Grado en Estadística

TRABAJO FIN DE GRADO

Problemas de Redes y Flujos

Presentado por:

Silvia Flores Ramos

Tutor:

Antonio Rufián Lizana

Sevilla, junio de 2021

Índice general

Prólogo

El Trabajo Fin de Grado que se presenta a continuación recibe el nombre de "Problemas de Redes y Flujos", y ha sido elaborado entre enero y junio de 2021. Como se verá en el desarrollo del mismo, existen diversos algoritmos que resuelven este tipo de problemas, los cuales son más comunes de lo que se piensa en la vida cotidiana.

Me gustaría agradecer a mi tutor Antonio por su orientación y entrega durante el proceso de realización de mi trabajo. También me gustaría dar las gracias a Pedro Luis Luque por facilitar la creación del TFG con su plantilla.
A mis amigas y compañeras de carrera: gracias por vuestro apoyo incondicional, y por seguir unidas en este proceso que hemos compartido.
Mi familia se merece un agradecimiento especial: vuestro apoyo y confianza en mí han conseguido que llegase hasta donde he llegado. Gracias como siempre.

Silvia Flores Ramos

Resumen

Las redes de flujos son un modelo que permite representar sistemas tales como mapas de carretera, redes de tuberías, conexiones de red, etc., desde un punto de vista abstracto. En este modelo también quedan representados los elementos que transitan en sus correspondientes sistemas: coches, líquidos, datos...
Gracias a su estudio, se resuelven problemas tan comunes como calcular cuántos litros de agua deben recorrer una determinada tubería en función de las necesidades de una población, o determinar el número máximo de vehículos que pueden circular por una carretera.

Esta investigación se centra, por tanto, en resolver estos problemas de forma eficiente. Para ello, el trabajo ha sido dividido en tres capítulos.

En el Capítulo 1, se realiza una introducción a la Teoría de Grafos, explicando su origen y planteando uno de los problemas más importantes en esta rama: El problema del camino más corto. Nos centramos en algoritmos como Dijkstra o A* para resolverlo.

En el Capítulo 2, se tratan Árboles, una importante parte de la Teoría de Grafos, que además es de gran utilidad para desarrollar futuros algoritmos. Estudiaremos algunas de sus propiedades, plantearemos el conocido problema del árbol de expansión de mínimo costo y resolveremos una ejemplificación del mismo.

Por último, en el Capítulo 3, se realiza un estudio de los flujos y redes, basándonos en los conceptos explicados en los dos capítulos anteriores. Indagamos en el problema más importante de una red de flujo: Problema de flujo máximo. Comezamos con su planteamiento, y aplicamos tres de los algoritmos más reconocidos para resolver este tipo de problema: Ford-Fulkerson, Edmonds-Karp y Dinic.

Abstract

Flow networks are a model that allows to represent systems such as maps of road, pipe networks, network connections, etc., from an abstract point of view. This model also represents the elements that pass through their corridors. relevant systems: cars, liquids, data...
Thanks to its study, common problems such as calculating how many liters of water must travel through a certain pipeline depending on the needs of a population, or determine the maximum number of vehicles that can circulate on a highway.

This research focuses, therefore, on solving these problems efficiently. For this reason, the work has been divided into three chapters.

In Chapter 1, an introduction to Graph Theory is made, explaining its origin and proposing one of the most important problems in this branch: Shortest Path problem. We focus on algorithms like Dijkstra or A* to solve it.

In Chapter 2, we discuss Trees, an important part of Graph Theory, which is also very useful for developing future algorithms. We will study some of its properties, we will propose the well-known Minimum Degree Spanning Tree problem (MDST) and we will resolve an exemplification of it.

Finally, in Chapter 3, a study of flows and networks is carried out, based on the concepts explained in the previous two chapters. We investigate the most important problem of a flow network: Maximum Flow problem. We start with its approach, and we apply three of the most recognized algorithms to solve this type problem: Ford-Fulkerson, Edmonds-Karp and Dinic.

Índice de figuras

Capítulo 1

Teoría de Grafos

1.1. Introducción

En este apartado vamos a introducir nociones básicas de la teoría de grafos, con el objetivo de ampliar posteriormente y con mayor facilidad los conocimientos en esta rama de las matemáticas.

1.2. El problema de los puentes de Königsberg

El origen de la teoría de grafos tiene lugar en 1736 en Königsberg, ciudad rusa conocida actualmente como Kaliningrado.

Esta ciudad es atravesada por el río Pregolia, dando lugar a la división de la misma en cuatro zonas que se unían mediante siete puentes:

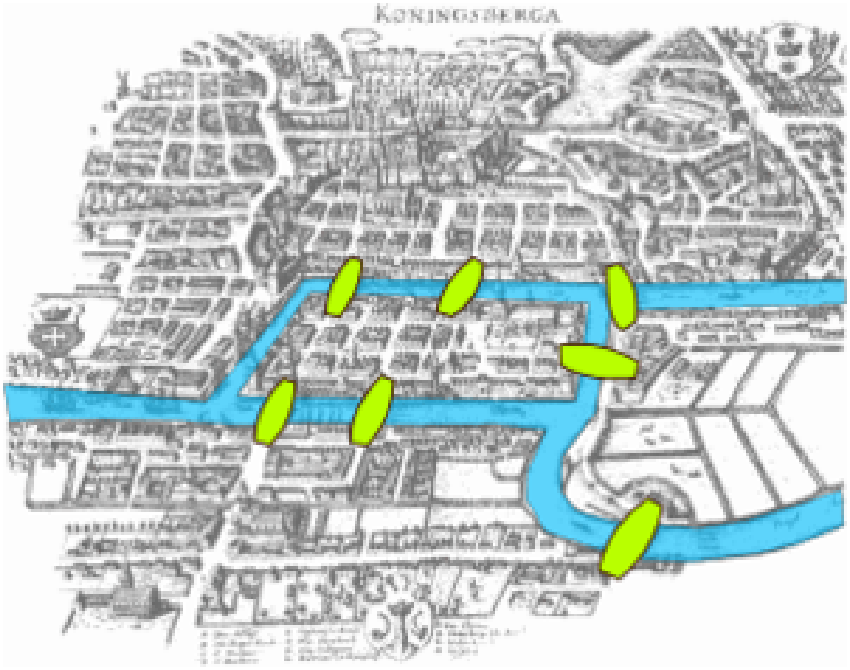
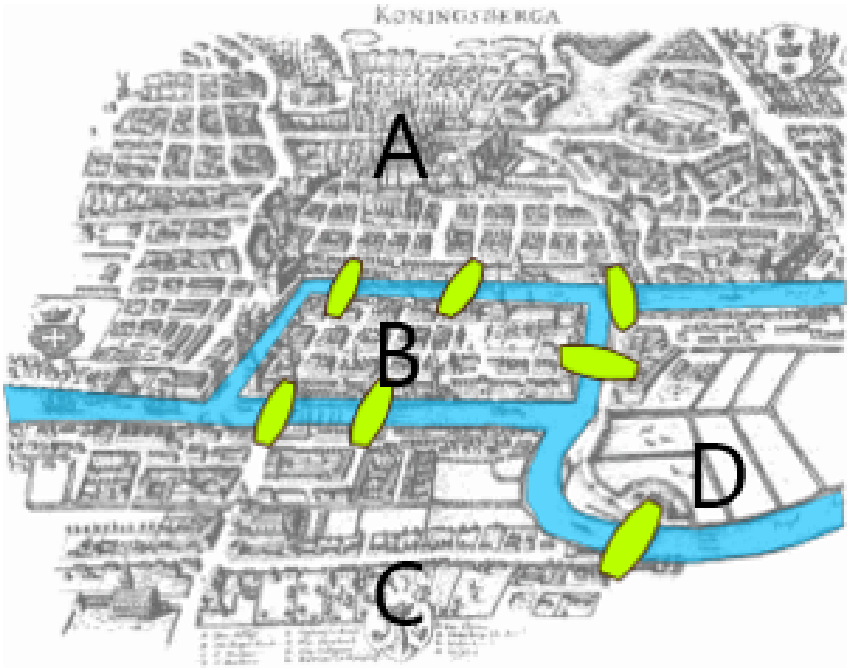


Figura 1.1: Mapa de Königsberg en la década de 1730

Leonhard Euler, uno de los matemáticos más importantes de la historia, planteó el siguiente problema: ¿Era posible atravesar los siete puentes, una sola vez cada uno, volviendo al punto de partida?

Este problema resultó viral entre los matemáticos e intelectuales de la época. Si nos fijamos en la imagen, obervamos que a lo sumo podremos cruzar seis de ellos de esa forma. Para hallar una explicación a esta respuesta, Euler representó cada una de las zonas como puntos, y los siete puentes, como líneas que unen dichos puntos.

Suponemos que cada zona de la ciudad se corresponde con las letras A, B, C y D con la siguiente distribución:



Así, el diagrama de Euler era de la forma:

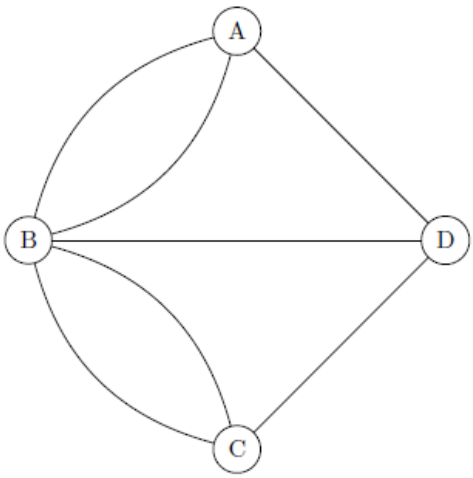


Figura 1.2: Diagrama de Euler.
Fuente: Elaboración propia

Si nos fijamos, a los puntos A, C y D llegan tres líneas, y al punto B, cinco.

Basándose en este problema, concluyó que la única manera de poder atraversar una sola vez cada puente, independientemente del punto de partida y llegada, era estableciendo que los puntos intermedios del recorrido correspondiente estuviesen conectados un número par de veces. Es decir, únicamente a los puntos inicial y final podrían llegar un número impar de líneas.

No obstante, el problema exige que el punto de partida y llegada sea el mismo, por lo que todos los puntos deberían estar conectados un número par de veces.

En conclusión, al estar todos los puntos conectados por un número impar de líneas, no solo no podemos atravesar todos los puentes una sola vez, sino que tampoco se podrá llegar al mismo punto del que se parte.

Se consigue así, en el año 1736, una primera aproximación al concepto de grafo.

1.3. Definiciones

Definición 1.3.1 *Un **grafo simple** G(V, E) consta de V, un conjunto no vacío de vértices, y de E, un conjunto de pares no ordenados de elementos distintos de V. A esos pares se les llama **aristas** o **lados**.*

En el ejemplo anterior, nuestro grafo está formado por los vértices A, B, C y D, así como por las aristas (A,B), (A,B) (A,D), (B,C), (B,C), (B,D), (C,D). Es decir, disponemos de cuatro vértices y 7 aristas.

Definición 1.3.2 *El **grado** de un vértice es el número de lados incidentes a él.*

Por tanto, los vértices A, C y D del grafo que describe el mapa de Königsberg tienen grado 3, mientras que el vértice B tiene grado 5.

Vamos a mostrar otro grafo, semejante al anterior pero con un pequeño matiz:

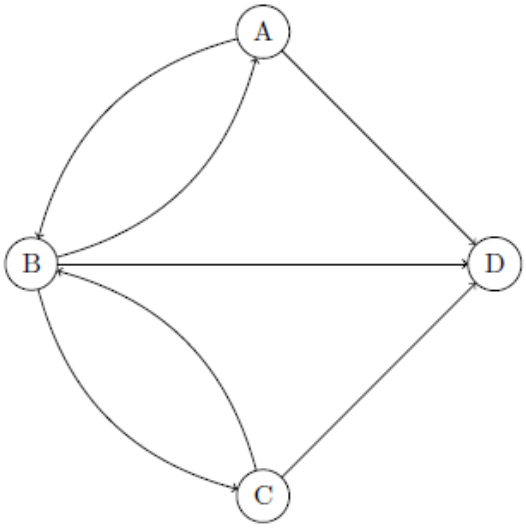


Figura 1.3: Ejemplo de grafo.
Fuente: Elaboración propia

Este tipo de grafo se denomina grafo dirigido.

Definición 1.3.3 *Un **grafo dirigido** o **digrafo** G = (V, E) consta de un conjunto V de vertices, un conjunto E de aristas, que son pares ordenados de elementos de V.*

En este caso, el conjunto de las aristas vendría dado por (A,B), (B,A) (A,D), (B,C), (C,B), (B,D), (C,D).

Hasta ahora hemos visto ejemplos de grafos sin pesos, que son los menos comunes. Imaginemos que queremos representar mediante grafos los distintos caminos entre un determinado punto inicial y un punto final, siendo dichos caminos unos más cortos que otros. Lo más lógico sería dar unos pesos a estos caminos (aristas en el grafo) para así poder identificar cuál convendría elegir en cada paso para llegar antes a nuestro destino.

Esto nos lleva a la definición de grafo ponderado:

Definición 1.3.4 *G es un **grafo ponderado** si a cada arista e de G se le asigna un número no negativo w(e) denominado **peso** o **longitud** de e. El peso (o longitud de un camino) en un grafo ponderado G se define como la suma de los pesos de las aristas del camino.*

Veamos un ejemplo:

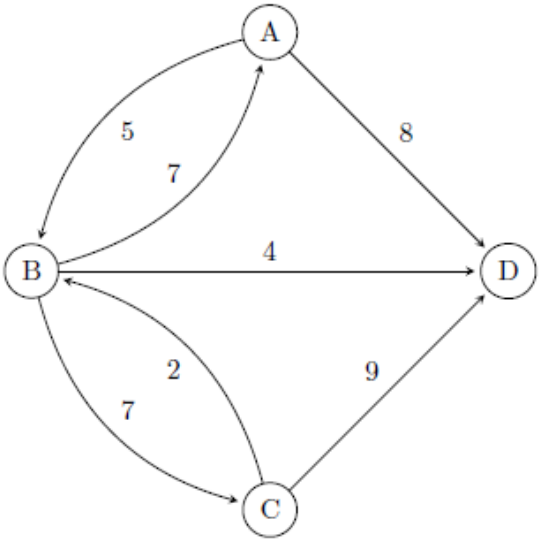


Figura 1.4: Ejemplo de grafo.
Fuente: Elaboración propia

1.4. Problema del camino más corto

En Teoría de Grafos, uno de los problemas más importantes es encontrar el camino más corto, o lo que es lo mismo, el camino de menor longitud (o costo) entre dos vértices cualesquiera.

Un claro ejemplo sería elegir el camino para llegar desde una ciudad A hasta una ciudad E, de manera que dispongamos de diferentes caminos y queramos recorrer la menor distancia posible.

Representamos los pesos o costos de nuestro grafo como $l(i, j)$.

A continuación, se explicarán los algortimos más utilizados para resolver este tipo de problema, y los aplicaremos sobre el siguiente ejemplo para entender sus pasos:

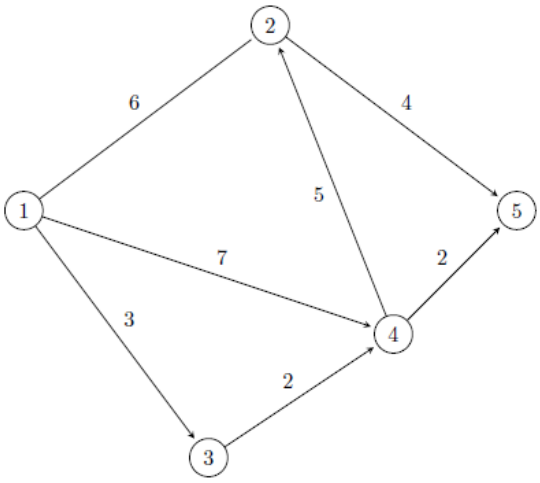


Figura 1.5: Grafo para calcular su camino más corto.
Fuente: Elaboración propia

1.4.1. Algoritmo de Dijkstra

Un método para hallar dicho camino, muy útil sobre todo para grafos (dirigidos o no) con muchos nodos y pesos positivos es aplicar el algoritmo de Dijkstra.

Definimos

- $d(i)$: longitud más corta desde el nodo inicial x_1, de donde parte el camino hasta el momento.
- $p(i)$: nodo anterior a x_i en el camino que se tiene en ese momento desde x_1.

1.4.1.1. Pasos del algoritmo

Paso 1

- $d(x_1) = 0$
- $p(x_1) = *$ (sin asignación)
- $d(j) = \infty$, $p(j) = *$, $\forall j \neq 1$
- El nodo x_1 es cerrado, es decir, su etiqueta es permanente y no modificable. Los demás, abiertos (sus etiquetas son provisionales y se pueden modificar)
- $k = 1$, que es el último nodo cerrado

Paso 2

- Se calcula $d(j) = min\{d(j), d(k) + l(k, j)\}$ para todos los x_j para los que exista una arista (x_k, x_j)

Paso 3

- Cerrar nodo con menor $d(j)$ de los nodos abiertos
- Si hay empate, se elige arbitrariamente
- Sea x_i el nodo cerrado

Paso 4

- Para encontrar el predecesor del nodo x_i, $p(i)$, se consideran las aristas (j, i) con j cerrado
- Se escoge el (j, i) tal que $d(i) - l(i, j) = d(j)$
- En caso de empate, escoger arbitrariamente
- $p(i) = j$

Paso 5

- Se cierra el nodo i
- Si todos los nodos están cerrados: STOP
- En caso contrario, $k = i$ y volver al paso 2

1.4.1.2. Aplicación

En las siguientes figuras, se representan en color verde los nodos abiertos, y en rojo, los cerrados.

El punto de partida es A, por lo que comenzamos cerrándolo. A continuación, se cierra el nodo conectado a A cuyo coste sea el mínimo de todos los conectados a este punto. En este caso, sería C.

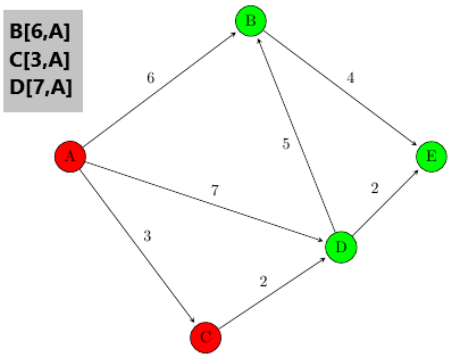


Figura 1.6: Algoritmo de Dijkstra.
Fuente: Elaboración propia

Tenemos que evaluar ahora las posibles salidas desde el nodo C. Como vemos, solo podemos tomar D. Observamos que de los nodos abiertos, el nodo D es el que tiene menor distancia respecto al origen, con $p(i) = C$, por lo que D pasa a ser cerrado:

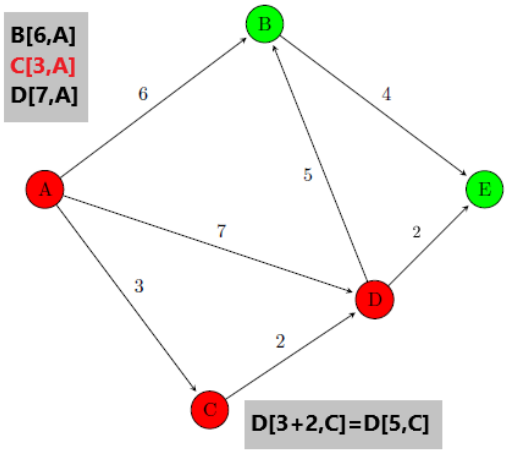


Figura 1.7: Algoritmo de Dijkstra.
Fuente: Elaboración propia

Procedemos a evaluar las posibles salidas desde el nodo D. El nodo abierto con menor distancia respecto al nodo de origen es B, con $p(i) = A$, por lo que B pasa a ser cerrado:

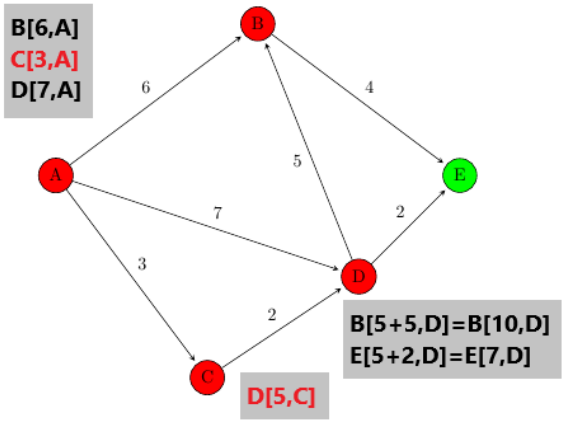


Figura 1.8: Algoritmo de Dijkstra.
Fuente: Elaboración propia

Nos queda, por tanto, evaluar las posibles salidas desde B. Vemos que solo está conectado con E, que es el nodo de llegada y a su vez el único nodo que nos queda por cerrar. No obstante, la menor distancia desde este nodo respecto al origen se obtiene para $p(i) = D$, por lo que cerramos E:

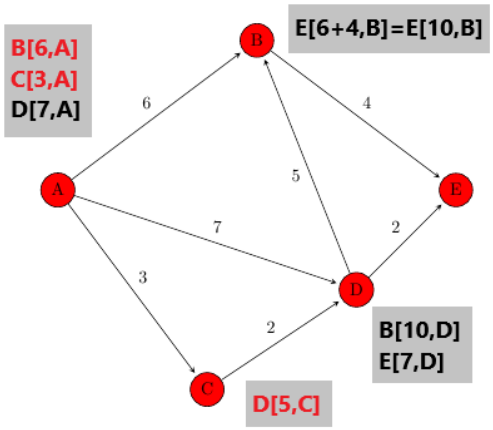


Figura 1.9: Algoritmo de Dijkstra.
Fuente: Elaboración propia

Obtenemos de esta forma el resultado final de nuestro problema aplicando el algoritmo de Dijkstra. Se muestra el camino más corto desde el nodo de origen A hasta el nodo de llegada E en color rojo:

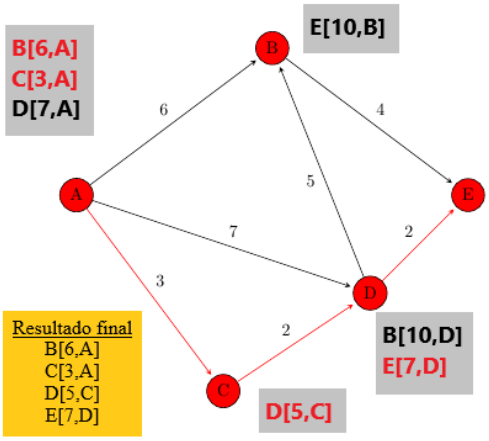


Figura 1.10: Algoritmo de Dijkstra.
Fuente: Elaboración propia

1.4.1.3. Implementación del algoritmo en R

Procedemos a resolver el problema del camino más corto del grafo dado en la Figura 1.2 en R.

Para ello, necesitamos crear una lista de cada arista con sus respectivos pesos. Es decir, creamos tres columnas:

- Las dos primeras, representan los vértices conectados.
- La tercera, apotará el valor del peso de la arista que los conecta.

Escribimos los datos en un *data frame* utilizando la función *edit*

```
data <- edit(data.frame())
```

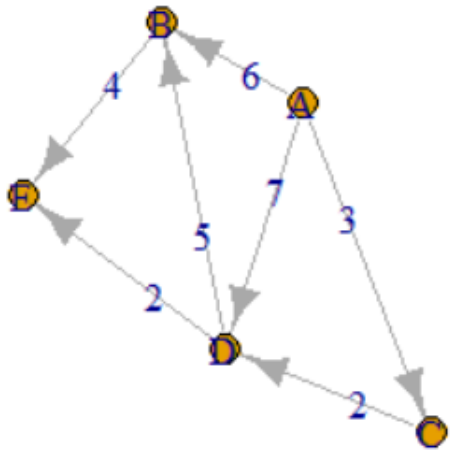
	nodo1	nodo2	peso
1	A	B	6
2	A	C	3
3	A	D	7
4	B	E	4
5	C	D	2
6	D	B	5
7	D	E	2
8			

Creamos a continuación un **igraph** a partir de este *data frame* de la siguiente manera:

```
install.packages("igraph")                    # Descargamos e instalamos
                                                igraph
library("igraph")                             # Cargamos igraph

# Empezamos a usar igraph
g <- graph.data.frame(data, directed = TRUE) # Creamos igraph
V(g)$name                                     # Nombres de los vértices
E(g)$peso                                     # Peso de las aristas
plot(g, edge.label = paste(E(g)$weight, sep = "")) # Representación del
                                                     grafo
```

Obtenemos así:




```
[1] "A" "B" "C" "D" "E"
[1] 6 3 7 4 2 5 2
```

Pasamos a calcular la distancia y el camino más corto entre los vértices A y E mediante el algoritmo de Dijkstra, que es el que utiliza **igraph** por defecto:

```
# Camino más corto entre A y E
sp <- shortest.paths(g, v = "A", to = "E")
sp[]                                          # Distancia

gsp <- get.shortest.paths(g, from = "1", to = "5")
V(g)[gsp$vpath[[1]]]                          # Secuencia de vértices
```

```
> # Camino más corto entre A y E
> sp <- shortest.paths(g, v = "A", to = "E")
> sp[]                                         # Distancia
  E
A 7
> gsp <- get.shortest.paths(g, from = "A", to = "E")
> V(g)[gsp$vpath[[1]]]                         # Secuencia de vértices
+ 4/5 vertices, named, from a08905f:
[1] A C D E
```

Observamos que mediante **igraph** obtenemos el mismo resultado: el camino más corto entre los nodos A y E viene dado por $\{A, C, D, E\}$, con un coste de 7.

1.4.2. Algoritmo A*

Otro método útil para resolver el problema del camino más corto es el Algoritmo A*. A diferencia del anterior, este algoritmo no comprueba que todas las rutan existen, por lo que no nos devolverá el camino óptimo, sino uno de los mejores. El Algoritmo A* utiliza una función de evaluación heurística h.

Definición 1.4.1 *Una **función de evaluación heurística** es una función que hace corresponder situaciones del problema con números*

Si el valor de la función h es $h(x_i) = 0\ \forall i$, entonces el algoritmo A* se comportará como el algoritmo de Dijkstra.

Cabe destacar que este algoritmo se utiliza para grafos con pesos positivos, independientemente de si el grafo es dirigido o no lo es. Vamos a suponer que nuestro nodo inicial es x_1, y el nodo de destino x_n.

La función de evaluación $f(x_i)$ se define usando otras dos funciones:

- $g(x_i)$: Indica el costo del camino desde x_1 hasta x_i.
- $h(x_i)$: Indica el costo estimado desde el nodo x_i hasta el nodo final x_n.

Por otra parte, definimos un conjunto Q que contendrá a los nodos en exploración y un conjunto P con los nodos ya explorados.

1.4.2.1. Pasos del algoritmo

Paso 1

Definimos

- x_1 nodo inicial
- $Q = \{\}$
- $P = \{\}$

Paso 2

- Calcular $f(x_1)$
- $Q = \{x_1\}$

Paso 3

- Seleccionar x_i tal que $f(x_i) = \text{mín}_{x_k \in Q}\, f(x_k)$
- $P = P \cup \{x_i\}$
- $Q = \{\}$

Paso 4

Para cada nodo vecino x_j de x_i

- Calcular $f(x_j)$
- $Q = Q \cup \{x_j\}$

Paso 5

- Si x_j tal que $\text{mín}_{x_j \in Q}\, f(x_j)$ es el nodo objetivo x_n: hacer $P = P \cup \{x_n\}$ y STOP. La solución es el camino en P
- En caso contrario, volver al paso 3

1.4.2.2. Aplicación

Buscaremos de nuevo el camino más corto desde el nodo A hasta el nodo E.

Como sabemos, para aplicar el algoritmo A* debemos dar una estimación de los costos desde cada uno de los nodos hasta el nodo final, en este caso E. Estas estimaciones vienen indicadas mediante la función h anteriormente explicada.

Supondremos:

$h(x_A)$	8
$h(x_B)$	4
$h(x_C)$	6
$h(x_D)$	2
$h(x_E)$	0

Se mostrarán en color rojo los nodos pertenecientes a P.

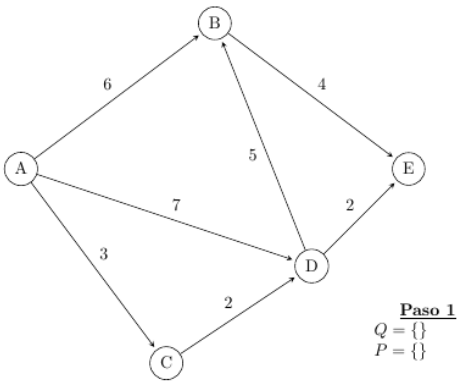


Figura 1.11: Algoritmo A*.
Fuente: Elaboración propia

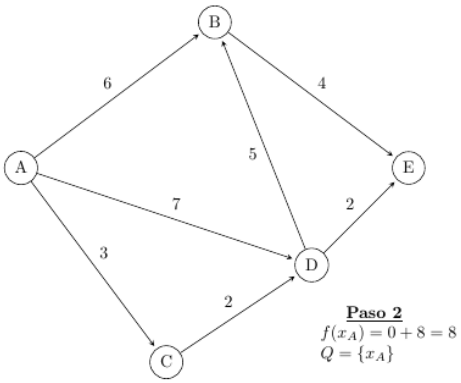


Figura 1.12: Algoritmo A*.
Fuente: Elaboración propia

Como Q únicamente está formado por x_A, lo añadimos directamente a P:

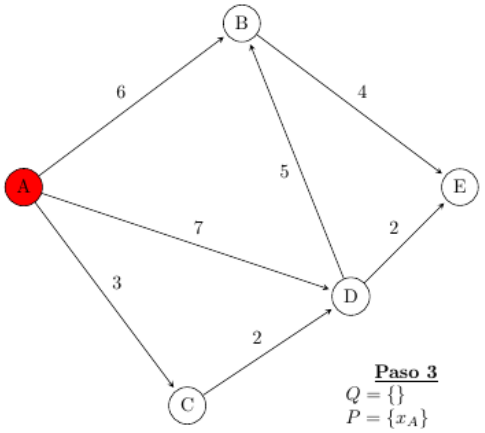


Figura 1.13: Algoritmo A*.
Fuente: Elaboración propia

Los nodos vecinos de x_A son x_B, x_C y x_D:

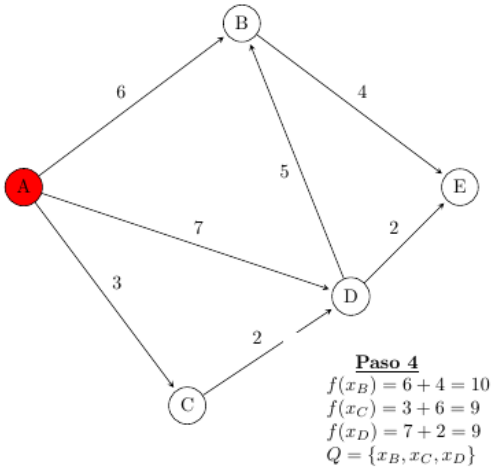


Figura 1.14: Algoritmo A*.
Fuente: Elaboración propia

Obtenemos

$$\min_{x_B, x_C, x_D} \{f(x_B), f(x_C), f(x_D)\} = 9 = f(x_C) = f(x_D)$$

Como ni x_C ni x_D son el nodo objetivo, retomamos el paso 3, tomando x_i como cualquiera de los dos anteriores. Elegimos x_C al azar:

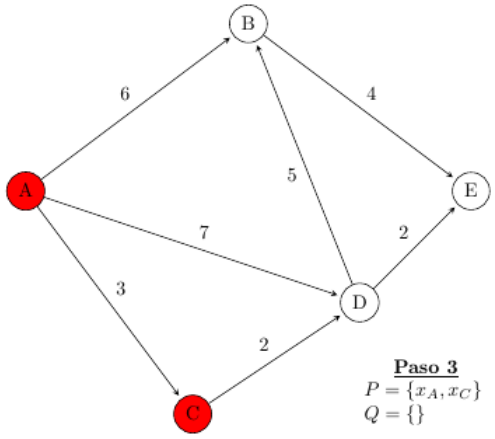


Figura 1.15: Algoritmo A*.
Fuente: Elaboración propia

El nodo vecino de x_C es x_D:

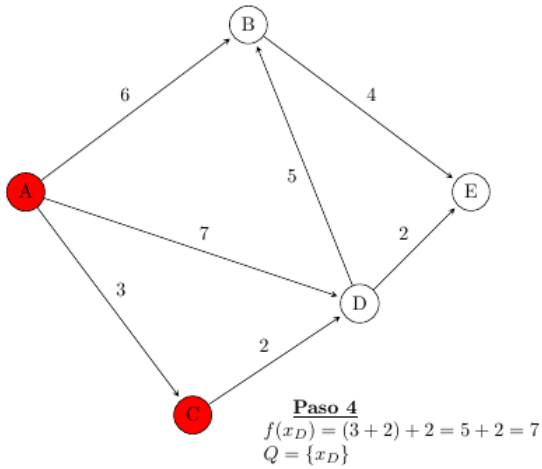


Figura 1.16: Algoritmo A*.
Fuente: Elaboración propia

Como x_D es el único elemento de Q y no es el nodo objetivo, retomamos el paso 3:

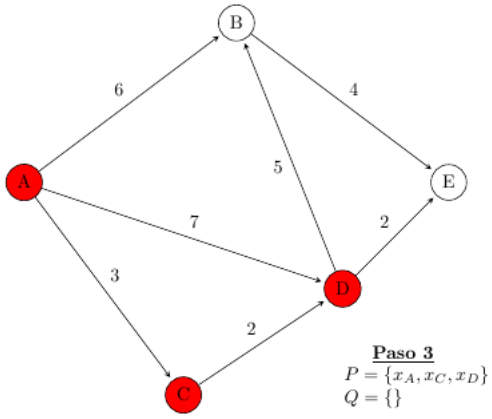


Figura 1.17: Algoritmo A*.
Fuente: Elaboración propia

Los nodos vecinos de x_D son x_B y x_E:

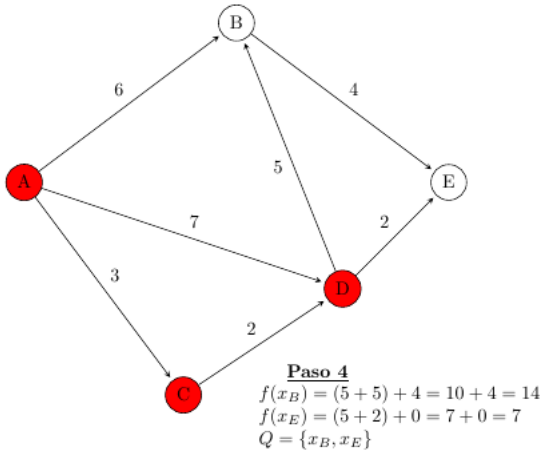


Figura 1.18: Algoritmo A*.
Fuente: Elaboración propia

Obtenemos

$$\min_{x_B, x_E} \{f(x_B), f(x_E)\} = 7 = f(x_E)$$

Como x_E es el nodo objetivo:

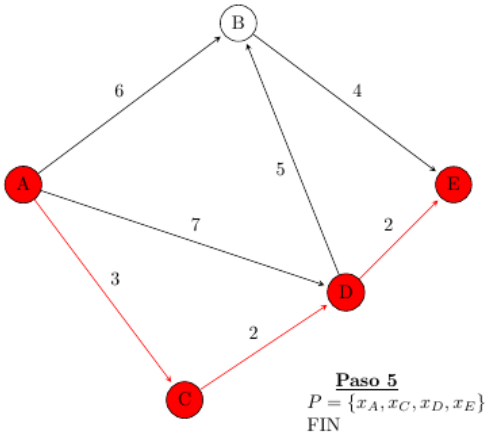


Figura 1.19: Algoritmo A*.
Fuente: Elaboración propia

Obtenemos de esta forma el resultado final de nuestro problema aplicando el algoritmo A*. El camino más corto desde el nodo de inicio A hasta el nodo de llegada E es aquel que pasa por los nodos A, C, D y E según este algoritmo.

1.4.3. Algoritmo de Floyd

Hemos observado que en el algoritmo de Dijkstra se obtiene el camino más corto desde un punto inicial al resto de puntos, y que al aplicar el algoritmo A* se consigue el camino más corto entre dos puntos específicos. En cambio, mediante el algoritmo de Floyd podemos conseguir el camino más corto entre cualquier par de nodos.

Para aplicar este algoritmo definimos dos matrices:

$$D_0(i,j) = \begin{cases} l(i,j) & \text{si i y j conectados,} \\ \infty & \text{caso contrario.} \end{cases}$$

$$P_0(i,j) = \begin{cases} i & \text{si } i \neq j, \\ - & \text{caso contrario.} \end{cases}$$

1.4.3.1. Pasos del algoritmo

Paso 1

- $k = 1$

Paso 2

- En $D_k(i,j)$, $(i,j) = \text{mín}\{(i,j), (i,k) + (k,j)\}$

Paso 3

- Mostrar $P_k(i,j)$, sustituyendo los valores de las posiciones modificadas en D por k

Paso 4

- Siendo n el número total de nodos, si $k = n$, STOP
- Si $k < n$, hacer $k = k + 1$ y volver al Paso 2

Output: El algoritmo devuelve las matrices P y D, que se corresponden con el camino mínimo y su costo asociado.

1.4.3.2. Aplicación

Vamos a encontrar el camino más corto así como los costos entre todos los nodos del grafo de la Figura 1.2.

Comenzamos sustituyendo el nombre de los nodos (A, B, C...) por sus correspondientes valores ordinales, es decir, A por el número 1, B por el número 2, etc.:

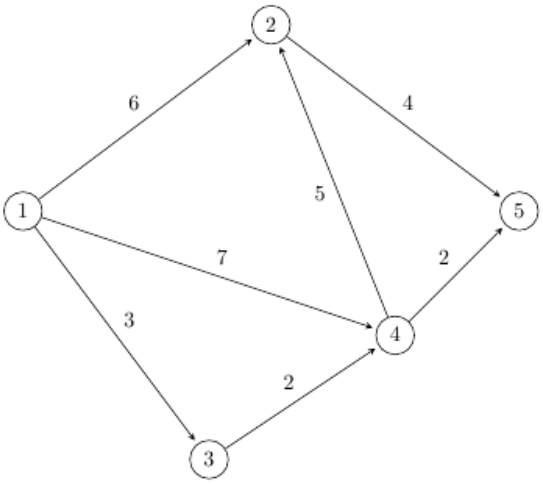


Figura 1.20: Algoritmo de Floyd.
Fuente: Elaboración propia

Comenzamos construyendo las matrices D_0 y P_0:

$$D_0 = \begin{pmatrix} 0 & 6 & 3 & 7 & \infty \\ \infty & 0 & \infty & \infty & 4 \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 5 & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$P_0 = \begin{pmatrix} - & 1 & 1 & 1 & 1 \\ 2 & - & 2 & 2 & 2 \\ 3 & 3 & - & 3 & 3 \\ 4 & 4 & 4 & - & 4 \\ 5 & 5 & 5 & 5 & - \end{pmatrix}$$

Paso 1

$k = 1$

Paso 2

Como $l(i,j) = 0, \forall i = j$, no es necesario que apliquemos el mínimo entre los valores dados en este paso, pues 0 será siempre el mínimo valor posible.

(i,j)	$\text{mín}\{l(i,j), l(i,k) + l(k,j)\}$
$(1,2)$	$\text{mín}\{6, 0+6\} = 6$
$(1,3)$	$\text{mín}\{3, 0+3\} = 3$
$(1,4)$	$\text{mín}\{7, 0+7\} = 7$
$(1,5)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(2,1)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(2,3)$	$\text{mín}\{\infty, \infty+3\} = \infty$
$(2,4)$	$\text{mín}\{\infty, \infty+7\} = \infty$
$(2,5)$	$\text{mín}\{4, \infty+\infty\} = 4$
$(3,1)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(3,2)$	$\text{mín}\{\infty, \infty+6\} = \infty$
$(3,4)$	$\text{mín}\{2, \infty+7\} = 2$
$(3,5)$	$\text{mín}\{\infty, \infty+\infty\} = \infty$
$(4,1)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(4,2)$	$\text{mín}\{5, \infty+6\} = 5$
$(4,3)$	$\text{mín}\{\infty, \infty+3\} = \infty$
$(4,5)$	$\text{mín}\{2, \infty+\infty\} = 2$
$(5,1)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(5,2)$	$\text{mín}\{\infty, \infty+6\} = \infty$
$(5,3)$	$\text{mín}\{\infty, \infty+3\} = \infty$
$(5,4)$	$\text{mín}\{\infty, \infty+7\} = \infty$

Por tanto, para $k = 1$ no se modifica nada.

Es decir, $D_1 = D_0$.

Paso 3

$P_1 = P_0$.

Paso 4

$k = 2$ y volvemos al paso 2.

Paso 2

(i,j)	$\text{mín}\{l(i,j), l(i,k)+l(k,j)\}$
$(1,2)$	$\text{mín}\{6, 6+0\} = 6$
$(1,3)$	$\text{mín}\{3, 6+\infty\} = 3$
$(1,4)$	$\text{mín}\{7, 6+\infty\} = 7$
$l(1,5)$	$\text{mín}\{\infty, 6+4\} = \text{mín}\{\infty, 10\} = 10$
$(2,1)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(2,3)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(2,4)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(2,5)$	$\text{mín}\{4, 0+4\} = 4$
$(3,1)$	$\text{mín}\{\infty, \infty+\infty\} = \infty$
$(3,2)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(3,4)$	$\text{mín}\{2, \infty+\infty\} = 2$
$(3,5)$	$\text{mín}\{\infty, \infty+4\} = \infty$
$(4,1)$	$\text{mín}\{\infty, \infty\} = \infty$
$(4,2)$	$\text{mín}\{5, 5+0\} = 5$
$(4,3)$	$\text{mín}\{\infty, 5+\infty\} = \infty$
$(4,5)$	$\text{mín}\{2, 5+4\} = \text{mín}\{2, 9\} = 2$
$(5,1)$	$\text{mín}\{\infty, \infty+\infty\} = \infty$
$(5,2)$	$\text{mín}\{\infty, \infty+0\} = \infty$
$(5,3)$	$\text{mín}\{\infty, \infty+\infty\} = \infty$
$(5,4)$	$\text{mín}\{\infty, \infty+\infty\} = \infty$

Por tanto, para $k = 2$ cambiamos el valor de la posición (1,5) por 10 en la matriz D_2, y sustituimos el valor de la misma posición en la matriz P_2 por k (2 en este caso).

$$D_2 = \begin{pmatrix} 0 & 6 & 3 & 7 & 10 \\ \infty & 0 & \infty & \infty & 4 \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 5 & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Paso 3

$$P_2 = \begin{pmatrix} - & 1 & 1 & 1 & 2 \\ 2 & - & 2 & 2 & 2 \\ 3 & 3 & - & 3 & 3 \\ 4 & 4 & 4 & - & 4 \\ 5 & 5 & 5 & 5 & - \end{pmatrix}$$

Paso 4

$k = 3$ y volvemos al paso 2.

Paso 2

(i,j)	$\min\{l(i,j), l(i,k)+l(k,j)\}$
$(1,2)$	$\min\{6, 3+\infty\} = 6$
$(1,3)$	$\min\{3, 3+0\} = 3$
$(1,4)$	$\min\{7, 3+2\} = \min\{7,5\} = 5$
$(1,5)$	$\min\{10, 3+\infty\} = 10$
$(2,1)$	$\min\{\infty, \infty+\infty\} = \infty$
$(2,3)$	$\min\{\infty, \infty+0\} = \infty$
$(2,4)$	$\min\{\infty, \infty+2\} = \infty$
$(2,5)$	$\min\{4, \infty+\infty\} = 4$
$(3,1)$	$\min\{\infty, 0+\infty\} = \infty$
$(3,2)$	$\min\{\infty, 0+\infty\} = \infty$
$(3,4)$	$\min\{2, 0+2\} = 2$
$(3,5)$	$\min\{\infty, 0+\infty\} = \infty$
$(4,1)$	$\min\{\infty, \infty+\infty\} = \infty$
$(4,2)$	$\min\{5, \infty+\infty\} = 5$
$(4,3)$	$\min\{\infty, \infty+0\} = \infty$
$(4,5)$	$\min\{2, \infty+\infty\} = 2$
$(5,1)$	$\min\{\infty, \infty+\infty\} = \infty$
$(5,2)$	$\min\{\infty, \infty+\infty\} = \infty$
$(5,3)$	$\min\{\infty, \infty+0\} = \infty$
$(5,4)$	$\min\{\infty, \infty+2\} = \infty$

Por tanto, para $k=3$ cambiamos el valor de la posición (1,4) por 5 en la matriz D_3, y sustituimos el valor de la misma posición en la matriz P_3 por k (3 en este caso).

$$D_3 = \begin{pmatrix} 0 & 6 & 3 & 5 & 10 \\ \infty & 0 & \infty & \infty & 4 \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 5 & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Paso 3

$$P_3 = \begin{pmatrix} - & 1 & 1 & 3 & 2 \\ 2 & - & 2 & 2 & 2 \\ 3 & 3 & - & 3 & 3 \\ 4 & 4 & 4 & - & 4 \\ 5 & 5 & 5 & 5 & - \end{pmatrix}$$

Paso 4

$k=4$ y volvemos al paso 2.

Paso 2

(i,j)	$\min\{l(i,j), l(i,k)+l(k,j)\}$
$(1,2)$	$\min\{6, 5+5\} = \min\{6, 10\} = 6$
$(1,3)$	$\min\{3, 5+\infty\} = 3$
$(1,4)$	$\min\{5, 5+0\} = 5$
$(1,5)$	$\min\{10, 5+2\} = \min\{10, 7\} = 7$
$(2,1)$	$\min\{\infty, \infty+\infty\} = \infty$
$(2,3)$	$\min\{\infty, \infty+\infty\} = \infty$
$(2,4)$	$\min\{\infty, \infty+0\} = \infty$
$(2,5)$	$\min\{4, \infty+2\} = 4$
$(3,1)$	$\min\{\infty, 2+\infty\} = \infty$
$(3,2)$	$\min\{\infty, 2+5\} = \min\{\infty, 7\} = 7$
$(3,4)$	$\min\{2, 2+0\} = 2$
$(3,5)$	$\min\{\infty, 2+2\} = \min\{\infty, 4\} = 4$
$(4,1)$	$\min\{\infty, 0+\infty\} = \infty$
$(4,2)$	$\min\{5, 0+5\} = 5$
$(4,3)$	$\min\{\infty, 0+\infty\} = \infty$
$(4,5)$	$\min\{2, 0+2\} = 2$
$(5,1)$	$\min\{\infty, \infty+\infty\} = \infty$
$(5,2)$	$\min\{\infty, \infty+5\} = \infty$
$(5,3)$	$\min\{\infty, \infty+\infty\} = \infty$
$(5,4)$	$\min\{\infty, \infty+0\} = \infty$

Por tanto, para $k = 4$ cambiamos en la matriz D_4 el valor de la posición (1,5) por 5; el de la posición (3,2) por 7, y el valor de la posición (3,5) por 4. Además, sustituimos los valores de estas posiciones en la matriz P_4 por k (4 en este caso).

$$D_4 = \begin{pmatrix} 0 & 6 & 3 & 5 & 7 \\ \infty & 0 & \infty & \infty & 4 \\ \infty & 7 & 0 & 2 & 4 \\ \infty & 5 & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Paso 3

$$P_4 = \begin{pmatrix} - & 1 & 1 & 3 & 4 \\ 2 & - & 2 & 2 & 2 \\ 3 & 4 & - & 3 & 4 \\ 4 & 4 & 4 & - & 4 \\ 5 & 5 & 5 & 5 & - \end{pmatrix}$$

Paso 4

$k = 5$ y volvemos al paso 2.

Paso 2

(i,j)	$\text{mín}\{l(i,j), l(i,k)+l(k,j)\}$
$(1,2)$	$\text{mín}\{6, 7+\infty\} = 6$
$(1,3)$	$\text{mín}\{3, 7+\infty\} = 3$
$(1,4)$	$\text{mín}\{5, 7+\infty\} = 5$
$(1,5)$	$\text{mín}\{7, 7+0\} = 7$
$(2,1)$	$\text{mín}\{\infty, 4+\infty\} = \infty$
$(2,3)$	$\text{mín}\{\infty, 4+\infty\} = \infty$
$(2,4)$	$\text{mín}\{\infty, 4+\infty\} = \infty$
$(2,5)$	$\text{mín}\{4, 4+0\} = 4$
$(3,1)$	$\text{mín}\{\infty, 4+\infty\} = \infty$
$(3,2)$	$\text{mín}\{7, 4+\infty\} = 7$
$(3,4)$	$\text{mín}\{2, 4+\infty\} = 2$
$(3,5)$	$\text{mín}\{4, 4+0\} = 4$
$(4,1)$	$\text{mín}\{\infty, 2+\infty\} = \infty$
$(4,2)$	$\text{mín}\{5, 2+\infty\} = 5$
$(4,3)$	$\text{mín}\{\infty, 2+\infty\} = \infty$
$(4,5)$	$\text{mín}\{2, 2+0\} = 2$
$(5,1)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(5,2)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(5,3)$	$\text{mín}\{\infty, 0+\infty\} = \infty$
$(5,4)$	$\text{mín}\{\infty, 0+\infty\} = \infty$

Por tanto, para $k = 5$ no se modifica nada.

Es decir, $D_5 = D_4$.

$$D_5 = \begin{pmatrix} 0 & 6 & 3 & 5 & 7 \\ \infty & 0 & \infty & \infty & 4 \\ \infty & 7 & 0 & 2 & 4 \\ \infty & 5 & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Paso 3

$P_5 = P_4$, es decir,

$$P_5 = \begin{pmatrix} - & 1 & 1 & 3 & 4 \\ 2 & - & 2 & 2 & 2 \\ 3 & 4 & - & 3 & 4 \\ 4 & 4 & 4 & - & 4 \\ 5 & 5 & 5 & 5 & - \end{pmatrix}$$

Paso 4

Tenemos que $k = 5 = n$. Por tanto, STOP.

Resultado final

La matriz P_5 proporciona la información correspondiente a los caminos mínimos entre todos los nodos, y D_5 indica el coste asociado. Por ejemplo, el camino mínimo para ir

desde el nodo 1 al nodo 4 es $1-3-4$, con un coste de 5. De igual forma, podemos observar que el camino mínimo para ir desde el nodo 3 al nodo 5 es $3-4-5$, con un coste de 4.

Calculemos ahora el camino mínimo desde el nodo 1 al nodo 5 (camino calculado con el resto de algoritmos). Vemos que dicho camino viene dado por $1-3-4-5$, con un costo de 7.

Capítulo 2

Árboles

2.1. Introducción

En este apartado nos centraremos en un tipo concreto de grafo: Árboles. Se introducirán definiciones básicas, así como propiedades de los mismos, que nos ayudarán a entender esta parte de la Teoría de Grafos y a resolver problemas de interés.

2.2. Definiciones

Definición 2.2.1 *Se llama* ***subgrafo*** *de un grafo* $G = (V, A)$ *a un grafo* $G' = (V', A')$*, donde* $V' \subseteq V$ *y* $A' \subseteq A$*.*

Definición 2.2.2 *Se denomina* ***camino*** *a una secuencia de nodos unidos por aristas.*

Definición 2.2.3 *Un* ***ciclo*** *es un camino que empieza y acaba en el mismo nodo, es decir, es un camino cerrado.*

Gracias a estas definiciones, podemos explicar fácilmente el concepto de *árbol* como sigue:

Definición 2.2.4 *Se llama* ***árbol*** *a un subgrafo con todos sus vértices conectados, pero que no contiene ciclos. Si dicho árbol incluye a todos los nodos, se denomina* ***árbol de expansión****. En un grafo dirigido, los árboles de expansión poseen un nodo "raíz", conectado a todos los demás nodos de forma única.*

De esta forma, un árbol podría representarse como sigue:

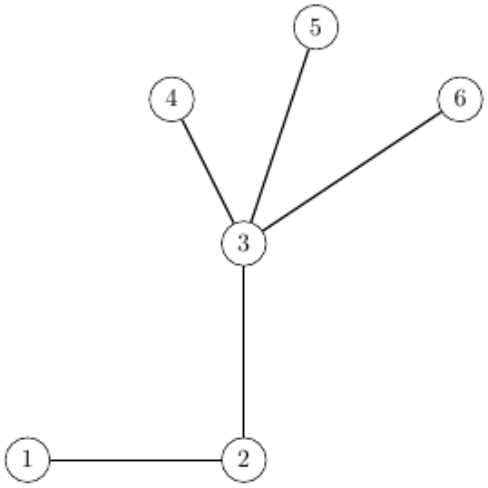


Figura 2.1: Árbol de 6 vértices.
Fuente: Elaboración propia

Como vemos, todos los vértices están conectados y forman un camino abierto, es decir, no se puede empezar y acabar en el mismo nodo (no contiene ciclos).

Suponemos ahora que disponemos del mismo árbol, pero esta vez dirigido de la siguiente manera:

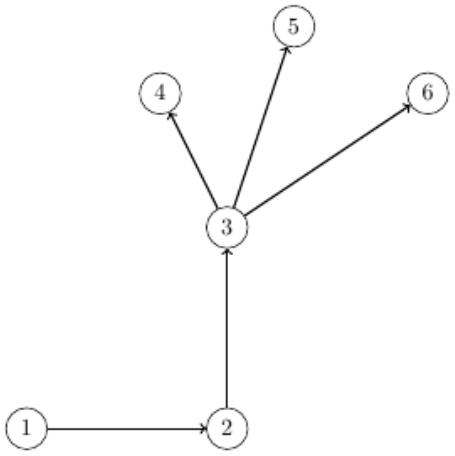


Figura 2.2: Árbol dirigido.
Fuente: Elaboración propia

Observamos que el nodo 1 es el único que está conectado con el resto de nodos. Por tanto, 1 sería el denominado nodo "raíz".

Definición 2.2.5 *Llamaremos* ***hoja*** *a todo nodo del árbol que tenga grado 1. Por tanto, los nodos hojas serán los nodos extremos del árbol.*

En el ejemplo anterior, los nodos hojas vienen representados por los nodos 4, 5 y 6.

2.3. Propiedades

A continuación, mostraremos algunas de las principales propiedades de los árboles, así como sus respectivas demostraciones:

Teorema 2.3.1 Sea T un árbol. Entonces

1. Todo árbol con al menos dos vértices tiene al menos dos hojas.
2. Si v es una hoja de T , entonces $T - \{v\}$ formará otro árbol.

Demostración. La demostración sería la siguiente:

1. En todo grafo con al menos dos vértices, cada extremo de un camino de al menos una arista tendrá como único nodo adyacente a su vecino en el camino. Todo grafo conexo con al menos dos vértices, tendrá al menos una arista, por lo que existirá un camino maximal. Los extremos de dicho camino serán hojas, por lo que el árbol tendrá al menos dos hojas.
2. Sea v una hoja del árbol T. Llamamos $T' = T - \{v\}$. Una hoja solo es adyacente a otro nodo, ya que es de grado 1, por lo que no puede pertenecer a un camino entre dos vértices distintos a v. Por tanto, todo camino entre dos vértices u y w de T, también existirá en T', siempre que $u \neq v$ y $w \neq v$. Por tanto, T' también será un grafo conexo, y permanecerá sin ciclos, puesto que al eliminar un eje no se pueden formar ciclos nuevos.

■

Teorema 2.3.2 Sea T un grafo de n nodos. Son equivalentes:

1. T es conexo y sin ciclos.
2. T es conexo y tiene $n - 1$ arcos.
3. T es conexo, pero al quitar un arco se vuelve inconexo.
4. T no tiene ciclos, pero al agregar un nuevo arco se forma un ciclo.
5. Solo hay un camino que conecte dos nodos.

Demostración. Demostraremos que toda afirmación implica a la siguiente, así como que la útilma afirmación implica a la primera

- $\underline{1 \Rightarrow 2}$
  Probemos por inducción que T tiene $n - 1$ arcos. En primer lugar, si $n = 1$, el grafo solo tendrá un nodo y ningún eje, por lo que se verifica la hipótesis. Supongamos ahora que se verifica para $n - 1$. Sea por tanto el grafo T con n nodos. Entonces, al ser conexo y sin ciclos, tendrá forzosamente un nodo de grado 1 (nodo "raíz"), que llamaremos v. Entonces, el grafo $T' = T - \{v\}$ seguirá siendo conexo y sin ciclos, con $n - 1$ vértices. Por hipótesis de inducción, el número de arcos de T' será el número de nodos menos 1, es decir, $n - 2$. Por tanto, como el número de arcos de T será uno más que de T' al haber eliminado uno, concluimos finalmente que T posee $n - 1$ ejes.

- $\underline{2 \Rightarrow 3}$
  Sean $T_1, ..., T_k$ las k componentes conexas de T, que serán conexas y sin ciclos. Llamamos $E[T_i]$ al número de arcos de T_i y $V(T_i)$ al número de vértices. De esta forma, se tendrá que para cada $i = 1, ..., k$, $|E[T_i]| = |V[T_i]| - 1$. Por tanto,

  $$|E[T]| = \sum_{i=1}^{k} |E[T_i]| = \sum_{i=1}^{k} (|V[T_i]| - 1) = n - k$$

  Si se eliminase un arco de T, se tendría que $|E[T]| = n - 2$, es decir, tendríamos $k = 2$. Así, T tendría dos componentes conexas, por lo que no sería conexo.

- $\underline{3 \Rightarrow 4}$
  Al existir un ciclo en un grafo, se tiene que existen al menos dos caminos para ir de un nodo del ciclo a otro. Por hipótesis, al quitar un arco de T, el grafo deja de ser conexo, por lo que T no posee ningún ciclo. Además, al ser T conexo, se tiene que existe al menos un camino que une a todos los vértices. Por tanto, al añadir un nuevo eje entre dos nodos se obtiene un nuevo camino entre ellos, formando así un ciclo.

- $\underline{4 \Rightarrow 5}$
  Razonaremos por reducción al absurdo. Supongamos que para un par de nodos u y v existen dos caminos que los conectan. Luego forzosamente debe existir un ciclo que los contenga, lo cual contradice la hipótesis (4). Supongamos ahora que existe un nodo w en T que no está conectado con ningún otro nodo. Entonces, al añadir un arco desde w, el grado de este nodo será 1. Se vuelve a contradecir la hipótesis (4) porque un nodo de grado 1 nunca puede formar parte de un ciclo. Por tanto, podemos concluir que existe un único camino entre cada par de nodos del grafo.

- $\underline{5 \Rightarrow 1}$
  Al existir un camino entre cada par de nodos del grafo, T será obviamente conexo. Por hipótesis, no existe más de un camino entre ningún par de nodos, luego T no posee ningún ciclo.

■

2.4. Problema del árbol de expansión de mínimo costo

Como vimos anteriormente, el árbol de expansión de un grafo es un subgrafo conexo y sin ciclos que contiene todos los nodos del grafo. Suponemos que este grafo es ponderado y tiene un peso o un costo asociado a las aristas. Si queremos encontrar el camino de mínimo costo entre dos vértices cualesquiera, en el árbol de expansión se traduce en solucionar el problema del árbol de expansión de mínimo costo (MST). Claramente, si el grafo tiene n vértices, el árbol de expansión poseerá $n - 1$ aristas. Hallar el MST es otra manera de solucionar el problema del camino más corto visto en el capítulo anterior, pues al hallar el árbol de mínimo coste, se tiene un camino entre cada par de vértices con el menor coste posible.

A continuación, se explicarán los algoritmos más utilizados para resolver este problema, y los aplicaremos sobre un grafo no dirigido y otro dirigido, que son:

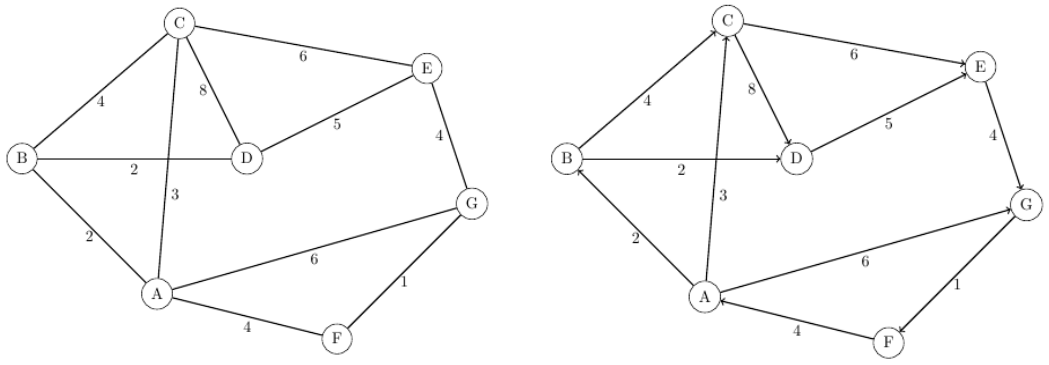


Figura 2.3: Grafos para el problema del MST.
Fuente: Elaboración propia

2.4.1. Algoritmo de Prim

Este algoritmo calcula el MST en grafos no dirigidos y ponderados. Si partimos de un grafo no conexo, el algoritmo devolverá el MST de una de sus componentes conexas. Por otra parte, si el grafo es dirigido, el algoritmo devolverá un árbol de expansión, sin asegurar que sea el de mínimo costo.

2.4.1.1. Pasos del algoritmo

Suponemos un grafo G con n nodos.

Paso 1

- Elegir aleatoriamente un vértice i
- i forma parte del árbol
- Hallar el nodo j más cercano a i y añadirlo al árbol
- $k = 1$

Paso 2

- Si $k = n - 1$, STOP. Todos los nodos están en el árbol
- Si $k < n - 1$
  - Tomar el vértice j no conectado al árbol que sea más cercano a cualquier vértice del árbol, es decir, el de menor costo
  - Añadir j al árbol

Paso 3

- $k = k + 1$ y volver al paso 2

2.4.1.2. Aplicación

En las siguientes figuras se representan en color rojo los nodos pertenecientes al árbol, y en verde, los que no. Se señalan también las aristas con el menor costo en cada paso.

GRAFO NO DIRIGIDO

Comenzamos aplicando el algoritmo de Prim sobre el primer grafo, que es no dirigido.

Partimos del vértice A:

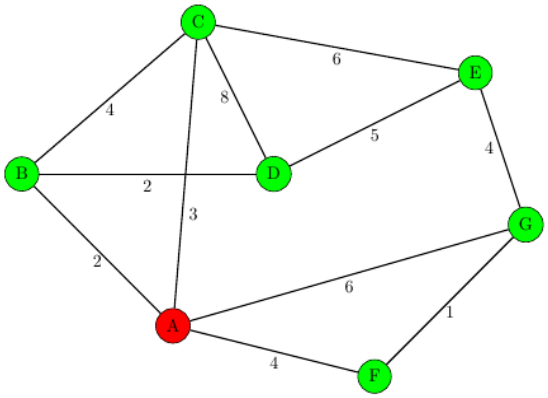


Figura 2.4: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Observamos que el nodo más cercano a A es B, siendo 2 el costo correspondiente.

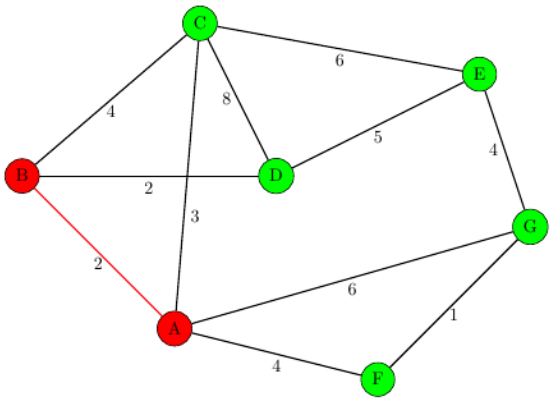


Figura 2.5: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Hacemos $k = 1$. En este ejemplo, $n = 7$. Por tanto, pararemos cuando k sea igual a 6.

El vértice no conectado al árbol más cercano a A o a B, que son los vértices pertenecientes al árbol de momento, es D. Está conectado a B mediante una arista con costo 2.

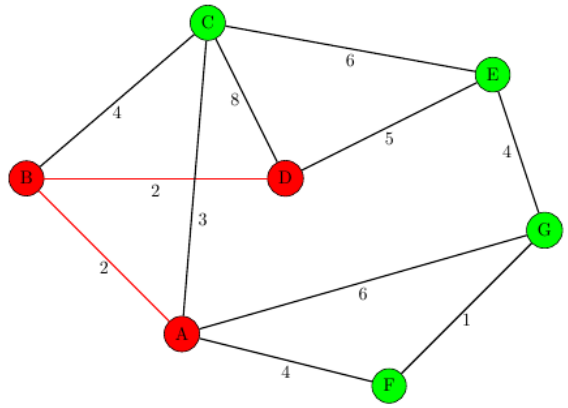


Figura 2.6: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Hacemos $k = 2$. Observamos que el vértice no conectado al árbol más cercano a los nodos A, B o D, es C, que se conecta con A mediante una arista de costo 3.

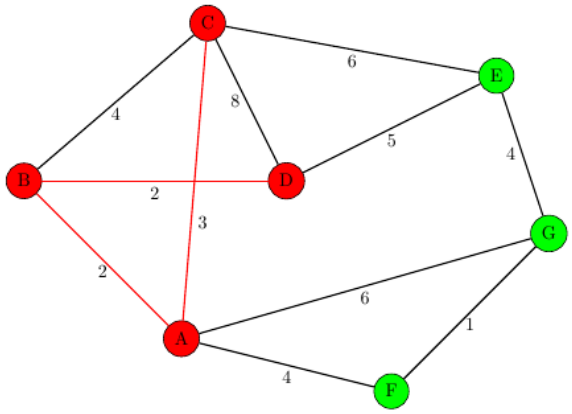


Figura 2.7: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Hacemos $k = 3$. Si nos fijamos, hay dos aristas con el mínimo costo (4) que parte de un nodo incluido ya en el árbol: la que une B y C, y la que une A y F. No obstante, tanto B como C pertenecen ya al árbol, y si tomásemos dicha arista no estaríamos formando un árbol, pues habría un ciclo. Esta es la explicación de por qué debemos tomar en el paso 2 un vértice que no forme parte del árbol que estamos creando. Por tanto, incluimos el nodo F en el árbol:

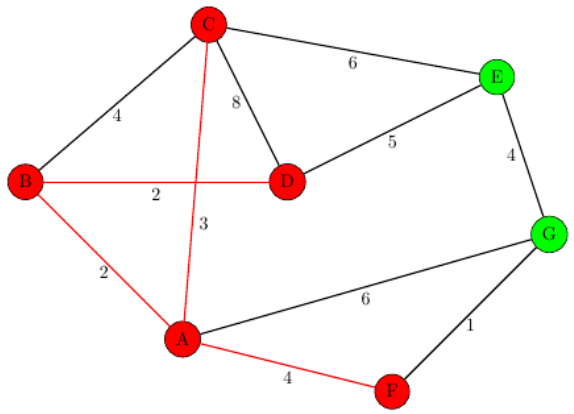


Figura 2.8: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Hacemos $k = 4$. En este caso, G es el nodo más cercano a alguno de los vértices que forman nuestro árbol, pues se conecta con F con costo 1.

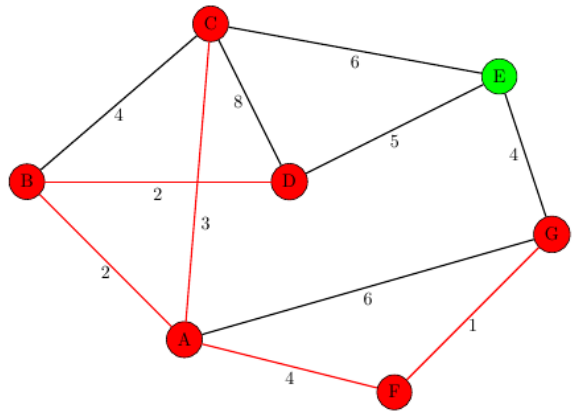


Figura 2.9: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

Hacemos $k = 5$. Únicamente falta por añadir el nodo E al árbol. Vemos que se conecta con G con un costo de 4, que es el mínimo costo de los restantes.

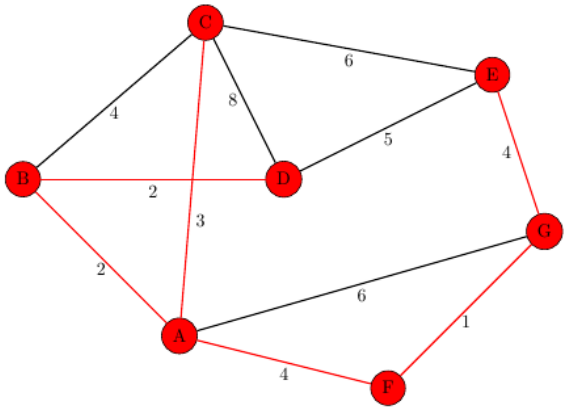


Figura 2.10: Algoritmo de Prim. Grafo no dirigido.
Fuente: Elaboración propia

$k = 6$. Por tanto, STOP (todos los nodos forman parte del árbol)

Resultado final

El árbol de expansión de mínimo costo dado por el algoritmo de Prim sería el siguiente:

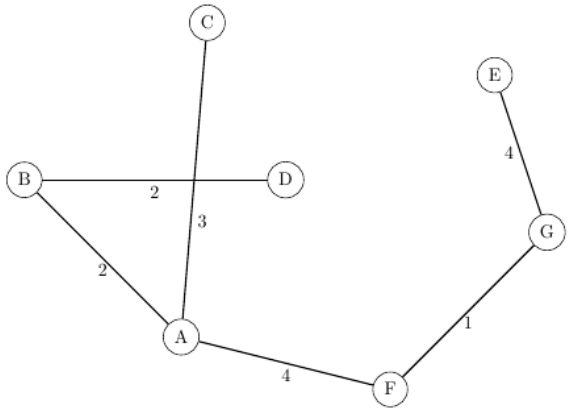


Figura 2.11: Algoritmo de Prim. Grafo no dirigido. Resultado final.
Fuente: Elaboración propia

GRAFO DIRIGIDO

Utilizamos la misma mecánica para calcular el MST del grafo dirigido. Partiendo del vértice A, tendríamos lo siguiente:

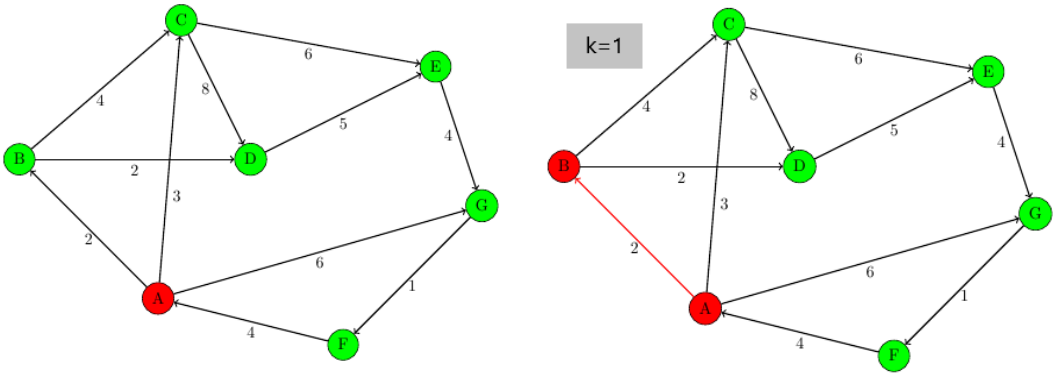


Figura 2.12: Algoritmo de Prim. Grafo dirigido.
Fuente: Elaboración propia

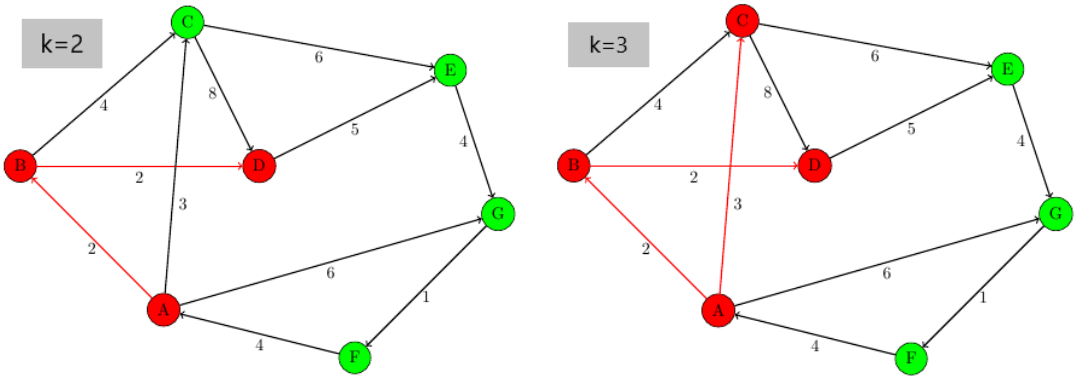


Figura 2.13: Algoritmo de Prim. Grafo dirigido.
Fuente: Elaboración propia

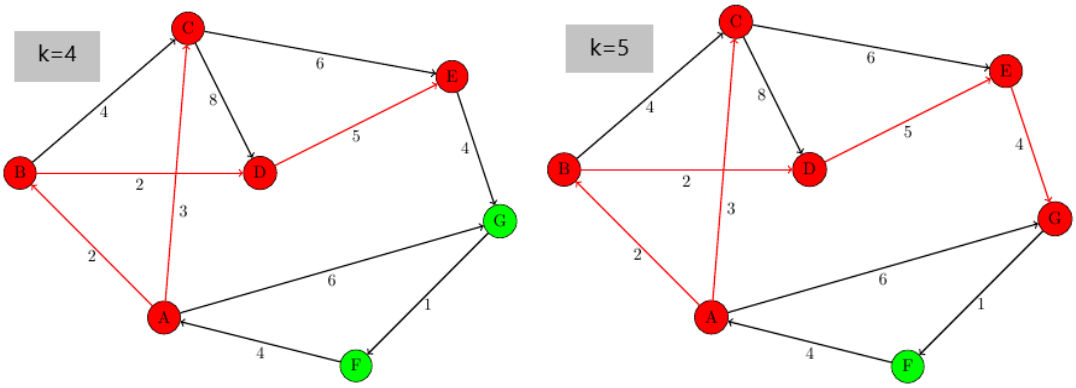


Figura 2.14: Algoritmo de Prim. Grafo dirigido.
Fuente: Elaboración propia

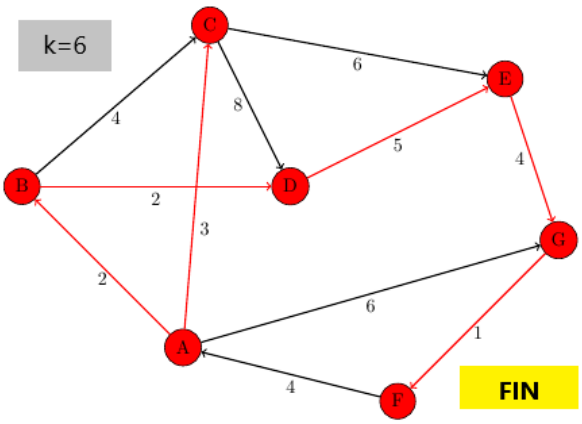


Figura 2.15: Algoritmo de Prim. Grafo dirigido.
Fuente: Elaboración propia

Resultado final

El árbol de expansión de mínimo costo dado por el algoritmo de Prim sería el siguiente:

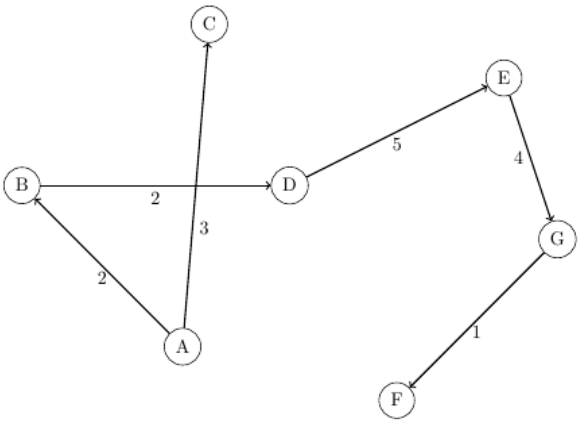


Figura 2.16: Algoritmo de Prim. Grafo dirigido. Resultado final.
Fuente: Elaboración propia

2.4.2. Algoritmo de Kruskal

Este algoritmo, al igual que el de Prim, calcula el MST de un grafo ponderado dado. Como el anterior, si se parte de un grafo no conexo, devuelve el MST de una de sus componentes conexas. Veamos cómo se implementa:

2.4.2.1. Pasos del algoritmo

Suponemos un grafo G con n nodos.

Paso 1

- Se toma cada nodo del grafo como un árbol independiente. Se tienen entonces n árboles diferentes
- Se define E, el conjunto de todos las aristas del grafo

Paso 2

- Si $E \neq \{\}$, se toma la arista de menor costo de E que una dos árboles diferentes sin formar ningún ciclo

Paso 3

- Si se tiene un único árbol, STOP. Se tiene el MST.
- En caso contrario, volver al paso 2.

2.4.2.2. Aplicación

En las siguientes figuras se representan en color rojo los nodos pertenecientes al árbol, y en verde, los que no. Se señalan también las aristas con el menor costo en cada paso.

GRAFO NO DIRIGIDO

Comenzamos aplicando el algoritmo de Kruskal sobre el primer grafo, que es no dirigido.

Se describen con diferentes colores los árboles definidos en el grafo. Como hay 7 nodos, partiremos de 7 árboles independientes:

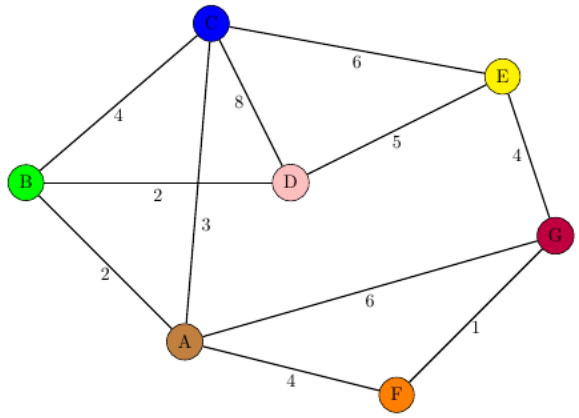


Figura 2.17: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

E está formado por todas las aristas del grafo. Observamos que la arista de menor costo que une dos árboles diferentes (en este caso que une dos vértices cualesquiera) es aquella que conecta a los nodos F y G, con costo 1. Así:

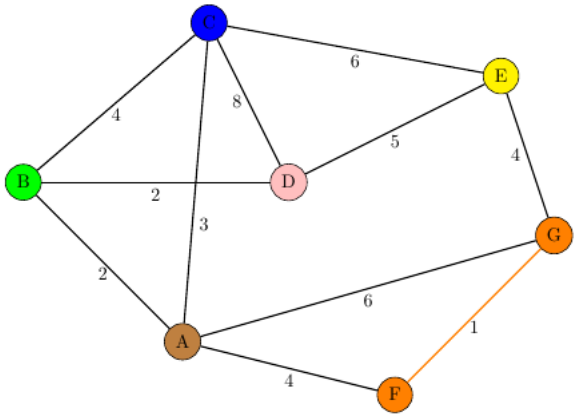


Figura 2.18: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

Podemos unir a continuación los vérices A y B, ó B y D indistintamente, pues ambas aristas tienen peso 2 y estarían uniendo árboles diferentes en cada caso. Nos decantamos por la primera opción:

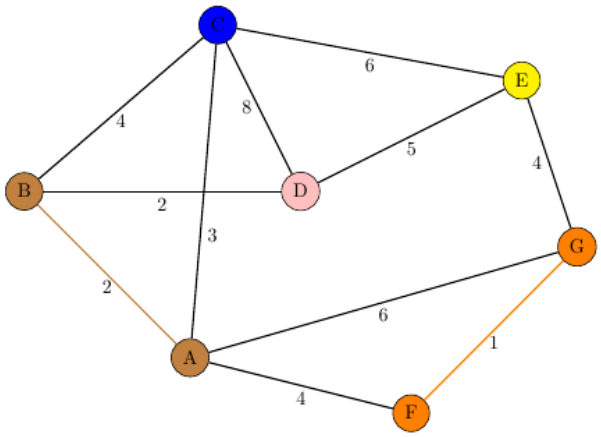


Figura 2.19: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

El siguiente paso sería unir el vértice D al árbol formado anteriormente (aquel formado por los nodos A y B), mediante la arista que une a D con B:

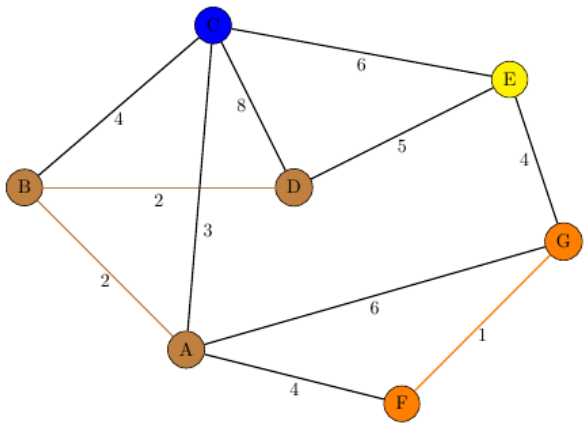


Figura 2.20: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

De las aristas restantes pertenecientes a E, la que tiene menor costo es aquella que conecta a C con el árbol formado por los nodos A, B y D (mediante la arista que une a C con A, que tiene costo 3):

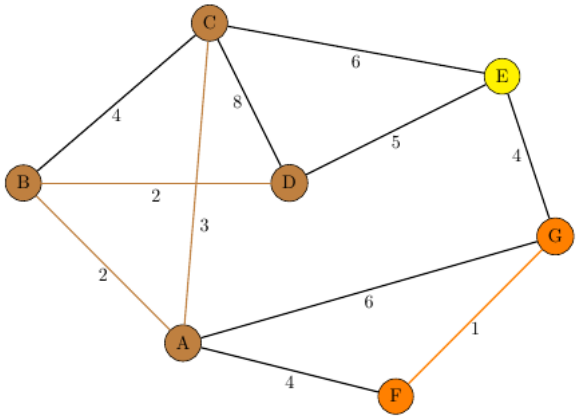


Figura 2.21: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

Nos encontramos en el mismo caso que antes. Podemos comenzar uniendo los nodos E y G ó A y F, pues ambas aristas tienen el costo mínimo de E, que es 4. Claramente el resultado será el mismo. Tomando la primera opción de nuevo:

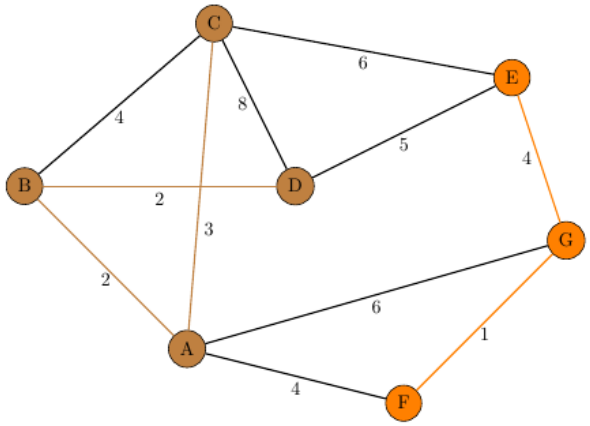


Figura 2.22: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

Unimos los dos árboles restantes mediante la arista que acabamos de mencionar, la que conecta A y F con costo 4.

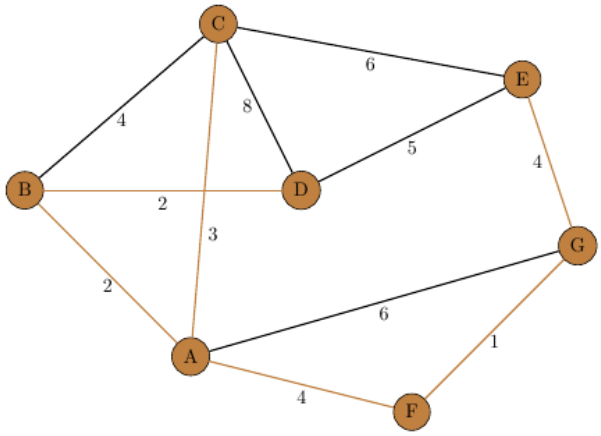


Figura 2.23: Algoritmo de Kruskal. Grafo no dirigido.
Fuente: Elaboración propia

Se tiene ya un árbol único. Por tanto, STOP.

Resultado final

El árbol de expansión de mínimo costo dado por el algoritmo de Kruskal sería el siguiente:

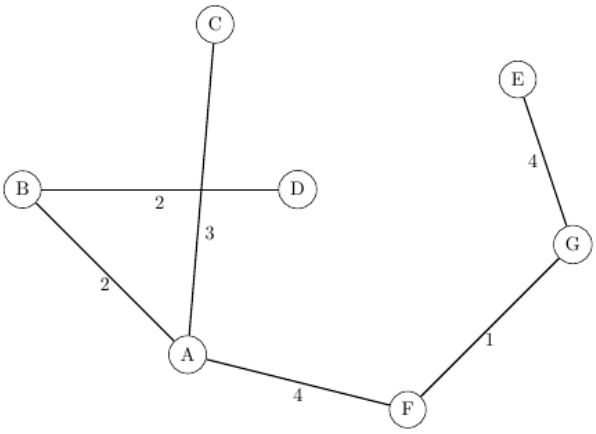


Figura 2.24: Algoritmo de Kruskal. Grafo no dirigido. Resultado final.
Fuente: Elaboración propia

GRAFO DIRIGIDO

Utilizamos la misma mecánica para calcular el MST del grafo dirigido:

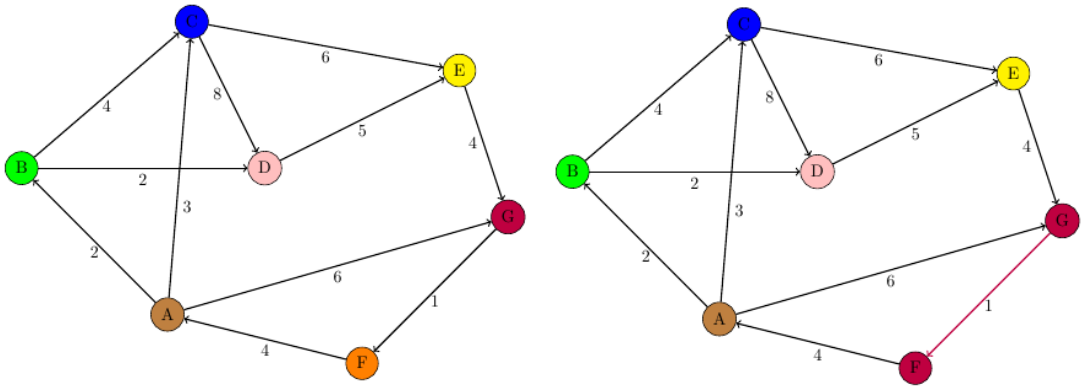


Figura 2.25: Algoritmo de Kruskal. Grafo dirigido.
Fuente: Elaboración propia

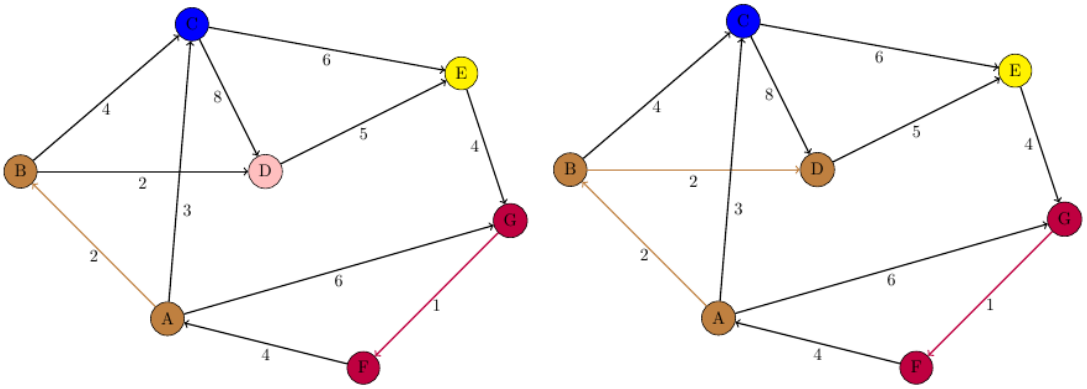


Figura 2.26: Algoritmo de Kruskal. Grafo dirigido.
Fuente: Elaboración propia

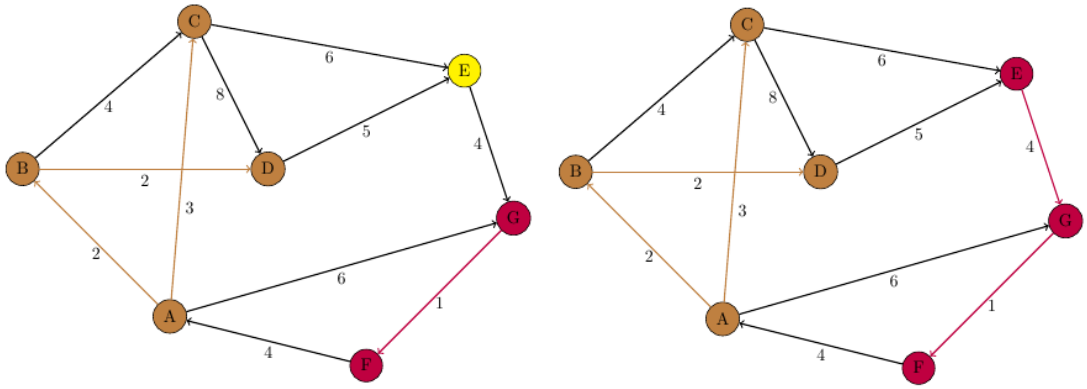


Figura 2.27: Algoritmo de Kruskal. Grafo dirigido.
Fuente: Elaboración propia

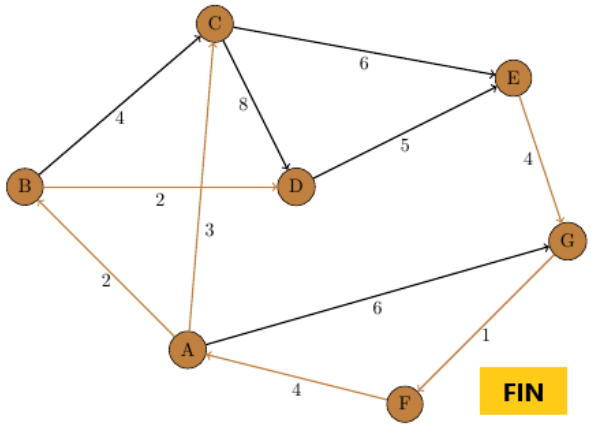


Figura 2.28: Algoritmo de Kruskal. Grafo dirigido.
Fuente: Elaboración propia

Resultado final

El árbol de expansión de mínimo costo dado por el algoritmo de Kruskal sería el siguiente:

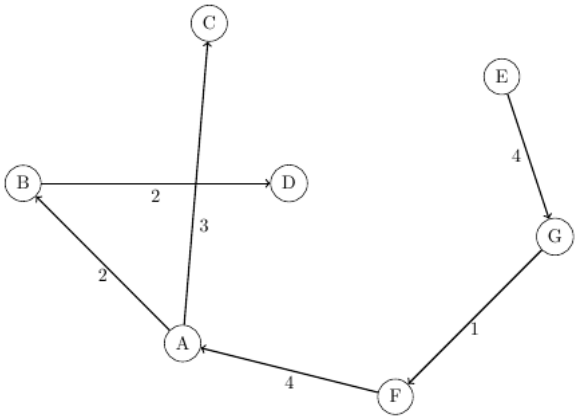


Figura 2.29: Algoritmo de Kruskal. Grafo dirigido. Resultado final.
Fuente: Elaboración propia

Capítulo 3

Flujos y Redes

3.1. Introducción

En esta sección introduciremos las redes de flujo, que son un modelo que permite representar sistemas tales como mapas de carreteras, redes de tuberías o conexiones de red, desde un punto de vista abstracto. Estos sistemas se pueden asimilar a grafos cuyos arcos tienen una capacidad máxima y por los cuales transitan elementos (coches, líquidos, datos, etc.).

Sobre las redes de flujo se pueden plantear numerosos problemas, entre ellos el de la búsqueda del flujo máximo, que consiste en encontrar la máxima cantidad de elementos que se pueden enviar entre dos vértices de la red. Nos centraremos tanto en el planteamiento de este tipo de problema como en su resolución gracias a varios algoritmos.

3.2. Red de flujo

Definición 3.2.1 *Una* ***red de flujo*** *es un grafo dirigido $G = (V, E)$ en el cual cada arco $(u, v) \in E$ tiene una capacidad no negativa $c(u, v) \geq 0$. Se asume que si $(u, v) \notin E$, entonces $c(u, v) = 0$. En toda red de flujo se distinguen dos vértices especiales, la* ***fuente*** *s y el* ***destino*** *o* ***sumidero*** *t.*

Mostraremos a continuación un ejemplo de red de flujo, en el que el valor de cada arco representa su capacidad:

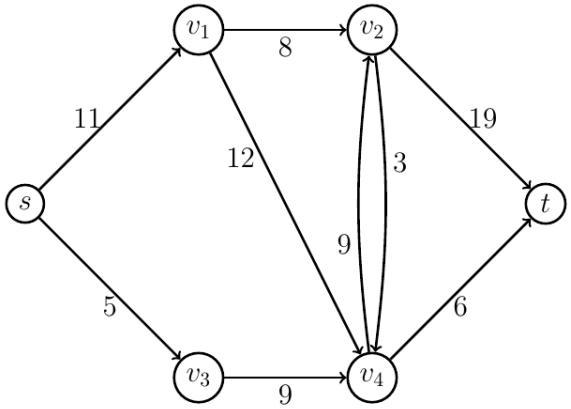


Figura 3.1: Red de flujo (con capacidad).
Fuente: Elaboración propia

Esta red de flujo podría representar, por ejemplo, una red de pozos, en la que se bombea agua hacia un pozo principal, representado por el vértice t. Suponemos que la capacidad de esta red se mide en litros por segundo. De esta forma, podemos decir que la capacidad de la tubería que conecta el pozo fuente (s) hacia el pozo v_1 es de 11 litros por segundo; la capacidad de la tubería que parte desde el pozo v_3 hacia el pozo v_4 es de 9 litros por segundo; la de la tubería que se dirige desde éste último pozo hacia el pozo principal es de 6 litros por segundo...

Dada una determinada red de flujo, un **flujo** para ella es una función $f : V \times V \to \mathbb{R}$ que satisface las siguientes condiciones:

- **Restricción de capacidad**: $\forall u, v \in V : f(u, v) \leq c(u, v)$. La cantidad $f(u, v)$, que puede ser positiva, negativa o nula, es llamada flujo desde el vértice u al vértice v. Esta restricción impone un límite a la red en cuanto a la capacidad de flujo que se puede enviar por cada arco.
- **Simetría**: $\forall u, v \in V : f(u, v) = -f(v, u)$. Se trata de una propiedad añadida para facilitar la notación. Básicamente, nos dice que si un flujo f circula desde el vértice u hasta el vértice v, entonces ese mismo flujo, pero con signo negativo, puede suponerse que circula en sentido inverso (de v a u).
- **Conservación de flujo**: $\forall u \in V - \{s, t\} : \sum_{v \in V} f(u, v) = 0$. Es decir, para cualquier vértice, sin tener en cuenta la fuente y el destino, la suma del flujo saliente de dicho vértice es nula.

Si combinamos las dos últimas propiedades, podemos llegar a la conclusión de que para cualquier vértice, sin contar s y t, se cumple que entra y sale la misma cantidad de flujo. Esto es:

$$\forall u \in V - \{s, t\} : f(u, V) = f(V, u) = 0$$

Además, podemos ya definir el valor del flujo f de la siguiente manera:

$$|f| = \sum_{v \in V} f(s, v),$$

es decir, el valor de un flujo es la cantidad total de flujo que parte desde la fuente.

En la siguiente figura mostraremos la misma red de flujo expuesta anteriormente, pero con un flujo concreto. Cada arco lleva asociado un par de valores que representan, respectivamente, el flujo y la capacidad de dicho arco.

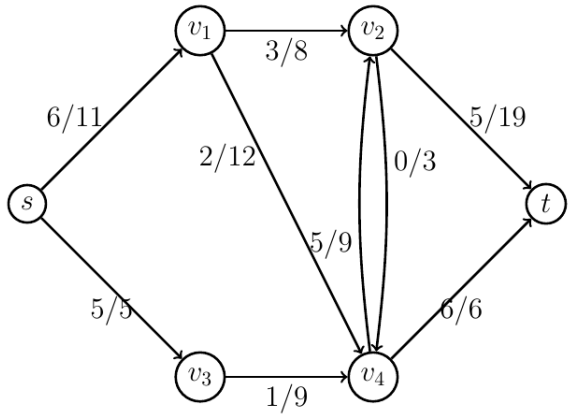


Figura 3.2: Red de flujo (con flujo y capacidad).
Fuente: Elaboración propia

Volviendo al caso anterior, podemos decir que aunque la tubería que conecta el pozo fuente con el primer pozo tiene capacidad de 11 litros por segundo, ésta recibe un flujo de 6 litros por segundo; el tercer pozo, v_3, envía un litro por segundo hacia el pozo v_4, aunque podría enviar hasta 9 litros por segundo; éste último pozo expulsa hacia el pozo de destino todo el agua que puede, esto es, 6 litros por segundo...

3.3. Problema de flujo máximo

Tras esta introducción a las redes de flujos y su aplicación, podríamos preguntarnos: ¿Cuál es la cantidad máxima de flujo que se puede enviar, en una determinada red de flujo, desde el vértice fuente s al vértice destino t? Encontrar la solución a esta cuestión es encontrar la solución del denominado **problema de flujo máximo**.

El problema de maximizar un flujo a lo largo de las aristas de una red de flujo fue estudiado por primera vez por el matemático estadounidense Ted Harris en 1955. Este estudio, que en un principio fue catalogado como información confidencial del Estado, modelizaba la red ferroviaria soviética en el área comprendida entre Moscú, el mar Báltico, Polonia, Rumanía y el mar Negro:

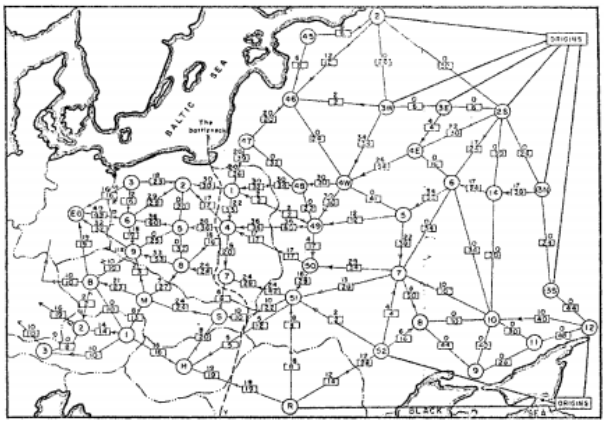


Figura 3.3: Red de ferrocarriles rusos de 1955

El planteamiento del problema era el siguiente:

"Considere una red de ferrocarril conectando dos ciudades a través de un número de ciudades intermedias, donde cada conexión en esta red tiene un número asignado representando su capacidad. En estas condiciones, encuentre un flujo máximo de trenes entre una ciudad y la otra."

El artículo publicado por Harris serviría de inspiración para que los matemáticos L.R. Ford y D.R. Fulkerson profundizaran en el tema, describiendo un algoritmo para hallar ese flujo máximo.

Procedemos a continuación a explicar este algoritmo, así como otros dos, que ayudarán a resolver este tipo de problemas dentro de una red de flujos. Además, los aplicaremos al siguiente ejemplo, en el cual cada arco tiene asignado su capacidad correspondiente:

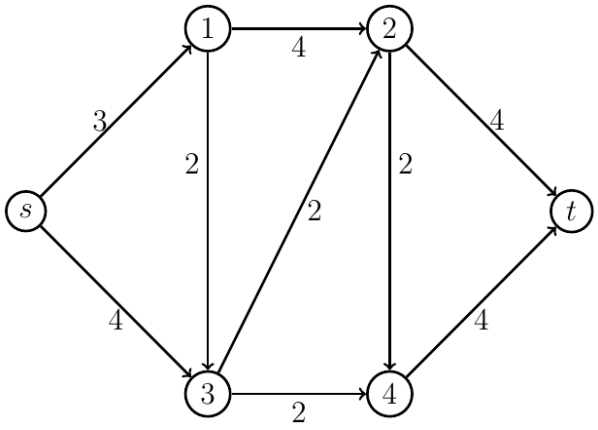


Figura 3.4: Red de flujo para calcular flujo máximo.
Fuente: Elaboración propia

3.3.1. Algoritmo de Ford-Fulkerson

El algoritmo de Ford-Fulkerson es un algoritmo voraz que calcula el flujo máximo de una red de flujo.

La idea detrás del algoritmo es como sigue: mientras que haya un camino desde la fuente al sumidero, con capacidad disponible para todas las aristas del camino, enviaremos flujo a través de uno de los posibles caminos. Luego buscaremos otro camino y repetiremos mientras haya alguno, y así sucesivamente.

En primer lugar, introduciremos una serie de conceptos que nos ayudarán a entender su mecánica.

3.3.1.1. Definiciones previas

Definición 3.3.1 *La **capacidad residual** de un arco con respecto de un flujo f, denotado como c_f, es la diferencia entre las capacidades de los arcos y su flujo. Es decir, $c_f(e) = c(e) - f(e)$. A partir de esto, podemos construir una **red residual**, denotada $G_f(V, E_f)$, la cual modela la cantidad de capacidad disponible del conjunto de arcos en $G = (V, E)$. Formalmente, dada una red de flujo G, la red residual G_f posee el conjunto de nodos V, el conjunto de aristas $E_f = \{e \in V \times V : c_f(e) > 0\}$ y la función de capacidad c_f.*

Nótese que puede haber un camino de u a v en la red residual, aunque no exista un camino de u a v en la red original. Dado que los flujos en direcciones opuestas se cancelan, decrecer el flujo de v a u es lo mismo que incrementar el flujo de u a v.

Definición 3.3.2 *Un **camino incremental** es un camino $(u_1, u_2, ..., u_k)$ en la red residual, donde $u_1 = s$, $u_k = t$ y $c_f(u_i, u_{i+1}) > 0$.*

Una red es de máximo flujo si y sólo si no hay caminos incrementales en la red residual G_f.

A veces, cuando se modela una red con más de una fuente, una superfuente es introducida en el grafo.

Definición 3.3.3 *Una **superfuente** consiste en un vértice conectado a cada una de las fuentes por una arista con infinita capacidad, actuando así como una fuente global. Una construcción similar se puede hacer para los sumideros, creando un **supersumidero**.*

3.3.1.2. Pasos del algoritmo

Consideramos $G = (V, E)$ el grafo de n nodos, donde el nodo 1 es el inicial y el nodo n es el final. Cada arista (i, j) tiene una capacidad máxima de flujo, c_{ij}

Paso 1

- Considerar el grafo $G' = (V, E)$ con el mismo número de nodos y aristas que G, pero $cij = 0$ en G'

Paso 2

- Buscar un camino C en G que empiece en el nodo 1 y termine en el nodo n, pasando por las aristas que tengan $c_{ij} \neq 0$
- Definir $\Delta_{ij} = \text{mín}_{i,j \in C}\, c_{ij}$
- Por el camino C pueden fluir hasta Δ_{ij} unidades de flujo, como máximo

Paso 3

- Para cada arista (i, j) de C, añadir c_{ij} al coste de cada arista de G'
- Disminuir el de cada arista de G correspondiente

Paso 4

- Si aún quedan caminos sin recorrer en G del nodo 1 al n, volver al Paso 2
- En otro caso, el flujo es máximo. FIN

Respecto a estos pasos, nótese que el algoritmo consiste en imponer primero un flujo inicial nulo, para después crear un bucle que encuentre un camino incremental en la red residual de manera repetida, el cual actualiza el flujo convenientemente. Cuando no existen más caminos, el flujo es máximo y el algoritmo ha finalizado.

3.3.1.3. Aplicación

Llamamos G al grafo dado en la Figura 3.4, en el que renombramos los nodos como 1, 2, 3, 4, 5 y 6, siendo 1 la fuente y 6 el destino. A partir de este, creamos el grafo G' con las características explicadas en el Paso 1:

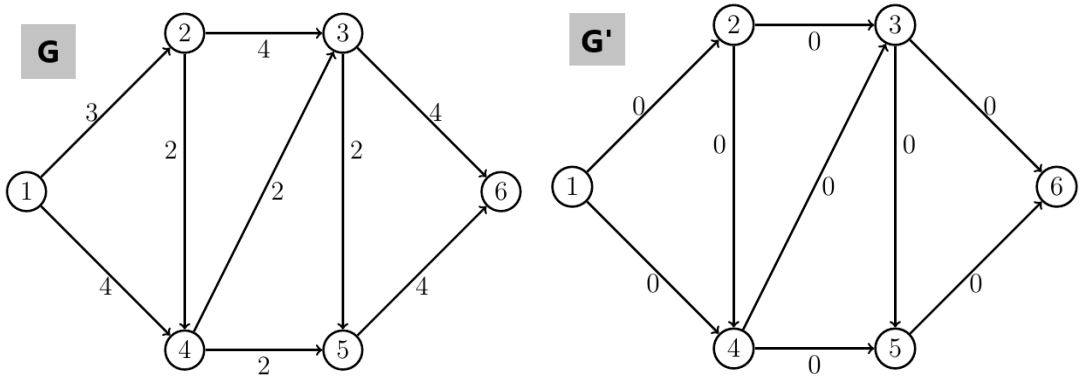


Figura 3.5: Algoritmo de Ford-Fulkerson.
Fuente: Elaboración propia

Empezamos tomando el camino incremental $C = \{1, 2, 3, 6\}$. Como vemos, $\Delta = \text{mín}\{3, 4, 4\} = 3$. Es decir, por este camino pueden fluir 3 unidades de flujo como máximo. Sumamos este coste al coste de las aristas que conectan los nodos 1, 2, 3 y 6 de G' (en verde) y lo restamos al coste de las mismas aristas de G (en rojo).

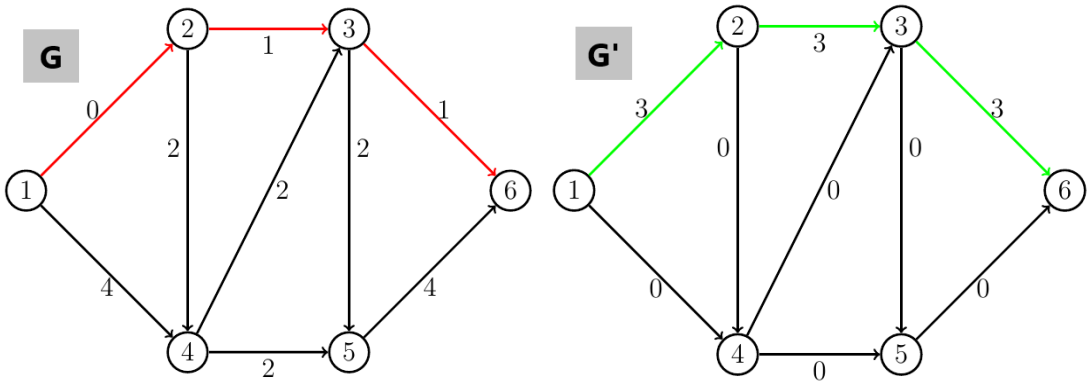


Figura 3.6: Algoritmo de Ford-Fulkerson.
Fuente: Elaboración propia

Tomamos ahora el siguiente camino incremental: $C = \{1, 4, 3, 5, 6\}$. Por él pueden fluir como mucho 2 unidades de flujo, puesto que $\Delta = \text{mín}\{4, 2, 2, 4\} = 2$. Sumamos 2 al peso de las aristas $(1, 4)$, $(4, 3)$, $(3, 5)$ y $(5, 6)$ en G' y restamos dos al peso de las mismas en G.

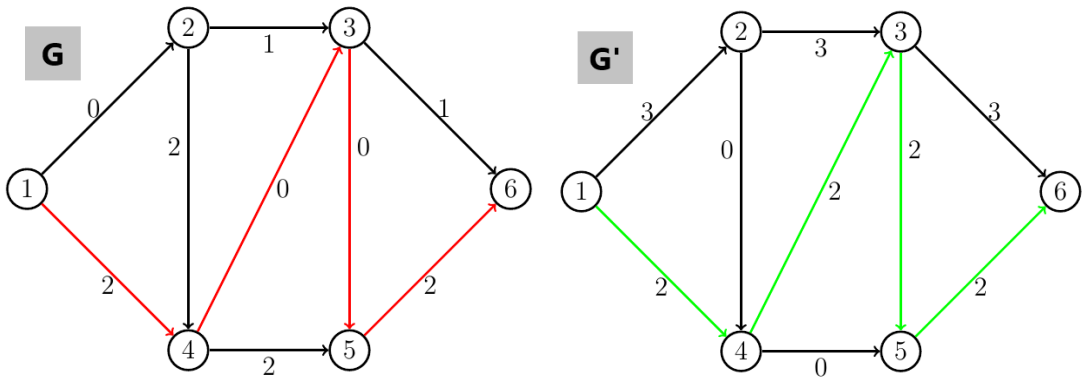


Figura 3.7: Algoritmo de Ford-Fulkerson.
Fuente: Elaboración propia

Observamos que solo nos queda un camino por tomar en el que ningún c_{ij} sea nulo: aquel formado por los vértices $\{1, 4, 5, 6\}$. Como $\Delta = \text{mín}\{2, 2, 2\} = 2$, por el camino incremental formado por esos vértices pueden fluir, a lo sumo, 2 unidades de flujo. De nuevo, sumamos 2 al peso de las aristas $(1, 4)$, $(4, 5)$ y $(5, 6)$ en G' y restamos dos al peso de las mismas en G.

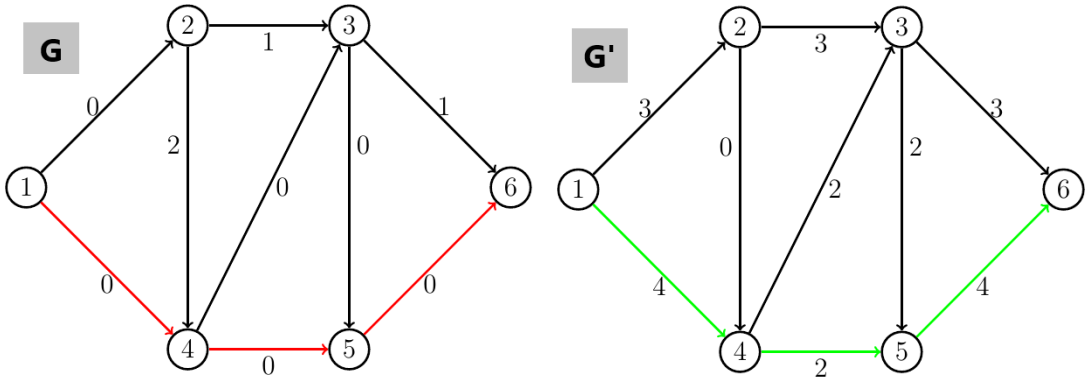


Figura 3.8: Algoritmo de Ford-Fulkerson.
Fuente: Elaboración propia

Por tanto, el problema ha terminado, y el flujo máximo viene dado por la suma de cada Δ calculado, o bien por la suma de los costes correspondientes a las aristas que llegan al vértice 6.

En este caso, el flujo máximo es 7; o lo que es lo mismo, la cantidad máxima de flujo que se puede enviar en esta red de flujo desde la fuente hasta el destino es 7.

3.3.1.4. Posible generalización: redes con múltiples fuentes y sumideros

En la práctica es habitual encontrar problemas de flujo máximo caracterizados por poseer varias fuentes y/o sumideros. Ante esta "complicación", podemos reducir el problema a un problema de flujo máximo en una red ordinaria, como la vista anteriormente.

Para ello, se añaden una superfuente s y una arista (s, s_i) con capacidad $c(s, s_i) = \infty$ por cada fuente $(i = 1, ..., m)$. De forma análoga, se crea un supersumidero t con aristas (t_i, t) con capacidad $c(t_i, t) = \infty$ para los sumideros $(i = 1, ..., n)$. La fuente s provee a cada

una de las fuentes s_i de tanto flujo como necesiten, y el sumidero t actúa de igual forma: provee a cada sumidero t_i de tanto flujo como necesiten. Mediante esta transformación, el flujo máximo transportado por la nueva red equivale al flujo máximo que se puede transportar en la red original, por lo que hemos logrado simplificar este tipo de problema "más raro" a uno ya conocido.

3.3.2. Algoritmo de Edmonds-Karp

Al resolver el problema aplicando el algoritmo de Ford-Fulkerson, podemos observar que no se sigue ningún criterio para encontrar los respectivos caminos incrementales. Esta es la única diferencia entre el algoritmo de Ford-Fulkerson y el de Edmonds-Karp: el camino encontrado por este último algoritmo debe ser el más corto posible que tenga capacidad disponible. Para ello, aplicaremos el algoritmo de búsqueda en anchura, el cual visita un nodo inicial y luego a los nodos que están a una arista de distancia del mismo; a continuación, a los nodos que están a dos aristas de distancia del nodo inicial y así, sucesivamente, hasta alcanzar a todos los nodos desde los que pueda llegar el nodo inicial.

3.3.2.1. Pasos del algoritmo

Consideramos $G = (V, E)$ el grafo de n nodos, donde el nodo 1 es el inicial y el nodo n es el final. Cada arista (i, j) tiene una capacidad máxima de flujo, c_{ij}

Paso 1

- Considerar el grafo $G' = (V, E)$ con el mismo número de nodos y aristas que G, pero $cij = 0$ en G'

Paso 2

- Hallar un camino C en G que empiece en el nodo 1 y termine en el nodo n, pasando por las aristas que tengan $c_{ij} \neq 0$, **aplicando el algoritmo de búsqueda en anchura**
- Definir $\Delta_{ij} = \text{mín}_{i,j \in C}\, c_{ij}$
- Por el camino C pueden fluir hasta Δ_{ij} unidades de flujo, como máximo

Paso 3

- Para cada arista (i, j) de C, añadir c_{ij} al coste de cada arista de G'
- Disminuir el de cada arista de G correspondiente

Paso 4

- Si aún quedan caminos sin recorrer en G del nodo 1 al n, volver al Paso 2
- En otro caso, el flujo es máximo. FIN

3.3.2.2. Aplicación

Definimos G y G' de la misma forma que en el algoritmo de Ford-Fulkerson:

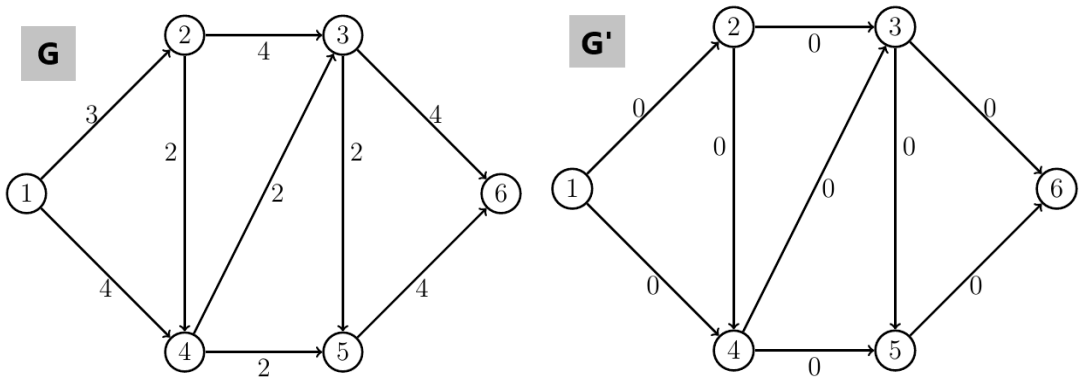


Figura 3.9: Algoritmo de Edmodns-Karp.
Fuente: Elaboración propia

Para hallar los caminos incrementales aplicamos el algoritmo de búsqueda en anchura tomando como nodo inicial, obviamente, el nodo 1. La forma más clara de ver la mecánica de este algortimo es crear un árbol, como los vistos en el Capítulo 2. Así:

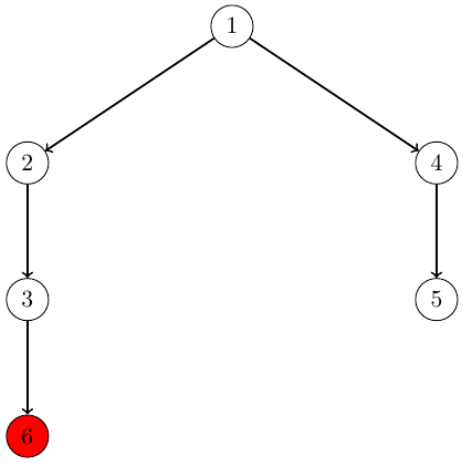


Figura 3.10: Algoritmo de búsqueda en anchura.
Fuente: Elaboración propia

Los nodos que se encuentran a una arista de distancia de éste son los nodos 2 y 4. Tomamos de referencia el nodo 2: tenemos que tanto el nodo 3 como el nodo 4 están a dos aristas de distancia respecto al nodo inicial 1. No obstante, como el nodo 4 ya ha sido tomado anteriormente no lo tenemos en cuenta. De igual forma, el nodo 5 es el que se sitúa a dos aristas de distancia respecto a 1 tomando el nodo 4 (también 3 se encontraría en esta situación, pero ya ha sido tomado antes). Por último, el nodo que queda, 6, se sitúa a tres aristas de distancia de 1 mediante el nodo 3.

Obtenemos así el primer camino incremental disponible: $C = \{1, 2, 3, 6\}$, con $\Delta = \text{mín}\{3, 4, 4\} = 3$. De esta forma, utilizando la misma mecánica que en el algoritmo de Ford-Fulkerson:

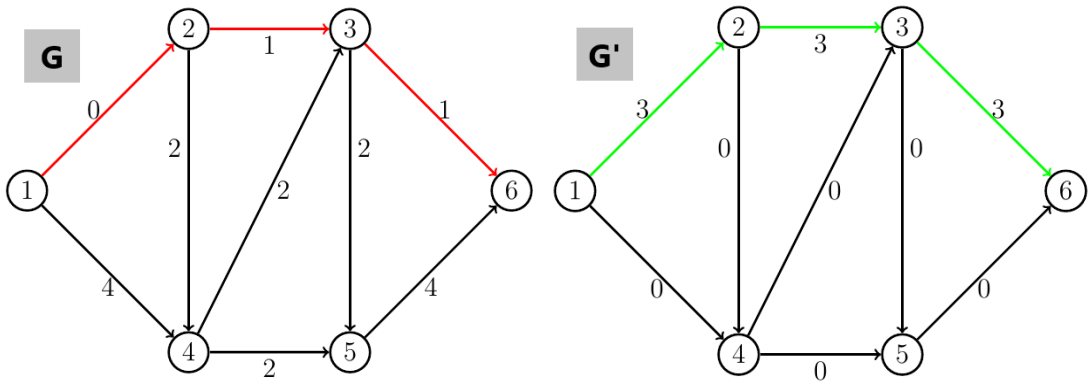


Figura 3.11: Algoritmo de Edmonds-Karp.
Fuente: Elaboración propia

Buscamos a continuación el segundo camino incremental. Volvemos a crear un árbol como el anterior, pero uniremos aquellos nodos conectados por aristas con coste distinto de cero:

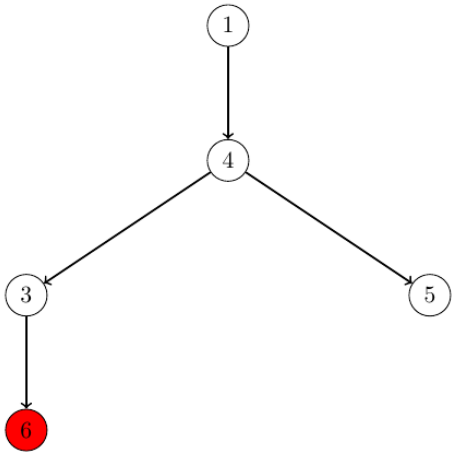


Figura 3.12: Algoritmo de búsqueda en anchura.
Fuente: Elaboración propia

Al ser $c_{12} = 0$, no podemos volver a tomar el nodo 2 de nuevo para encontrar un camino. De esta forma, como muestra el árbol, el segundo camino incremental viene dado por $C = \{1, 4, 3, 6\}$, con $\Delta = \text{mín}\{4, 2, 1\} = 1$.

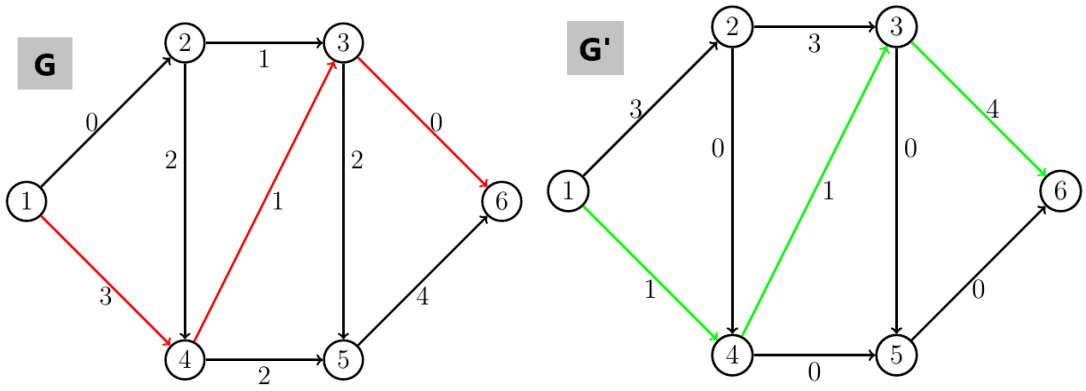


Figura 3.13: Algoritmo de Edmonds-Karp.
Fuente: Elaboración propia

Creamos de nuevo el árbol con los nodos disponibles (aquellos conectados por aristas cuyo coste es distinto a 0):

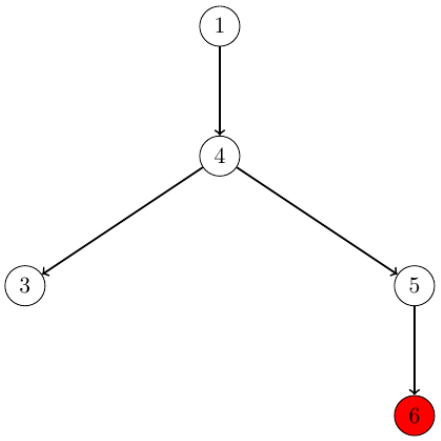


Figura 3.14: Algoritmo de búsqueda en anchura.
Fuente: Elaboración propia

Observamos que el tercer camino incremental viene dado por $C = \{1, 4, 5, 6\}$, con $\Delta = \text{mín}\{3, 2, 4\} = 2$.

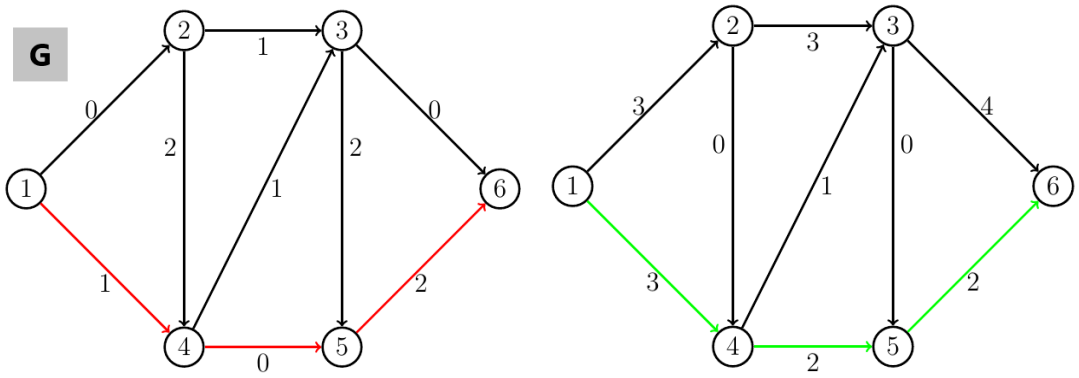


Figura 3.15: Algoritmo de Edmonds-Karp.
Fuente: Elaboración propia

Formando el árbol de nuevo,

Figura 3.16: Algoritmo de búsqueda en anchura.
Fuente: Elaboración propia

creamos el tercer camino de incremental: $C = \{1, 4, 3, 5, 6\}$, con $\Delta = \text{mín}\{1, 1, 2, 2\} = 1$.

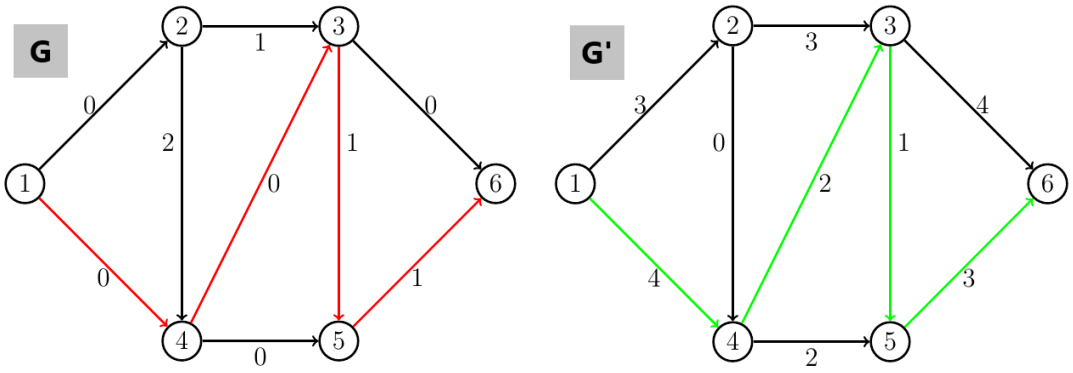


Figura 3.17: Algoritmo de Edmonds-Karp.
Fuente: Elaboración propia

Ya no quedan más caminos por recorrer desde 1 hasta 6 Por tanto, el problema ha terminado, y el flujo máximo viene dado por 7, al igual que antes.

3.3.3. Algoritmo de Dinic

El algoritmo de Dinic es similar al algoritmo de Edmonds-Karp en lo que respecta al uso de los caminos incrementales más cortos.

Para este algoritmo se usaremos muchas de las definiciones vistas anteriormente, aunque debemos hacer uso de otras nuevas que plantearemos a continuación.

3.3.3.1. Definiciones previas

Definición 3.3.4 *El* ***grafo de nivel*** *del grafo residual G_f es el grafo $G_L = ((V, E_L), c_f|_{E_L}, s, t)$, donde $E_L = \{(u, v) \in E_f : dist(v) = dist(u) + 1\}$ y donde, a su vez, $dist(v)$ es la distancia del camino incremental más corto de v a s en G_f y $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$.*

Definición 3.3.5 *Un* ***flujo bloqueante*** *es un flujo desde s a t de longitud f tal que el grafo $G' = ((V, E'_L), s, t)$, donde $E'_L = \{(u, v) : f(u, v) < c_f|_{E_L}(u, v)\}$ no contiene caminos de s a t.*

3.3.3.2. Pasos del algoritmo

Consideramos $G = (V, E)$ el grafo de n nodos, donde el nodo 1 es el inicial y el nodo n es el final. Cada arista (i, j) tiene una capacidad máxima de flujo, c_{ij}

Paso 1

- Establecer $f(e) = 0, \forall e \in E$

Paso 2

- Contruir el grafo de nivel G_L a partir del grafo residual G_f, el cual se obtiene de G
- Si $dist(t) = +\infty$, parar y devolver f

Paso 3

- Encontrar un flujo bloqueante de longitud f' en G_L

Paso 4

- Sumar la cantidad f' al flujo f

3.3.3.3. Aplicación

Llamamos G al grafo dado en la Figura 3.4, en el que renombramos los nodos como 1, 2, 3, 4, 5 y 6, siendo 1 la fuente y 6 el destino. Como se indica en el Paso 1, todos los flujos que circulan en esta red deben tomar el valor 0. Al lado, mostramos el grafo residual G_f, así como el grafo de nivel G_L, en el que los vértices indican los valores $dist(v)$:

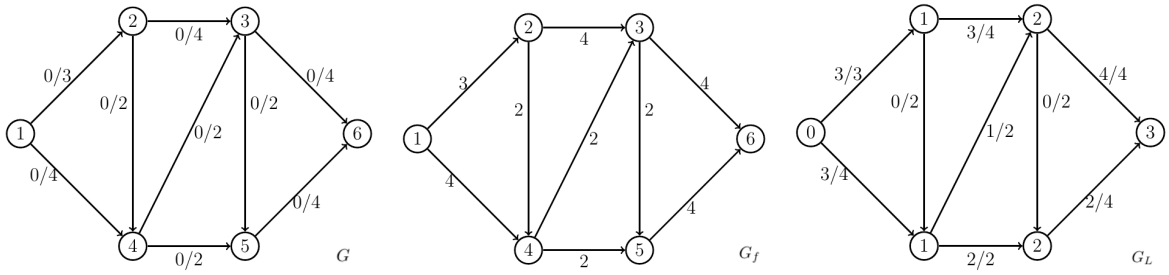


Figura 3.18: Algoritmo de Dinic.
Fuente: Elaboración propia

Observamos que el grafo de nivel muestra en sus nodos los niveles del grafo original tras haber calculado el bloqueo de flujo.

Nótese que el bloqueo de flujo está constituido por

- $\{1, 2, 3, 6\}$ con 3 unidades de flujo
- $\{1, 4, 3, 6\}$ con 1 unidad de flujo
- $\{1, 4, 5, 6\}$ con 2 unidades de flujo

Por lo tanto, el bloqueo del flujo es de 6 unidades y el valor del flujo $|f|$ es 6. Observamos que el algoritmo solo toma en este ejemplo estos 3 bloqueos de flujos, pues según la definición de nivel, se debe ir a un niel $u + 1$.

Repetimos el proceso con el grafo resultante:

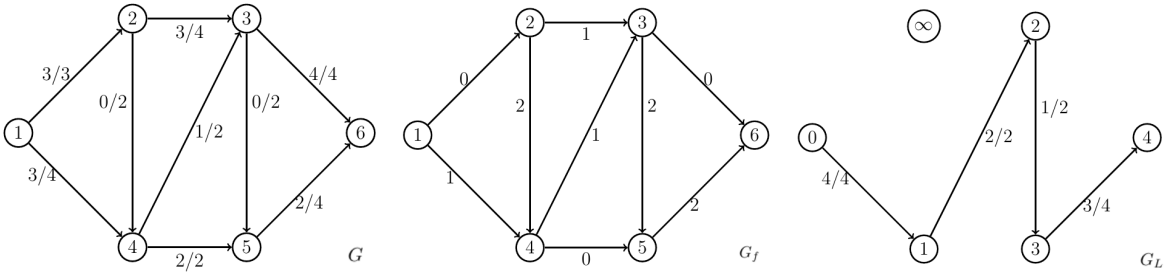


Figura 3.19: Algoritmo de Dinic.
Fuente: Elaboración propia

En este caso, el bloqueo está constituido por

- $\{1, 4, 3, 5, 6\}$ con 1 unidad de flujo

Por lo tanto, el bloque de flujo es de una unidad y el valor del flujo $|f|$ es de 6+1=7.

Repetimos de nuevo el proceso:

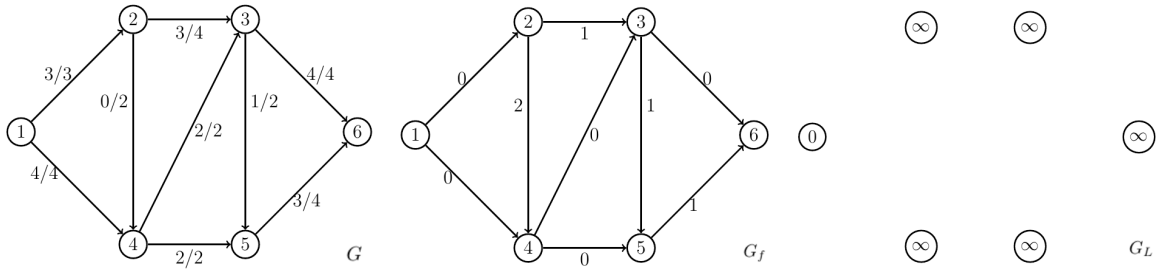


Figura 3.20: Algoritmo de Dinic.
Fuente: Elaboración propia

Observamos que desde el nodo inicial no se puede conseguir un camino hasta el nodo de destino en G_f. De esta manera, el algoritmo termina, aportando un flujo máximo de 7.

Conclusiones

Este trabajo ha sido elaborado con el fin de introducir al lector en el planteamiento y resolución de problemas que podemos describir como Problemas de Redes y Flujos.

Se pretende que las situaciones y ejemplos incluidos en el proyecto permitan entender mejor no sólo la manera de plantearlos y las conclusiones finales que se han obtenido, sino también el funcionamiento de los algoritmos explicados.

Como para entender "Redes y Flujos" es necesario un conocimiento previo de "Teoría de Grafos" y "Árboles", he visto oportuno explicar y aplicar algoritmos para resolver los problemas más importantes en estas ramas.

Bibliografía

M. E. Abajo Casado. Grafos con tamaño máximo y cintura inferiormente acotada. 2009. URL https://idus.us.es/bitstream/handle/11441/24322/2009abajografo.pdf?sequence=1&isAllowed=y.

J. ADIEGO RODRIGUEZ and N. ZIVIANI. *Diseño de algoritmos con implementaciones en Pascal y C*. Editorial Paraninfo, 2007.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. 1988.

M. F. Alvarez Nuñez. Teoría de grafos. 2013. URL http://repobib.ubiobio.cl/jspui/bitstream/123456789/1953/3/Alvarez_Nunez_Marcelino.pdf.

R. Balakrishnan and K. Ranganathan. *A textbook of graph theory*. Springer Science & Business Media, 2012.

J. Barelles Menes. Algoritmos para la resolución de problemas en redes. 2017. URL http://repositori.uji.es/xmlui/bitstream/handle/10234/173687/TFG_2017_BarellesMenes_Jorge.pdf?sequence=1.

L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

L. R. Ford and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian journal of Mathematics*, 9: 210–218, 1957.

A. Gibbons. *Algorithmic graph theory*. Cambridge university press, 1985.

J. L. Gross and J. Yellen. *Graph theory and its applications*. CRC press, 2005.

D. Jungnickel and D. Jungnickel. *Graphs, networks and algorithms*. Springer, 2005. URL https://link.springer.com/content/pdf/10.1007/978-3-540-72780-4.pdf.