujanosprima[:numc])
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      s=nivelar(pacientesconsultas,cirujanosprima[:numc])
      FOcirujanosprima=cajanegra(pibest,s)
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      if mejora(FOcirujanosprima,FOccprimaprima):
         ccprimaprima=ccprima.copy()
         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         FOccprimaprima=cajanegra(pibest,ccprimaprima[:numc])
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()

         hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
         hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
         s=nivelar(pacientesconsultas,ccprimaprima[:numc])
         FOccprimaprima=cajanegra(pibest,s)
         hcc=hcccopia.copy()#para que las pruebas no alteren el real
         hcor=hcorcopia.copy()
      i=i+1
      if i>numc:#cuandobya el cirujano es asignado a consultas inexistentes, break
         break
   j=j+1

#FIN LOCAL SEARCH

if mejora(FOccprimaprima,FOcc):
   cc=ccprimaprima.copy()

   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
   FOcc=cajanegra(pibest,cc[:numc])
   hcc=hcccopia.copy()#para que las pruebas no alteren el real
   hcor=hcorcopia.copy()

   s=nivelar(pacientesconsultas,cc[:numc])
   hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
   hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
   FOcc=cajanegra(pibest,s)
```

```
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()

            if mejora(FOcc,FOccbest):
               ccbest=cc.copy()
               hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
               hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
               FOccbest=cajanegra(pibest,ccbest[:numc])
               hcc=hcccopia.copy()#para que las pruebas no alteren el real
               hcor=hcorcopia.copy()

               s=nivelar(pacientesconsultas,ccbest[:numc])

               hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
               hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
               FOccbest=cajanegra(pibest,s)
               hcc=hcccopia.copy()#para que las pruebas no alteren el real
               hcor=hcorcopia.copy()

         elif random.uniform(0, 1)<=-math.exp(-(FOccprima[1]-FOcc[1])/Temp):
            print('se acepta peor solucion')
            cc=ccprima.copy()
            hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
            hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
            FOcc=cajanegra(pibest,cc)
            hcc=hcccopia.copy()#para que las pruebas no alteren el real
            hcor=hcorcopia.copy()
         Temp=0.95*Temp
         it=it+1

   #aquí ya tendríamos FOccbest y ccbest, que serían las mejores soluciones tras el IG

      print('FO con los cirujs tras LS', FOccbest)
      secuencia=pibest.copy()
      cirujanosconsultas=ccbest.copy()
      hcorcopia = hcor.copy() #horas cirujanos en OR a la semana
      hcccopia = hcc.copy() #horas cada cirujano en consulta a la semana
      FO=cajanegra(secuencia, cirujanosconsultas[:numc]) #FO DEFINITIVA#
      hcc=hcccopia.copy()#para que las pruebas no alteren el real
      hcor=hcorcopia.copy()

      s=nivelar(pacientesconsultas,cirujanosconsultas[:numc])
      FO=cajanegra(secuencia,s)
      print('MEJOR FO DEFINITIVA',FO)
      sumaFO=sumaFO+FO[3]
      actualizarexcel(pacientesaborrar)
      dia=dia+1

   escribirsol(listaciruj, listaasist, listapacs)
   return sumaFO

sumaFO=0
j=0
i=[[]]
inicial=time.time()
for numor in [9]: #3.6.9
   for numc in [5]:
      for alfa in [2]:
```

```
for beta in [2]:
    instancia=0
    while instancia<1:
        generarinstancia(numor, numc, alfa, beta)
        original=r"D:\Documents\ETSI\3funciones\instancia.xlsx"
        nuevo=r"D:\Documents\ETSI\3funciones\data.xlsx"
        shutil.copyfile(original, nuevo)
        print('empieza IG0')
        inicio=time.time()
        sumaFO=0
        ig0=IG0()
        tig0=time.time()-inicio
        sumaFO=0

        shutil.copyfile(original, nuevo)
        print('empieza IG0LS')
        inicio=time.time()
        ig0ls=IG0LS()
        tig0ls=time.time()-inicio
        sumaFO=0
        shutil.copyfile(original, nuevo)
        print('empieza IG1')
        inicio=time.time()
        ig1=IG1()
        tig1=time.time()-inicio
        sumaFO=0
        shutil.copyfile(original, nuevo)
        print('empieza IG1LS')
        inicio=time.time()
        ig1ls=IG1LS()
        tig1ls=time.time()-inicio
        shutil.copyfile(original, nuevo)
        print('empieza IG2')
        sumaFO=0
        inicio=time.time()
        ig2=IG2()
        tig2=time.time()-inicio
        shutil.copyfile(original, nuevo)
        print('empieza IG2LS')
        sumaFO=0
        inicio=time.time()
        ig2ls=IG2LS()
        tig2ls=time.time()-inicio
        if j==0:
            nc=[numc]
            nor=[numor]
            alfas=[alfa]
            betas=[beta]
            surgeons=[S]
            patients=[P-1]
            FOig0=[ig0]
            tiempoig0=[tig0]
            FOig0ls=[ig0ls]
            tiempoig0ls=[tig0ls]
            FOig1=[ig1]
            tiempoig1=[tig1]
            FOig1ls=[ig1ls]
            tiempoig1ls=[tig1ls]
```

```
                FOig2=[ig2]
                tiempoig2=[tig2]
                FOig2ls=[ig2ls]
                tiempoig2ls=[tig2ls]
            else:
                nc.append(numc)
                nor.append(numor)
                alfas.append(alfa)
                betas.append(beta)
                surgeons.append(S)
                patients.append(P-1)
                FOig0.append(ig0)
                FOig0ls.append(ig0ls)
                FOig1.append(ig1)
                FOig1ls.append(ig1ls)
                FOig2.append(ig2)
                FOig2ls.append(ig2ls)
                tiempoig0.append(tig0)
                tiempoig0ls.append(tig0ls)
                tiempoig1.append(tig1)
                tiempoig1ls.append(tig1ls)
                tiempoig2.append(tig2)
                tiempoig2ls.append(tig2ls)
            j=j+1
            instancia=instancia+1
            print('instancia número',j)


df = pd.DataFrame({'numc':nc,
                   'numor':nor,
                   'alfa':alfas,
                   'beta':betas,
                   'Cirujanos':surgeons,
                   'Pacientes':patients,
                    'FO IG0':FOig0,
                    'Tiempo IG0':tiempoig0,
                    'FO IG0LS':FOig0ls,
                    'Tiempo IG0LS':tiempoig0ls,
                     'FO IG1':FOig1,
                     'Tiempo IG1':tiempoig1,
                    'FO IG1LS':FOig1ls,
                    'Tiempo IG1LS':tiempoig1ls,
                    'FO IG2':FOig2,
                    'Tiempo IG2':tiempoig2,
                    'FO IG2LS':FOig2ls,
                    'Tiempo IG2LS':tiempoig2ls

    })
with pd.ExcelWriter('analisis.xlsx') as writer:
    df.to_excel(writer, sheet_name = "ANÁLISIS DE RESULTADOS")
final=time.time()
print('Todo ha tardado',final-inicial)
```