TEORÍA DE GRAFOS

Pedro Miranda Menéndez

Resumen

En este tema estudiaremos una pequeña introducción de lo que se conoce como Teoría de Grafos. Se da la definición de grafo orientado y no orientado y los conceptos básicos relacionados con ellos; también se trata la representación matricial de un grafo y la resolución de ciclos eulerianos. Se pasa a continuación al estudio de árboles y sus propiedades. Se tratan algunos problemas en redes no dirigidas; en concreto, se estudia el problema de obtener el árbol de unión de valor mínimo. Luego se pasa a estudiar problemas en redes dirigidas, en concreto el camino de menor valor. Finalmente, se estudia el problema de flujo máximo en redes capacitadas.

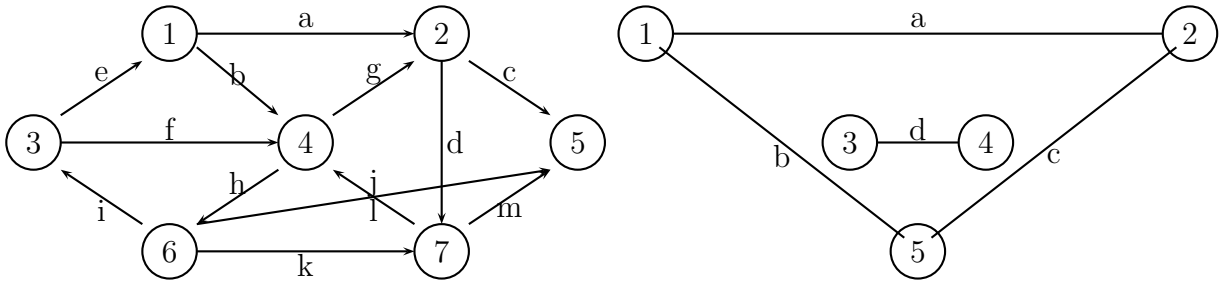
1. Grafos. Conceptos básicos

La Teoria de Grafos aparece con Euler y el conocido problema de los puentes de Königsberg. Básicamente, un grafo es un conjunto de puntos o vértices con conexiones entre ellos; estas conexiones pueden ser en un sentido o de doble sentido. Veamos la definición matemática de grafo.

1.1. Definición de grafo

Un **grafo no orientado** G es un par $G = \{V, E\}$ donde V es un conjunto de *vértices* y E es un conjunto de pares no ordenados $\{i, j\}$ con $i, j \in V$, $i \neq j$ llamados *aristas*.

Un **grafo orientado** G es un par $G = \{V, A\}$ donde V es un conjunto de *vértices* y A es un conjunto de pares ordenados (i, j) con $i, j \in V$, llamados *arcos*; el vértice i se llama *vértice inicial* y el vértice j se llama *vértice final*. Un arco del tipo (i, i) se llama un *bucle*.



Grafo orientado Grafo no orientado

Figura 1: Ejemplos de grafo orientado y no orientado.

Si en un grafo hay aristas o arcos repetidos se habla de *multigrafo*; la *multiplicidad* de un grafo es el máximo número de veces que está repetida una arista o un arco. Un

grafo se dice **simple** si tiene multiplicidad 1 y no contiene bucles (en el caso de grafos orientados). En lo que sigue supondremos siempre que tratamos con grafos simples.

En un grafo orientado, se define el conjunto de **sucesores** de un vértice i como

$$\Gamma^+(i) := \{j : (i,j) \in A\}.$$

Se define el conjunto de **predecesores** de un vértice i como

$$\Gamma^-(i) := \{j : (j,i) \in A\}.$$

Definimos el conjunto de **adyacentes** de i como

$$\Gamma(i) := \Gamma^+(i) \cup \Gamma^-(i).$$

En el caso de grafos no orientados sólo tiene sentido el concepto de **adyacente**, que se define por

$$\Gamma(i) := \{j : \{i,j\} \in E\}.$$

Para un grafo orientado simple, definimos el **semigrado exterior** de un vértice i, denotado $d^+(i)$ como el número de arcos en los que i es el vértice inicial, i.e. $d^+(i) = |\Gamma^+(i)|$ y se define el **semigrado interior** de un vértice i, denotado $d^-(i)$ como el número de arcos en los que i es el vértice final, es decir $d^-(i) = |\Gamma^-(i)|$. El **grado** del vértice i es

$$d(i) = d^+(i) + d^-(i).$$

Dado un grafo no orientado, se define el **grado** de un vértice i, denotado $d(i)$, como el número de aristas en las que aparece el vértice i.

Ejemplo 1. *Para los ejemplos de la Figura 1 se tiene lo siguiente:*
En el caso del grafo orientado, $\Gamma^+(3) = \{1,4\}$, $d^+(3) = 2$, $\Gamma^-(3) = \{6\}$, $d^-(3) = 1$.
En el caso del grafo no orientado, $\Gamma(1) = \{2,5\}$, $d(1) = 2$.

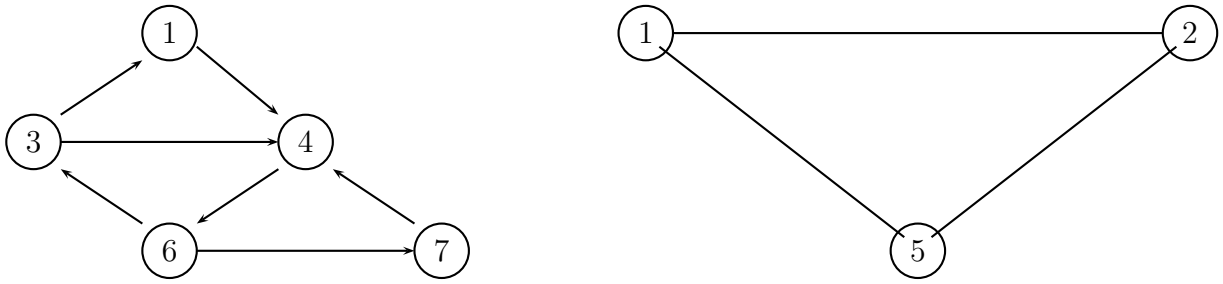
Dado un grafo (no) orientado, se define un **subgrafo** como un par $(V', A_{V'})$ (o $(V', E_{V'})$) donde $V' \subseteq V$ y

$$A_{V'} := \{(i,j) : i,j \in V'\} \quad (E_{V'} := \{\{i,j\} : i,j \in V'\}).$$

Dado un grafo (no) orientado, se define un **grafo parcial** como un par (V, A') (o (V, E')) donde $A' \subseteq A$ (o $E' \subseteq E$).

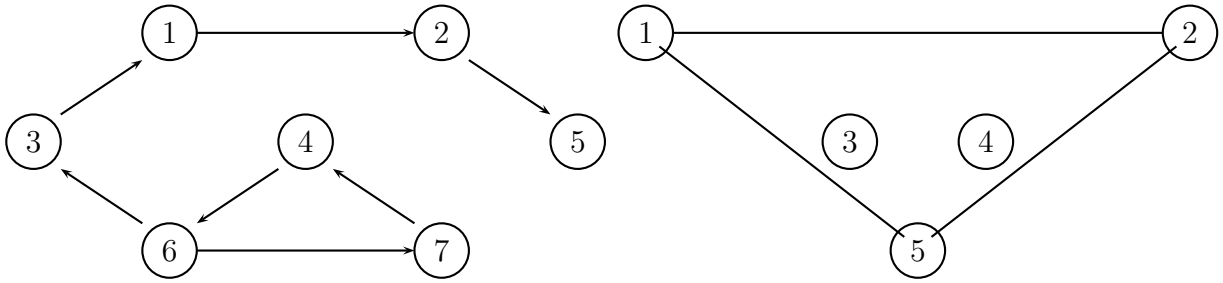
Ejemplo 2. *Para los ejemplos de la Figura 1, tenemos por ejemplo los subgrafos de la Figura 2, con* $V' = \{1,3,4,6,7\}$ *y* $V' = \{1,2,5\}$ *respectivamente.*
Y tenemos los grafos parciales de la Figura 3, con A' *y* E' *dados por*

$$A' = \{(3,1),(1,2),(2,5),(6,3),(7,4),(6,7)\},\ E' = \{\{1,5\},\{1,2\},\{2,5\}\}.$$



Grafo orientado Grafo no orientado

Figura 2: Ejemplos de subgrafos.



Grafo orientado Grafo no orientado

Figura 3: Ejemplos de grafos parciales.

1.2. Conectividad

Consideremos un grafo no orientado. Dada una sucesión de aristas $e_1, ..., e_r$ diremos que determinan una **cadena** si para cualquier e_i, uno de sus vértices es común con e_{i-1} y el otro con $e_{i+1}, i = 2, ..., r-1$.

Una cadena representa una posibilidad de comunicación entre el vértice libre de e_1 y el vértice libre de e_r (y viceversa). Si estos dos vértices son el mismo, entonces tenemos un **ciclo**.

Ejemplo 3. *En el ejemplo de la Figura 1 tenemos la cadena* $\{1,5\},\{5,2\}$*, que se suele poner como* $1-5-2$*. La cadena* $1-5-2-1$ *es un ciclo.*

Consideremos un grafo orientado. Dada una sucesión de arcos $a_1, ..., a_r$ diremos que determinan un **camino** si para cualquier $a_i = (j,k)$, se tiene que j es el vértice final de a_{i-1} y k es el vértice inicial de $a_{i+1}, i = 2, ..., r-1$.

Un camino representa una posibilidad de comunicación entre el vértice libre de a_1 y el vértice libre de a_r (pero no al revés). Si estos dos vértices son el mismo, entonces tenemos un **circuito**.

Ejemplo 4. *En el ejemplo de la Figura 1 tenemos el camino* $(3,1),(1,4),(4,6)$*, que se suele poner como* $3 \to 1 \to 4 \to 6$*. El camino* $3 \to 1 \to 4 \to 6 \to 3$ *es un circuito.*

Un grafo no orientado se dice **conexo** si dados dos vértices distintos, existe al menos una cadena que los une. Si el grafo no es conexo, podemos entonces definir sobre el conjunto de vértices la relación $i\mathcal{R}j$ si existe al menos una cadena entre i y j ó $i = j$. Esta relación es de equivalencia; sus clases de equivalencia determinan una partición en el conjunto de vértices y se llaman **componentes conexas**. La clase de equivalencia de un vértice es por tanto el conjunto de vértices que están comunicados con él.

Un grafo orientado se dice **fuertemente conexo** si dados dos vértices distintos i, j, existe al menos un camino entre i y j y al menos un camino entre j y i. Si el grafo no es fuertemente conexo, podemos entonces definir en el conjunto de vértices la relación $i\mathcal{R}j$ si existe al menos un camino entre i y j y otro entre j e i ó si $i = j$. Esta relación es también de equivalencia; sus clases de equivalencia determinan una partición en el conjunto de vértices y se llaman **componentes fuertemente conexas**. La clase de equivalencia de un vértice i es por tanto el conjunto de vértices que están comunicados con él, en el sentido de que es posible trasladarse desde i a cualquier vértice de su componente fuertemente conexa y volver.

1.3. Representación matricial de grafos

Para tratar computacionalmente los grafos tenemos dos representaciones matriciales: la matriz de adyacencia y la matriz de incidencia.

Para un grafo con n vértices, la *matriz de adyacencia* es una matriz $n \times n$.

- En el caso de grafos no orientados, en la coordenada (i, j) se pone el número de aristas $\{i, j\}$ que hay en E. Nótese que como es un grafo no orientado, en este caso la matriz es simétrica, con lo que sólo necesitamos $\frac{n(n-1)}{2}$ coeficientes.
- En el caso de grafos orientados, en la coordenada (i, j) se pone el número de arcos (i, j) que hay en A.

Ejemplo 5. *Para los grafos de la Figura 1, las matrices de adyacencia son:*

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ & 0 & 0 & 1 \\ & & 1 & 0 \\ & & & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Para un grafo con n vértices y m aristas o arcos, la *matriz de incidencia* es una matriz $n \times m$. En este caso tenemos que numerar las aristas o los arcos. Cada columna de la matriz está asociada a una arista o un arco, y las filas están asociadas a los vértices.

- En el caso de grafos no orientados, dada la arista $\{i, j\}$, cuyo orden de numeración es k, se pone en las coordenadas i y j de la columna k un 1 y en el resto de coordenadas un 0.

- En el caso de grafos orientados, dado el arco (i, j), cuyo orden de numeración es k, se pone un 1 en la coordenada i y un -1 en la coordenada j de la columna k. En el caso de que el arco sea el bucle (i, i), se pone un 1 en la coordenada i.

Ejemplo 6. *Para los grafos de la Figura 1, las matrices de incidencia son:*

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \end{pmatrix}$$

1.4. Árboles

Definición 1. *Dado un grafo no orientado, se define un* **árbol** *como un grafo simple conexo y sin ciclos.*

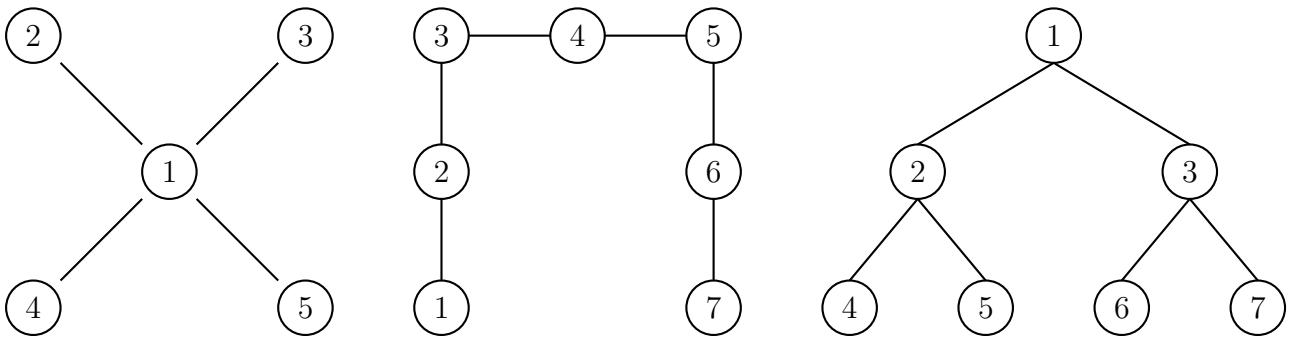


Figura 4: Ejemplos de árboles.

El concepto de árbol es muy importante en la práctica porque representa la situación en que es posible realizar conexiones entre los distintos vértices, ya que es conexo, y no hay conexiones redundantes, puesto que no hay ciclos. Nótese entonces que un árbol es un compromiso entre dos conceptos que parecen opuestos: es conexo, y para ello necesita un número mínimo de aristas, pero al mismo tiempo es acíclico, con lo que el número de aristas está limitado. Esto queda reflejado en el siguiente resultado:

Teorema 1. *Dado un grafo simple no orientado $G = (V, E)$ son equivalentes:*

1. *G es un árbol.*
2. *Dados dos vértices existe una única cadena que los une.*
3. *G es conexo y si se quita una arista se pierde la conexión.*

4. *G no tiene ciclos y si se añade una arista se forma un único ciclo.*

Demostración: $1 \Rightarrow 2$) Sean $i, j \in V$. Como G es un árbol, entonces es conexo, luego existe una cadena que los une

$$i = v_0 - v_1 - ... - v_r = j.$$

Supongamos que existiese otra cadena conectando los vértices i y j, dada por

$$i = w_0 - w_1 - ... - w_r = j.$$

Sea v_{k+1} el primer vértice tal que $v_0 = w_0, ..., v_k = w_k, v_{k+1} \neq w_{k+1}$. Sea v_l con $l > k+1$ el primer vértice tal que exista p tal que $v_l = w_p$. Entonces,

$$v_k - v_{k+1} - ... - v_{l-1} - v_l = w_p - w_{p-1} - w_{p-2} - ... - w_{k+1} - w_k = v_k$$

es un ciclo, lo que contradice que G sea un árbol.

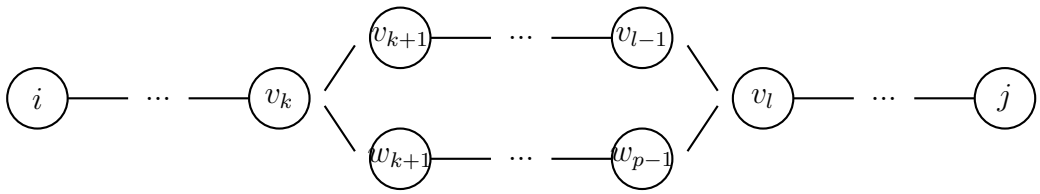


Figura 5: Ilustración del ciclo.

$2 \Rightarrow 3$) El grafo es conexo porque dados dos vértices existe una cadena que los une.

Supongamos que se suprime la arista $\{i, j\}$; entonces, como sólo hay una cadena uniendo los vértices i y j, se perdería la conexión entre estos dos vértices.

$3 \Rightarrow 4$) Supongamos que G tiene un ciclo

$$v_1 - v_2 - ... - v_r - v_1.$$

Suprimimos una arista del ciclo, por ejemplo $v_1 - v_2$. Sean $i, j \in V$ y supongamos que existe una cadena conectando i con j que usa $\{v_1, v_2\}$. Es decir,

$$i = w_0 - ... - w_p - v_1 - v_2 - w_{p+1} - ... - w_r = j.$$

Entonces podemos considerar la cadena

$$i = w_0 - ... - w_p - v_1 - v_r - ... - v_3 - v_2 - w_{p+1} - ... - w_r = j.$$

De esta forma se mantiene la conexión aunque se suprima una arista, lo que contradice la hipótesis.

Supongamos ahora que se añada una arista $\{i, j\}$. Entonces, como hay una cadena que une los vértices i y j

$$i - v_1 - ... - v_r - j,$$

al añadir $\{i, j\}$ se forma el ciclo

$$i - v_1 - ... - v_r - j - i.$$

Supongamos que se formasen dos ciclos. Como G es acíclico, entonces $\{i, j\}$ tiene que estar en ambos. Entonces, procediendo como $1 \Rightarrow 2$:

$$i - v_1 - .... - v_r - j - i$$

$$i - w_1 - ... - w_p - j - i.$$

Sea v_{k+1} el primer vértice tal que $v_1 = w_1, ..., v_k = w_k, v_{k+1} \neq w_{k+1}$. Sea v_l con $l > k+1$ el primer vértice tal que exista p tal que $v_l = w_p$. Entonces,

$$v_k - v_{k+1} - ... - v_{l-1} - v_l = w_p - w_{p-1} - w_{p-2} - ... - w_{k+1} - w_k$$

es un ciclo de G, lo que contradice que G sea acíclico.

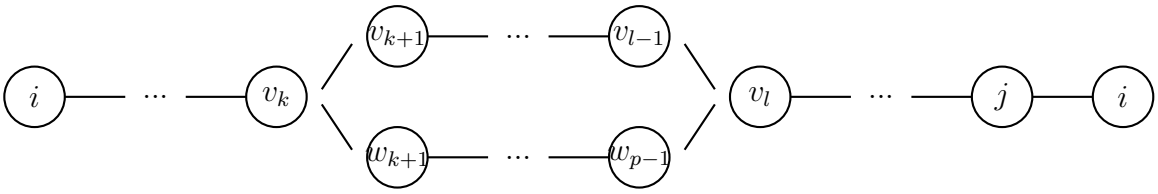


Figura 6: Ilustración del ciclo.

$4 \Rightarrow 1$) Sólo tenemos que demostrar la conexión. Sean $i, j \in V, i \neq j$ tales que la arista $\{i, j\} \notin E$. Al añadir la arista $\{i, j\}$, por hipótesis se forma un ciclo; además, como G es acíclico, entonces $\{i, j\}$ tiene que estar en el ciclo. Sea el ciclo

$$i - v_1 - ... - v_r - j - i.$$

Pero entonces existe una cadena que une i con j en G dada por $i - v_1 - ... - v_r - j$. □

Por lo tanto, un árbol se caracteriza porque tiene el número máximo de aristas que puede tener sin que se formen ciclos (condición 4), y al mismo tiempo, tiene el mínimo número de aristas necesarias para ser conexo (condición 3).

Lema 1. *Dado un árbol $G = (V, E)$, necesariamente existe un vértice que tiene grado de incidencia uno.*

Demostración: Supongamos que no hay ningún vértice en estas condiciones. Entonces, como el grafo es conexo, todos los vértices tienen grado de incidencia al menos dos. Tomamos entonces cualquier vértice v_0; como G es conexo, existe v_1 adyacente a v_0. Como $d(v_1) \geq 2$, existe v_2 adyacente a v_1 y distinto de v_0. Reiterando el proceso, en algún momento añadiremos v_{i+1} adyacente a v_i y distinto de v_{i-1}, pero tal que $v_{i+1} = v_p$ con $p < i - 1$. Entonces,

$$v_p - v_{p+1} - ... - v_i - v_{i+1} = v_p,$$

es un ciclo, lo que es una contradicción. □

Los vértices con grado de incidencia 1 se llaman *vértices pendientes.*

Veamos ahora una caracterización de grafo que nos va a ser de gran utilidad más adelante.

Teorema 2. *Dado un grafo no orientado $G = (V, E)$ con $|V| = n$; son equivalentes:*

1. *G es un árbol (conexo y sin ciclos).*
2. *G no tiene ciclos y tiene $n - 1$ aristas.*
3. *G es conexo y tiene $n - 1$ aristas.*

Demostración: $1 \Rightarrow 2)$ y $1 \Rightarrow 3)$ Se demostrará que el número de aristas es $n - 1$ por inducción en $n \equiv$ número de vértices.

Para $n = 2$ hay un solo árbol y tiene una arista. Luego el resultado es cierto.

Sea $n > 2$ y supongamos cierto el resultado hasta $n - 1$. Como tenemos un árbol, aplicando el lema anterior tenemos un vértice pendiente; si suprimimos ese vértice y la arista que incide en él, se obtiene otro árbol con un vértice menos. En efecto, es acíclico porque el grafo anterior era acíclico y hemos suprimido una arista. Y es conexo porque hemos suprimido la arista que une un vértice pendiente con el resto del grafo y entonces dados dos vértices distintos del vértice pendiente que se ha suprimido, la cadena que los une en el grafo inicial no usa la arista suprimida.

Por inducción ese árbol tiene $n - 2$ aristas, luego el árbol original tiene $n - 1$ aristas.

$2 \Rightarrow 1)$ Supongamos que el grafo no es conexo. Entonces tenemos más de una componente conexa, es decir, podemos particionar V en $V_1, ..., V_r, r > 1$. Consideremos $G_i := (V_i, E_i)$ con $E_i := \{\{a, b\} : a, b \in V_i, \{a, b\} \in E\}$. Entonces,

$$E = E_1 \cup ... \cup E_r.$$

Ahora bien, como G no tiene ciclos, entonces G_i, $\forall i$ tampoco los tiene, y como V_i son las componentes conexas, entonces G_i, $\forall i$ es un árbol. Entonces tiene $|E_i| - 1$ aristas. Luego el número de aristas sería

$$|E_1| - 1 + ... + |E_r| - 1 < |E| - 1,$$

lo que es una contradicción.

$3 \Rightarrow 1)$ Supongamos que se forma un ciclo con r aristas

$$v_1 - v_2 - ... - v_r - v_1.$$

Entonces, estas r aristas conectan r vértices y nos quedarían $n-r-1$ aristas para conectar a este conjunto las $n-r$ aristas que faltan, lo que no es posible. □

En definitiva, para comprobar que un grafo de n vértices es un árbol, basta comprobar dos de las tres propiedades siguientes: conexión, ausencia de ciclos o $n-1$ aristas.

1.5. Ciclos eulerianos

Los ciclos eulerianos son el primer problema de Teoría de Grafos y aparece propuesto en el problema de los puentes de Königsberg.

Dado un grafo no orientado $G=(V,E)$, un **ciclo euleriano** es un ciclo que usa todas las aristas de E pero una sola vez.

Proposición 1. *Para que un ciclo euleriano exista es necesario y suficiente que el grafo sea conexo y que el grado de inicidencia de todos los vértices sea par.*

Demostración: La demostración de este resultado nos da además el algoritmo de resolución. Se trata de escoger un vértice cualquiera y empezar el ciclo. Como el grado de incidencia es par y el grafo es conexo, podemos salir de ese vértice a otro; ahora, se puede salir de ese vértice porque su grado de incidencia es par, y así sucesivamente tal y como se hizo en el Lema 1. Sólo podemos parar el proceso (es decir, no podremos seguir añadiendo aristas) si llegamos al vértice inicial. De esta forma se ha formado un ciclo, pero es posible que no hayamos utilizado todas las aristas. Entonces debemos escoger un vértice en el que todavía no se han utilizado todas las aristas; ahora tenemos un grado de incidencia par en las aristas no utilizadas para todos los vértices puesto que se ha usado un número par de aristas incidente con cada vértice. Por lo tanto, nos sucederá lo mismo que antes. Y este proceso sigue hasta que se hayan utilizado todas las aristas. Tendremos entonces un conjunto de ciclos disjuntos $C_1,...,C_r$. Ahora, debido a la conexión, estos ciclos pueden combinarse en un gran ciclo, que será el ciclo euleriano buscado. En efecto, sea $x \in C_i = x - x_2 - ... - x_r - x, y \in C_j, i \neq j$. Como el grafo es conexo, existe una cadena que los une

$$v_1 = x - v_2 - ... - v_{r-1} - v_r = y.$$

Sea $v_1 - v_2$. Si está en C_i se pasa a $v_2 - v_3$. En caso contrario, $v_1 - v_2$ estará en una componente $C_k := v_1 - v_2 - w_3 - ... - w_s - v_1$ y podemos formar un ciclo con C_i, C_k dado por

$$x - v_2 - w_3 - ... - w_s - x - ...x_r.$$

Reiterando podemos crear un ciclo en el que están x e y. Y haciendo esto mientras haya más de un ciclo podemos unificar todos los ciclos en un ciclo euleriano.

Si el grafo no fuese conexo, no puede existir un ciclo euleriano. Supongamos entonces que es conexo pero que algún vértice tiene grado de inicidencia impar. Entonces, al formar el ciclo nos encontraremos que en algún momento se entra en el vértice pero no es posible salir por una arista no seleccionada, lo que implica que no se puede formar un ciclo euleriano. □

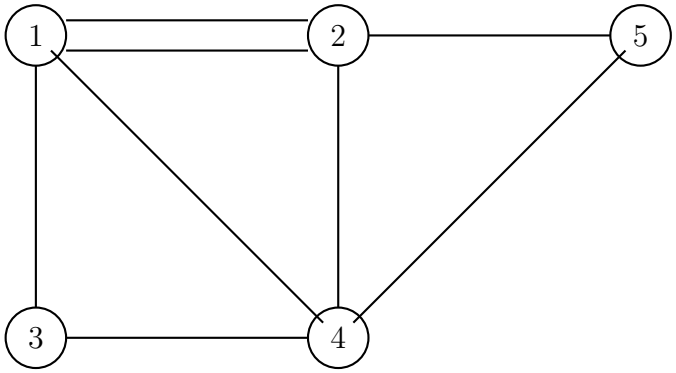


Figura 7: Ejemplo para ciclos eulerianos.

Ejemplo 7. *Consideremos el grafo no orientado de la Figura 5.*

Nótese que por lo anterior sabemos que es posible construir un ciclo euleriano. Si comenzamos con el vértice 1, tenemos por ejemplo el ciclo 1-2-4-3-1. Si volvemos a comenzar en el vértice 1, tenemos el ciclo 1-2-5-4-1.

Y ahora podemos unir estos dos ciclos en 1-2-5-4-1-2-4-3-1, que nos proporciona uno de los posibles ciclos eulerainos.

2. Árbol de unión de valor mínimo

Consideremos el siguiente ejemplo.

Ejemplo 8. *En un parque natural existe una red de caminos angostos para los guardabosques. La siguiente figura muestra este sistema.*

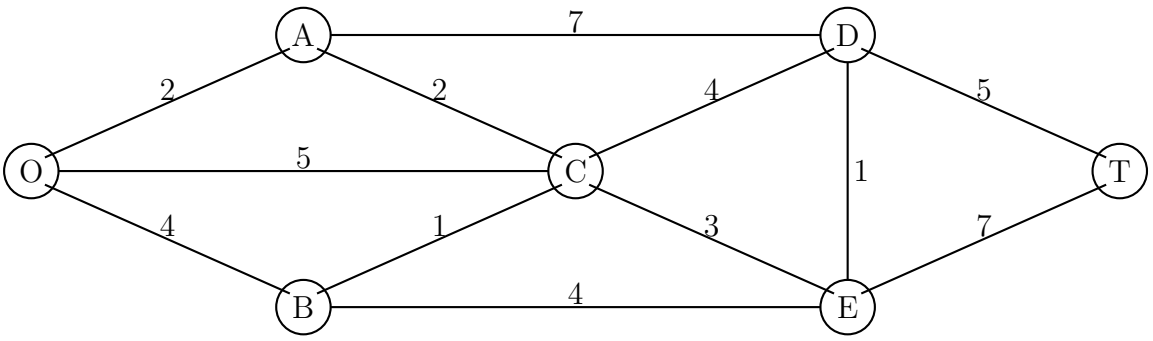


Figura 8: Grafo del Ejemplo 8.

La entrada del parque es O; las otras letras representan las casetas de los guardabosques y los números sobre cada arista representan las longitudes en kilómetros de esos caminos.

Se quieren instalar líneas telefónicas que conecten todas las casetas y la entrada del parque. ¿Por dónde deben tenderse esas líneas para lograr comunicar cualquier par de casetas de forma que se minimice el número total de kilómetros de cable utilizado?

Definición 2. *Una* **red no orientada** *es un par* (G, d) *donde* $G = (V, E)$ *es un grafo no orientado simple y* d *es una aplicación*

$$d : E \to \mathbb{R}^+.$$

Ahora, dado un grafo parcial de (G, E'), se define su *valor* como

$$\sum_{a_i \in E'} d(a_i).$$

El problema que nosotros tenemos que resolver es hallar un grafo parcial de forma que se tenga la conexión y que el coste de esas conexiones sea mínimo. Nótese que como $d(a_i) \geq 0$, si no tenemos un árbol pero sí conexión, siempre podemos eliminar algunas aristas y conservar la conexión de forma que tengamos un árbol. Por ello, el problema se restringe a encontrar el conjunto de aristas que sea árbol y cuyo valor sea mínimo. Esto se conoce como *árbol de unión de valor mínimo*.

El origen de la aplicación d es en distancias, de ahí que sólo tome valores positivos. Sin embargo, para hallar un árbol de valor mínimo no necesitamos esta restricción, pues podemos sumar a todos las aristas el mínimo valor de todas ellas sin que esto afecte al punto en el que se alcanza el mínimo, ya que todos los árboles tienen el mismo número de aristas. Nótese sin embargo que si hay aristas con valor negativo, el grafo parcial de menor valor no es necesariamente un árbol.

Este problema puede plantearse como un problema de P. Entera. Para ello, si denotamos por x_{ij} la variable binaria que toma el valor 1 si la arista $\{i, j\}$ está en el árbol buscado y 0 en caso contrario, se tiene que el problema de hallar el árbol de unión de valor mínimo se puede escribir como

$$\begin{aligned}
\text{mín} \quad & \sum_{\{i,j\} \in E} d(i,j) x_{ij} \\
& \sum_{\{i,j\} \in E} x_{ij} = n - 1 \\
& \sum_{\{i,j\} \in E, i,j \in A} x_{ij} \leq |A| - 1 \quad \forall 3 \leq |A| \leq |V| - 1 \\
& \{i,j\} \in \{0,1\} \qquad\qquad \forall \{i,j\} \in E
\end{aligned}$$

Para que el problema tenga solución es necesario y suficiente que el grafo sea conexo. Para resolver este problema tenemos dos algoritmos específicos: el algoritmo de Prim y el algoritmo de Kruskal.

2.1. El algoritmo de Kruskal

El algoritmo de Kruskal se basa en la definición equivalente de árbol como un grafo con $n - 1$ aristas y acíclico. Para conseguir que sea un árbol de valor mínimo, en cada paso se toma la arista de menor valor que no haya sido considerada; para esta arista se

estudia si al unirla a las otras aristas seleccionadas se forma un ciclo; si es así, esta arista es descartada; en caso contrario, la arista es seleccionada para el árbol de unión de valor mínimo.

RESUMEN DEL ALGORITMO DE KRUSKAL

- **PASO INICIAL:** Ordenar las aristas en orden creciente de valor. En caso de empate en los valores de dos aristas, se puede escoger cualquiera de ellas.

  Además, inicializamos el conjunto de las aristas del árbol de valor mínimo a $\mathcal{T} = \emptyset$.

- **PASO ITERATIVO:** Se toma la siguiente arista no seleccionada, según el orden establecido en el paso inicial.

  - Si esta arista, junto con las que están en $\mathcal{T}$, forman un ciclo, entonces la arista se descarta.
  - Si esta arista, junto con las que están en $\mathcal{T}$, no forman un ciclo, entonces la arista se incluye en el árbol.

La condición de parada es que $\mathcal{T}$ tenga $n-1$ aristas.

El paso más complicado para grafos grandes es comprobar en cada iteración si se forma una ciclo al añadir la arista correspondiente o no. Una forma de programar esto es inicializando una función auxiliar $p : V \to \mathbb{Z}$ a 0, y se inicializa una función contador aux a 0. La función p indica si los vértices están conectados por una cadena de aristas en $\mathcal{T}$; así, si tiene el valor cero, el vértice está aislado y si dos vértices toman el mismo valor y es diferente de cero, entonces están conectados por aristas de $\mathcal{T}$. Cada vez que se considera una arista $\{i, j\}$ se miran los valores correspondientes de los extremos de la arista:

- Si $p(i) = p(j) = 0$, entonces $aux = aux + 1$ y $p(i) = p(j) = aux$ (se está formando una nueva componente con i y j). En este caso $\{i, j\}$ se incluye en $\mathcal{T}$.
- Si $p(i) = p(j) \neq 0$, entonces al añadir la arista al conjunto de aristas en $\mathcal{T}$ se forma un ciclo (porque en este caso ya estaban conectados por una cadena de aristas en $\mathcal{T}$). En este caso, $\{i, j\}$ no se incluye en $\mathcal{T}$,
- Si $p(i) > p(j) > 0$, entonces la arista se añade al conjunto $\mathcal{T}$ y todas las aristas con valor $p(i)$ toman el valor $p(j)$ (se unen las dos componentes). En este caso $\{i, j\}$ se incluye en $\mathcal{T}$,
- Si $p(i) > p(j) = 0$, entonces la arista se añade al conjunto $\mathcal{T}$ y $p(j)$ toma el valor $p(i)$.

Ejemplo 9. *Vamos a aplicar el algoritmo de Kruskal al Ejemplo 8. Las aristas ordenadas en orden creciente de valor son:*

$\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\},\{O,B\},\{B,E\},\{C,D\},\{O,C\},\{D,T\},\{A,D\},\{E,T\}$.

Aplicando ahora el paso principal del algoritmo de Kruskal se obtiene:

Iteración	O	A	B	C	D	E	T	*Arista*	$\mathcal{T}$
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	-	$\emptyset$
1	*0*	*0*	*1*	*1*	*0*	*0*	*0*	$\{B,C\}$	$\{\{B,C\}\}$
2	*0*	*0*	*1*	*1*	*2*	*2*	*0*	$\{D,E\}$	$\{\{B,C\},\{D,E\}\}$
3	*3*	*3*	*1*	*1*	*2*	*2*	*0*	$\{O,A\}$	$\{\{B,C\},\{D,E\},\{O,A\}\}$
4	*1*	*1*	*1*	*1*	*2*	*2*	*0*	$\{A,C\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\}\}$
5	*1*	*1*	*1*	*1*	*1*	*1*	*0*	$\{C,E\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\}\}$
6	*1*	*1*	*1*	*1*	*1*	*1*	*0*	$\{O,B\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\}\}$
7	*1*	*1*	*1*	*1*	*1*	*1*	*0*	$\{B,E\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\}\}$
8	*1*	*1*	*1*	*1*	*1*	*1*	*0*	$\{C,D\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\}\}$
9	*1*	*1*	*1*	*1*	*1*	*1*	*0*	$\{O,C\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\}\}$
10	*1*	*1*	*1*	*1*	*1*	*1*	*1*	$\{D,T\}$	$\{\{B,C\},\{D,E\},\{O,A\},\{A,C\},\{C,E\},\{D,T\}\}$

Gráficamente esto aparece en la Figura 9

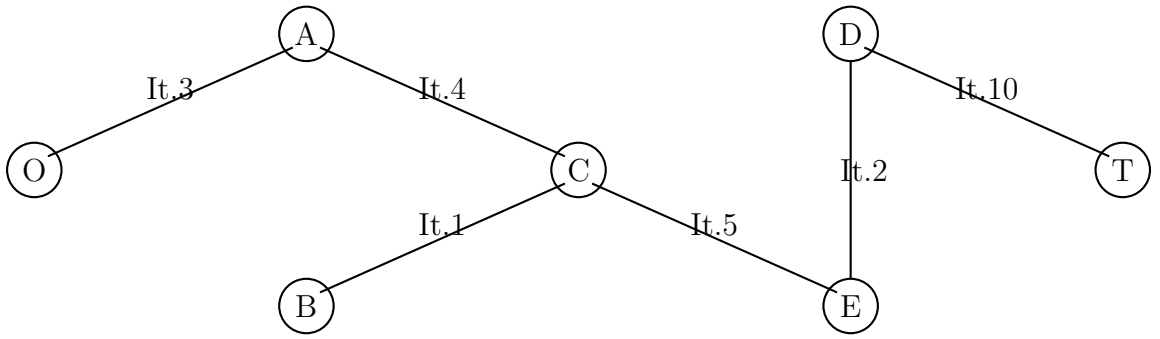


Figura 9: Ejemplo de aplicación del algoritmo de Kruskal

Proposición 2. *El algoritmo de Kruskal obtiene un árbol de valor mínimo.*

Demostración: Sea $\mathcal{T} := \{a_1, ..., a_{n-1}\}$ el árbol encontrado por el algoritmo, donde las aristas están ordenadas de menor a mayor valor. Supongamos que existe $\mathcal{T}' := \{b_1, ..., b_{n-1}\}$ óptimo mejor que $\mathcal{T}$, con las aristas también ordenadas.

Como $\mathcal{T}$ y $\mathcal{T}'$ no son iguales, existe i tal que $a_i \neq b_i$ pero $a_j = b_j, j < i$. Sea $\mathcal{T}' \cup \{a_i\}$; como $\mathcal{T}'$ es un árbol, sabemos que en $\mathcal{T}' \cup \{a_i\}$ se ha formado un único ciclo, en el que está a_i. Si eliminamos cualquier arista del ciclo se mantiene la conexión y hay $n-1$ aristas, luego es un árbol.

Suprimimos la arista de mayor valor en el ciclo. Por construcción, existe b_k en el ciclo con $k \geq i$ y entonces $d(b_k) \geq d(a_i)$. En otro caso, como $a_j = b_j$ si $j < i$ esto significa que se formaría un ciclo en $\mathcal{T}$, lo que no es posible porque $\mathcal{T}$ es un árbol.

Tenemos entonces dos casos:

- Si existe $d(b_k) > d(a_i)$, entonces hemos encontrado un árbol $(\mathcal{T}' \cup \{a_i\} \backslash \{b_k\})$ mejor que $\mathcal{T}'$, lo que contradice que $\mathcal{T}'$ sea óptimo.
- Si $d(b_k) = d(a_i), \forall k \geq i$ en el ciclo, eliminamos una de ellas y reiteramos el proceso. Como se parte de la hipótesis de que el árbol $\mathcal{T}$ obtenido por el algortimo no es óptimo, en un número finito de pasos se llegará a que se elimina una arista con valor mayor que el de la arista que se introduce en el árbol, lo que contradice la optimalidad de $\mathcal{T}'$. ■

2.2. El algoritmo de Prim

El algoritmo de Prim se basa en la definición equivalente de árbol como un grafo con $n-1$ aristas y conexo. Para conseguir que sea un árbol de valor mínimo, en cada paso se toma la arista de menor valor que nos permita conectar el conjunto de vértices seleccionados con el conjunto de vértices no seleccionados.

RESUMEN DEL ALGORITMO DE PRIM

- **INICIALIZACIÓN:** Inicializamos el conjunto de las aristas del árbol de valor mínimo a $\mathcal{T} = \emptyset$. Se toma un vértice cualquiera i del grafo y se inicializa $AUX = \{i\}$.
- **PASO ITERATIVO:** Se considera el conjunto de aristas en las que un vértice está en AUX y el otro vértice está en AUX^c.

  Para este conjunto, se toma la arista de menor valor. Si hubiese varias aristas de valor mínimo, se puede coger cualquiera de ellas. Sea $\{j, k\}$ esta arista, donde $j \in AUX$ y $k \in AUX^c$.

  Entonces, $\mathcal{T} = \mathcal{T} \cup \{j, k\}$, y $AUX = AUX \cup \{k\}$.

La condición de parada es que $\mathcal{T}$ tenga $n-1$ aristas, o equivalentemente que $AUX = V$.

Ejemplo 10. *Si aplicamos el algoritmo de Prim al Ejemplo 8 se obtiene:*

- **Inicialización:** $AUX = \{O\}, \mathcal{T} = \emptyset$.
- **Iteración 1:** *Tenemos las aristas* $\{O, A\}, \{O, B\}, \{O, C\}$. *Entonces,*

$$AUX = \{O, A\}, \mathcal{T} = \{\{O, A\}\}.$$

- **Iteración 2:** *Tenemos las aristas* $\{O, B\}, \{O, C\}, \{A, C\}, \{A, D\}$. *Entonces,*

$$AUX = \{O, A, C\}, \mathcal{T} = \{\{O, A\}, \{A, C\}\}.$$

- **Iteración 3:** *Tenemos las aristas* $\{O, B\}, \{A, D\}, \{C, B\}, \{C, D\}, \{C, E\}$. *Entonces,*

$$AUX = \{O, A, C, B\}, \mathcal{T} = \{\{O, A\}, \{A, C\}, \{C, B\}\}.$$

- **Iteración 4:** *Tenemos las aristas* $\{A, D\}, \{C, D\}, \{C, E\}, \{B, E\}$. *Entonces,*

$$AUX = \{O, A, C, B, E\}, \mathcal{T} = \{\{O, A\}, \{A, C\}, \{C, B\}, \{C, E\}\}.$$

- **Iteración 5:** *Tenemos las aristas* $\{A, D\}, \{C, D\}, \{E, D\}, \{E, T\}$. *Entonces,*

$$AUX = \{O, A, C, B, E, D\}, \mathcal{T} = \{\{O, A\}, \{A, C\}, \{C, B\}, \{C, E\}, \{E, D\}\}.$$

- **Iteración 6:** *Tenemos las aristas* $\{D, T\}, \{E, T\}$. *Entonces,*

$$AUX = \{O, A, C, B, E, D, T\}, \mathcal{T} = \{\{O, A\}, \{A, C\}, \{C, B\}, \{C, E\}, \{E, D\}, \{D, T\}\}.$$

Gráficamente el proceso seguido con el algortimo se ve en la Figura 10

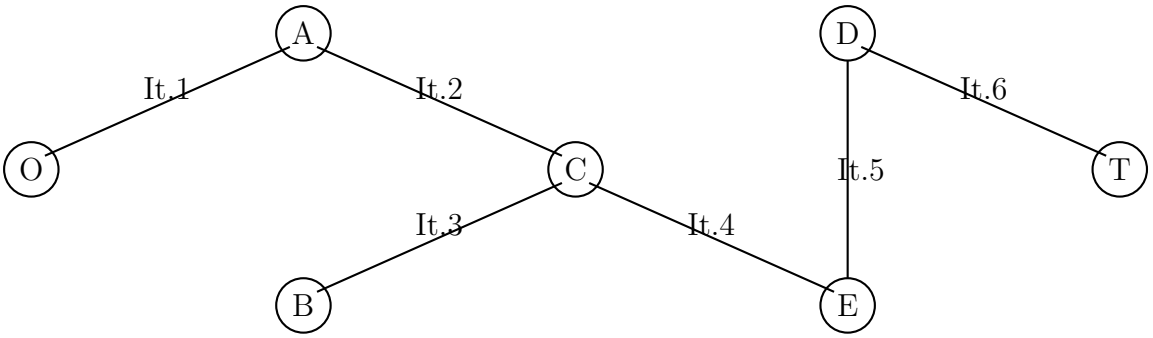


Figura 10: Ejemplo de aplicación del algoritmo de Prim.

Proposición 3. *El algoritmo de Prim obtiene un árbol de valor mínimo.*

Demostración: Sea $\mathcal{T} := \{a_1, ..., a_{n-1}\}$ el árbol encontrado por el algoritmo, donde las aristas están ordenadas por orden de inclusión. Supongamos que existe $\mathcal{T}' := \{b_1, ..., b_{n-1}\}$ óptimo mejor que $\mathcal{T}$, con las aristas ordenadas siguiendo en lo posible el orden de las aristas de $\mathcal{T}$.

Como $\mathcal{T}$ y $\mathcal{T}'$ no son iguales, existe i tal que $a_i \neq b_i$ pero $a_j = b_j, j < i$. Sea $\mathcal{T}' \cup \{a_i\}$; sabemos que se ha formado un único ciclo, en el que está a_i. Si eliminamos cualquier arista del ciclo se mantiene la conexión y hay $n-1$ aristas, luego es un árbol.

Suprimimos la arista de mayor valor del ciclo. Por construcción, existe una arista b_k en el ciclo tal que $d(b_k) \geq d(a_i)$. Esto es debido a que a_i conecta AUX_i y AUX_i^c. Entonces el ciclo tiene que conectar AUX_i y AUX_i^c de otra forma, que por construcción no puede ser mejor que a_i.

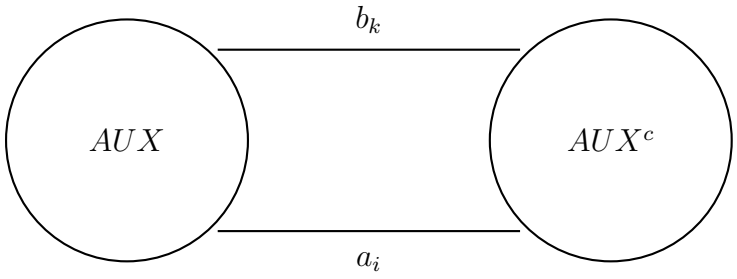


Figura 11: Ilustración del ciclo.

Tenemos entonces dos casos:

- Si existe $d(b_k) > d(a_i)$, entonces hemos encontrado un árbol $(\mathcal{T}' \cup \{a_i\} \backslash \{b_k\})$ mejor que $\mathcal{T}'$, lo que contradice que $\mathcal{T}'$ sea óptimo.
- Si ninguna arista $d(b_k)$ es mayor que $d(a_i)$, eliminamos una de ellas con igual valor de a_i y reiteramos el proceso. Como se parte de la hipótesis de que el árbol $\mathcal{T}$ obtenido por el algortimo no es óptimo, en un número finito de pasos se llegará a que se elimina una arista con valor mayor que el de la arista que se introduce en el árbol, lo que contradice la optimalidad de $\mathcal{T}'$. ■

3. Camino de menor valor

Consideremos el siguiente ejemplo.

Ejemplo 11. *Una ambulancia se encuentra estacionada en el vértice 1 del grafo de la Figura 9 y puede ser solicitada desde cuatro centros hospitalarios, que se corresponden con los vértices 2 a 5. No existe conexión directa entre los distintos vértices y en ocasiones sólo se tiene una calle de sentido único. Esta situación está reflejada en el grafo. Los valores de los arcos denotan el tiempo en minutos que dura el viaje entre dos vértices. ¿Cuál es la forma más rápida de acudir a cada uno de los hospitales.*

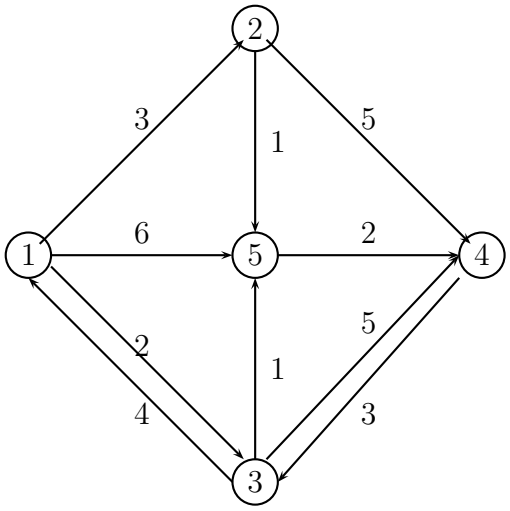


Figura 12: Grafo del Ejemplo 11.

Definición 3. *Una* **red orientada** *es un par* (G, d) *donde* $G = (V, A)$ *es un grafo orientado simple y* d *es una aplicación*

$$d : A \to \mathbb{R}.$$

Ahora, dado un grafo parcial de (V, A'), se define su *valor* como

$$\sum_{a_i \in A'} d(a_i).$$

El problema que nosotros tenemos que resolver es hallar un camino desde un vértice O a otro de forma que su valor sea lo menor posible. En los algoritmos que estudiaremos se halla el camino de menor valor desde un vértice O a todos los demás vértices. Para que el problema tenga solución es necesario que exista al menos un camino desde O a cualquier otro vértice; sin embargo, esta condición no es suficiente, puesto que podría existir un circuito de valor negativo. Debemos entonces evaluar si estos ciclos de valor negativo existen. Este problema puede resolverse como un problema de P. Entera. Para ello, definimos x_{ij} que la variable entera que representa si se usa el arco (i, j).

$$\begin{aligned} \text{mín} \quad & \sum_{(i,j)\in A} d(i,j)x_{ij} \\ & \sum_i x_{ij} = \sum_k x_{jk} \quad \forall j \in V \\ & x_{ij} \in \{0,1\} \end{aligned}$$

donde la restricción indica que todas las veces que se entra en un vértice se tiene que salir.

Si hay un circuito de valor negativo, entonces el problema anterior tiene una solución óptima negativa. En caso de que no haya circuitos negativos, entonces el problema de

hallar el camino de menor desde O hasta P puede plantearse también como un problema de programación entera

$$\begin{aligned} \text{mín} \quad & \sum_{(i,j)\in A} d(i,j)x_{ij} \\ & \sum_i x_{ij} = \sum_k x_{jk} \quad \forall j \neq O, P \\ & \sum_i x_{Oi} = 1 \\ & \sum_i x_{iP} = 1 \\ & x_{ij} \in \{0,1\} \end{aligned}$$

(nótese que como no hay circuitos de valor negativo, entonces $\sum_i x_{ij} = 1 = \sum_k x_{jk}, \forall j \neq O, P$).

Sin embargo, como pasaba en el caso del árbol de unión de valor mínimo, existen algoritmos específicos para resolver este problema. Para resolver este problema tenemos dos algoritmos: el algoritmo de Dijkstra y el algoritmo de Bellman-Ford.

Para justificar estos algoritmos necesitamos tener en cuenta el *principio de optimalidad de Bellman*, que dice lo siguiente:

Si tenemos un camino de menor valor entre dos vértices i y j, y este camino pasa por dos vértices k, l, entonces la parte del camino que une estos dos vértices constituye un camino de menor valor entre k y l.

Como consecuencia de este principio, si denotamos por $v(i)$ el valor del camino de menor valor desde O hasta i, entonces se cumple la condición

$$v(i) = \min_{j\in\Gamma^-(i)} (v(j) + d_{ij}).$$

3.1. El algoritmo de Dijkstra

Este algoritmo necesita que todos los valores de los arcos sean no negativos (lo que evita circuitos negativos) y calcula todos los caminos desde un vértice predeterminado a cualquier otro vértice. El algoritmo de Dijkstra funciona de la siguiente manera:

RESUMEN DEL ALGORITMO DE DIJKSTRA

- **INICIALIZACIÓN:** Definimos $p(i) = 1$, $\forall i, AUX = \{1\}$, e inicializamos la función v por

$$v(i) := \begin{cases} 0 & \text{si} \quad i = 1 \\ d(1,i) & \text{si} \quad i \in \Gamma(1) \\ \infty & \text{en otro caso} \end{cases}$$

- **PASO ITERATIVO:** Se escoge el vértice $j \notin AUX$ tal que tenga el menor valor de v; si hubiese varios en estas condiciones se puede escoger cualquiera de ellos.

  Para los vértices $i \in \Gamma(j) \cap AUX^c$, si $v(j) + d(j,i) < v(i)$, entonces se actualiza $p(i) \to j$ y $v(i) \to v(j) + d(j,i)$.

La función p identifica al predecesor inmediato; esta función se va actualizando en las sucesivas iteraciones, de forma que en la iteración final nos dará el predecesor inmediato en el camino de menor valor. La función v nos da el valor del camino de menor valor; como en el caso de p, sus valores se van actualizando en cada iteración y en la última nos dará el valor del camino de menor valor. El conjunto AUX es el conjunto de vértices para los que ya se ha hallado el camino de menor valor. La condición de parada es que $AUX = V$ o equivalentemente, que se hayan realizado $n-1$ iteraciones. Nótese que la última iteración no produce ningún cambio en la tabla, ya que el último vértice seleccionado no tiene vértices sucesores sin seleccionar.

Por otra parte, si en algún momento todos los vértices en AUX^c tienen valor infinito, esto implica que no existe un camino desde 1 a ninguno de esos vértices.

Ejemplo 12. *Si consideramos el ejemplo con el que hemos comenzado esta sección y aplicamos el algoritmo de Dijkstra se obtiene:*

	1	*2*	*3*	*4*	*5*	*Aux*
p(i)	*1*	*1*	*1*	*1*	*1*	$\{1\}$
v(i)	*0*	*3*	*2*	∞	*6*	
p(i)	*1*	*1*	*1*	*3*	*3*	$\{1,3\}$
v(i)	*0*	*3*	*2*	*7*	*3*	
p(i)	*1*	*1*	*1*	*3*	*3*	$\{1,3,2\}$
v(i)	*0*	*3*	*2*	*7*	*3*	
p(i)	*1*	*1*	*1*	*5*	*3*	$\{1,3,2,5\}$
v(i)	*0*	*3*	*2*	*5*	*3*	
p(i)	*1*	*1*	*1*	*5*	*3*	$\{1,3,2,5,4\}$
v(i)	*0*	*3*	*2*	*5*	*3*	

Y como conocemos la función p que nos proporciona el predecesor inmediato en el camino de menor valor, los valores de la última iteración nos permiten conocer el camino de menor valor a cada uno de los vértices. Por ejemplo, para 4, como su predecesor inmediato es el 5, del 5 es el 3 y del 3 es el 1, tenemos que el camino de menor valor es $1 \to 3 \to 5 \to 4$. *Gráficamente, los caminos de menor valor son los de la Figura 13.*

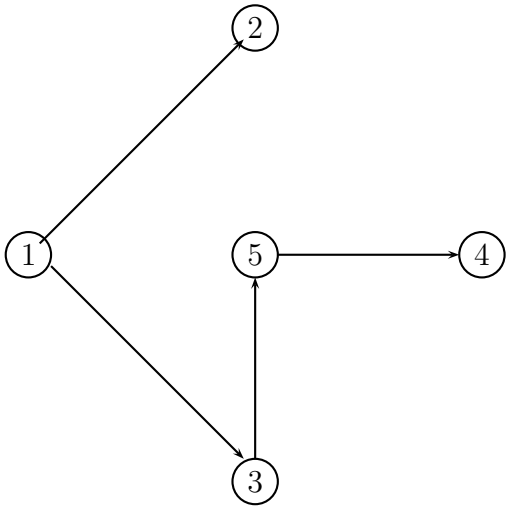


Figura 13: Ejemplo de aplicación del algoritmo de Dijkstra.

Proposición 4. *El algoritmo de Dijkstra obtiene los caminos de menor valor desde un vértice a todos los demás.*

Demostración: Se demuestra por inducción en la iteración que para el punto incluido en AUX el algoritmo encuentra el camino de menor valor desde O a ese vértice.

Para la primera iteración, se obtiene el camino de menor valor desde el vértice O al vértice seleccionado v_1, ya que todas las aristas tienen valor positivo y se escoge el vértice adyacente a O cuya arista es la de menor valor. Si hubiese otro camino mejor, debería llegar desde otro vértice v_j, pero entonces

$$v(j) + d_{v_1,v_j} \geq d_{O,v_1},$$

lo que es una contradicción.

Supongamos entonces que hasta la iteración $i-1$ se han obtenido los caminos de menor valor en los vértices seleccionados $v_1, ..., v_{i-1}$, y veamos que entonces en la siguiente iteración se obtiene también el camino de menor valor al vértice seleccionado en esta iteración v_i. Sea el camino hallado

$$O - v_{i1} - ... - v_{ir} - v_i.$$

Por construcción, este camino tiene un valor menor o igual que cualquier otro camino cuyo punto inmediatamente anteroir a v_i esté en AUX. Supongamos entonces que existiese

otro camino que llegase a v_i y cuyo predecesor inmediato fuese $v_j \notin AUX$. Entonces, en algún momento este camino ha pasado desde un vértice de AUX v_r a un vértice fuera de AUX v_s. Entonces el camino es

$$O - ... - v_r - v_s - ... - v_j - v_i.$$

Pero por construcción la parte hasta v_s ya tiene un valor mayor o igual que el valor hallado, y como la función d es no-negativa, esto lleva a una contradicción. ■

3.2. El algoritmo de Bellman-Ford

Al contrario que el algoritmo anterior, este algoritmo es aplicable aunque existan arcos con valor negativo. La idea de este algoritmo es que en la iteración r se halla el camino de menor valor desde el vértice 1 a cualquier otro vértice utilizando como máximo r arcos. La estructura del algoritmo de Bellaman-Ford es la siguiente:

RESUMEN DEL ALGORITMO DE BELLMAN-FORD

- **INICIALIZACIÓN:** Definimos $p(i) = 1$, $\forall i$, e inicializamos la función v por

$$v(i) := \begin{cases} 0 & \text{si} \quad i = 1 \\ d(1,i) & \text{si} \quad i \in \Gamma(1) \\ \infty & \text{en otro caso} \end{cases}$$

- **PASO ITERATIVO:** Para cada uno de los vértices i se consideran sus predecesores. Se calcula $\min v(j) + d(j,i) := v$; si $v \leq v(i)$ y v se ha obtenido para el predecesor j', entonces hacemos $p(i) \to j'$ y $v(i) \to v$.

La condición de parada es que haya dos iteraciones en que se repitan los valores de p y v, o que se hayan realizado $n - 1$ iteraciones. Si al realizar una iteración adicional (la número n) se producen cambios, esto implica que existe un ciclo de valor negativo y que el problema no tiene solución.

Ejemplo 13. *Si consideramos el ejemplo con el que hemos comenzado esta sección y aplicamos el algoritmo de Bellman-Ford se obtiene:*

	1	2	3	4	5
$p(i)$	1	1	1	1	1
$v(i)$	0	3	2	∞	6
$p(i)$	1	1	1	3	3
$v(i)$	0	3	2	7	3
$p(i)$	1	1	1	5	3
$v(i)$	0	3	2	5	3
$p(i)$	1	1	1	5	3
$v(i)$	0	3	2	5	3

Proposición 5. *El algortimo de Bellman-Ford obtiene el camino de menor valor desde el vértice 1 a todos los demás vértices.*

Demostración: Esto se puede ver demostrando por inducción que en la iteración i se encuentra el camino de menor valor con menos de $i+1$ aristas desde 1 a todos los demás vértices.

Para la primera iteración el resultado es trivial. Y supuesto cierto el resultado hasta $i-1$, el resultado se obtiene a partir las ecuaciones de optimalidad de Bellman. ■

4. El problema de flujo máximo

Consideremos el siguiente ejemplo.

Ejemplo 14. *Consideremos el grafo de la Figura 14, en el que los arcos denotan tuberías y los valores sobre los arcos denotan el máximo número de litros que pueden circular por segundo por cada una de esas tuberías. Los valores sobre los vértices denotan la máxima cantidad que puede entrar en esos vértices; si no tiene valor, es que puede entrar una cantidad ilimitada.*

¿Cuál es el máximo número de litros que pueden circular por segundo desde F_1 y F_2 hacia S_1 y S_2?

Definición 4. *Una* **red capacitada** *es un par (G, c) donde $G = (V, A)$ es un grafo orientado simple de forma que*

- *Existe un único vértice que no tiene predecesores. Este vértice se llama fuente y lo denotaremos por F.*
- *Existe un único vértice que no tiene sucesores. Este vértice se llama sumidero y lo denotaremos por S.*

y c es una aplicación llamada capacidad

$$c : A \to \mathbb{R}^+.$$

Denotaremos por c_{ij} el valor de la capacidad c sobre el arco (i, j).

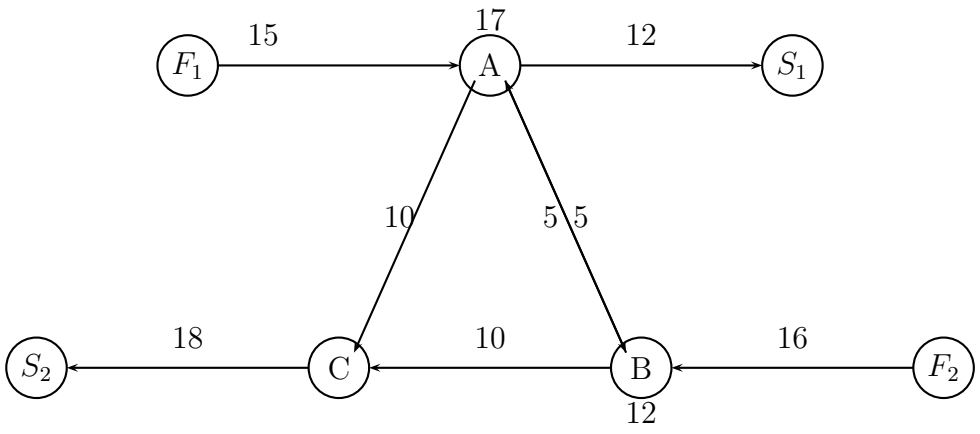


Figura 14: Ejemplo de problema de flujo máximo.

La aplicación c nos indica la máxima cantidad que se puede transportar por un arco por unidad de tiempo.

Ejemplo 15. *El grafo anterior no cumple la primera propiedad de flujo, ya que hay varios vértices que no tienen sucesores y varios vértices que no tienen predecesores; además, hay limitaciones en la cantidad que puede entrar en algunos vértices. Sin embargo, es posible construir un grafo equivalente que sea una red capacitada, obteniéndose el grafo de la Figura 15.*

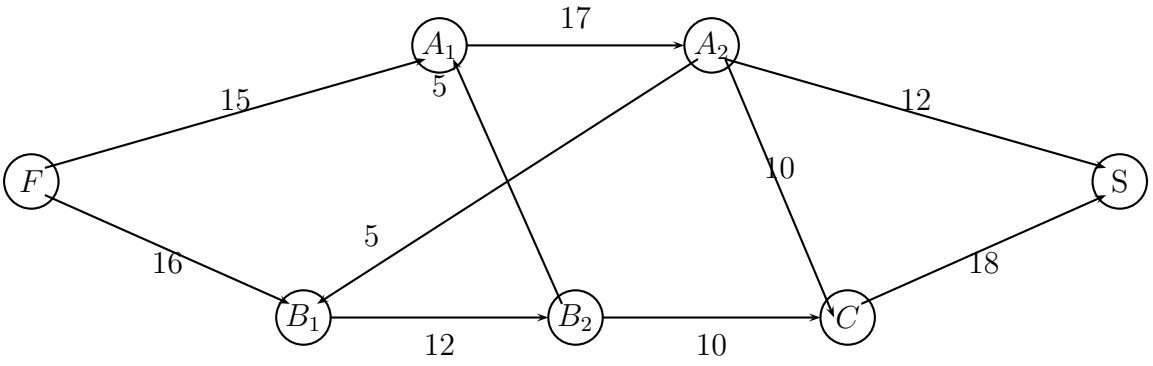


Figura 15: Ejemplo de red capacitada.

Definición 5. *Dada una red capacitada, un* **flujo** *es una aplicación $f : A \to \mathbb{R}$ que cumple las siguientes condiciones:*

- $0 \leq f_{ij} \leq c_{ij}, \forall (i,j) \in A$.
- *Para cualquier vértice i distinto de la fuente y el sumidero, se tiene que*

$$\sum_{(i,j)\in A} f_{ij} = \sum_{(k,i)\in A} f_{ki}.$$

El flujo modela una solución factible de cantidades que pueden pasar por el grafo desde la fuente al sumidero. Así, la primera propiedad indica que la cantidad que pasa por un arco no puede superar la capacidad del arco y tiene valor positivo; cuando el valor del flujo a través de un arco alcanza el valor de la capacidad del arco, se dice que el arco está *saturado*. La segunda propiedad se llama *conservación del flujo* y nos indica que, para los vértices intermedios, ni se crea ni se destruye flujo.

Definición 6. *Dado un flujo f, se define el* **valor del flujo** v_f *como*

$$v_f := \sum_{(F,i)\in A} f_{Fi} = \sum_{(j,S)\in A} f_{jS}.$$

El problema que nosotros tenemos que resolver es hallar un flujo de forma que su valor sea lo mayor posible.

El problema de hallar el flujo máximo en una red capacitada puede plantearse como un problema de Programación Lineal:

$$\begin{array}{lll} \text{máx} & \displaystyle\sum_{(F,i)\in A} f_{Fi} & \\ s.a & f_{ij} \geq 0, & \forall (i,j) \in A \\ & f_{ij} \leq c_{ij}, & \forall (i,j) \in A \\ & \displaystyle\sum_{j:(i,j)\in A} f_{ij} = \sum_{k:(k,i)\in A} f_{ij}, & \forall i \in V \backslash \{F,S\} \end{array}$$

Sin embargo, para resolver este problema utilizaremos un algoritmo que es más eficiente.

Una primera aproximación para resolver el problema sería considerar un camino desde la fuente al sumidero. Este camino pasaría por varios arcos, cada uno con su capacidad correspondiente. Una vez determinado el camino, se hace pasar por él la máxima cantidad posible, que coincide con la menor capacidad en los arcos del camino. Se añade este valor al flujo y de esta manera se satura al menos un arco, con lo que este camino deja de ser un camino válido para aumentar el flujo. A continuación se construye una nueva red en la que la capacidad de los arcos del camino es lo que resta para saturarlos y para los demás arcos se mantiene el valor. Se trata entonces de encontrar otro camino distinto en el que ningún arco esté saturado y repetir el proceso. El algoritmo terminaría cuando no se pudiese encontrar ningún camino en estas condiciones. Sin embargo, esta solución no es válida, como se puede ver en el siguiente ejemplo.

Ejemplo 16. *Consideremos la red capacitada de la Figura 16.*

Si consideramos el camino entre F y S dado por $F \to 2 \to 1 \to S$, entonces podemos pasar un flujo de valor 1. Y ya no es posible encontrar un camino por arcos no saturados entre F y S.

Sin embargo, es posible encontrar un flujo de valor 2 si consideramos los caminos $F \to 1 \to S$ y $F \to 2 \to S$ y hacemos pasar un flujo de valor 1 por cada uno de ellos.

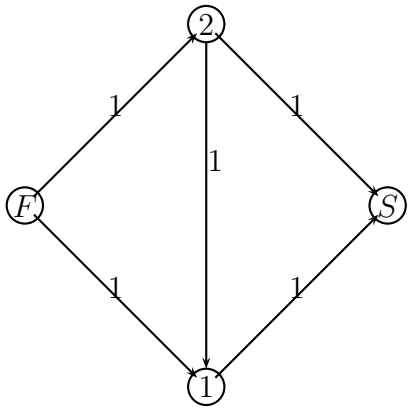


Figura 16: Red capacitada del Ejemplo 16.

El problema en el ejemplo anterior radica en que en el primer caso hemos escogido un camino que no es el adecuado. Y con el algoritmo que habíamos propuesto inicialmente no se puede rectificar esta elección.

Vamos entonces a estudiar el problema con algo más de detenimiento. En realidad, tenemos que tener en cuenta que lo que buscamos es la zona del grafo en el que se produce el "cuello de botella", es decir, la zona en la que los arcos están saturados y no permiten que se aumente el flujo. Y una característica de esta zona es que separa la fuente del sumidero. Esto nos lleva al concepto de corte.

Definición 7. *Dada una red capacitada, un* **corte** *es una partición* $\{\mathcal{A}, \mathcal{A}^c\}$ *del conjunto de vértices de forma que* $F \in \mathcal{A}$ *y* $S \in \mathcal{A}^c$.

La idea intuitiva de corte es que los vértices de $\mathcal{A}$ son los vértices que se pueden alcanzar desde F con el flujo máximo. Por ello, S no puede estar en $\mathcal{A}$, pues esto implicaría la existencia de un camino entre F y S por arcos no saturados. En definitiva, los cortes nos dan los posibles "cuellos de botella"del grafo.

Definición 8. *Dado un corte, se define la* **capacidad** *del corte como el valor*

$$c(\{\mathcal{A}, \mathcal{A}^c\}) := \sum_{i \in \mathcal{A}, j \in \mathcal{A}^c} c_{ij}.$$

La capacidad del corte nos indica de la máxima cantidad de flujo que puede pasar a través de ese corte. Nótese entonces que el valor de un flujo compatible nunca puede superar el valor de la capacidad de un corte cualquiera. Es decir,

$$\min_{\{\mathcal{A}, \mathcal{A}^c\}} c(\{\mathcal{A}, \mathcal{A}^c\}) \geq \max_{f} v_f.$$

Como consecuencia directa de este resultado, si encontramos un flujo y un corte tales que el valor del flujo coincida con la capacidad del corte, se tiene que el valor del flujo es máximo y la capacidad del corte es mínima.

Definición 9. *Dado un flujo f y un corte $\{\mathcal{A}, \mathcal{A}^c\}$, se define el* **flujo neto a través del corte** *como*

$$f(\{\mathcal{A}, \mathcal{A}^c\}) - f(\{\mathcal{A}^c, \mathcal{A}\}) := \sum_{i \in \mathcal{A}, j \in \mathcal{A}^c} f_{ij} - \sum_{i \in \mathcal{A}^c, j \in \mathcal{A}} f_{ij}.$$

El valor del flujo neto es la cantidad que realmente está pasando por ese corte.

Ejemplo 17. *Consideremos el ejemplo inicial de esta sección y el flujo dado por los valores de la Figura 17.*

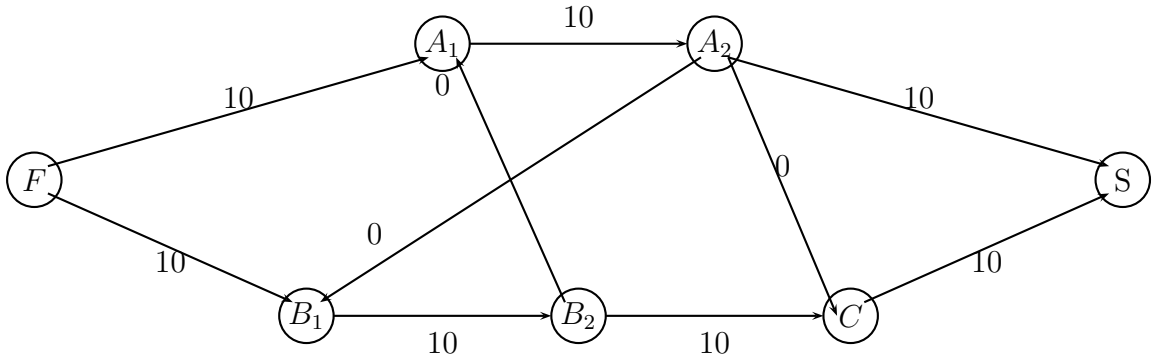


Figura 17: Ejemplo de flujo factible del Ejemplo 17

En este caso el valor del flujo es 20, que es la cantidad que sale desde la fuente. Consideremos ahora el corte $\{F, A_1, A_2\}, \{B_1, B_2, C, S\}$. Entonces, $c(\{\mathcal{A}, \mathcal{A}^c\}) = 16 + 12 + 10 + 5 = 43$ y $f(\{\mathcal{A}, \mathcal{A}^c\}) - f(\{\mathcal{A}^c, \mathcal{A}\}) = 10 + 10 = 20$.

Como era de esperar, el flujo neto a través del corte coincide con el valor del flujo. Este resultado es cierto siempre.

Lema 2. *Dado un flujo f y un corte $\{\mathcal{A}, \mathcal{A}^c\}$, se tiene que*

$$v_f = f(\{\mathcal{A}, \mathcal{A}^c\}) - f(\{\mathcal{A}^c, \mathcal{A}\}).$$

Demostración: Sea f un flujo y $\{\mathcal{A}, \mathcal{A}^c\}$. Entonces,

$$\begin{aligned}
v_f &= \sum_{i \in \Gamma(F)} f_{Fi} \\
&= \sum_{i \in \mathcal{A}} \left[\sum_{j \in \Gamma(i)} f_{ij} - \sum_{j \in \Gamma^-(i)} f_{ji} \right] \\
&= \sum_{i \in \mathcal{A}} \left[\sum_{j \in \Gamma(i) \cap \mathcal{A}^c} f_{ij} - \sum_{j \in \Gamma^-(i) \cap \mathcal{A}^c} f_{ji} \right] + \sum_{i \in \mathcal{A}} \left[\sum_{j \in \Gamma(i) \cap \mathcal{A}} f_{ij} - \sum_{j \in \Gamma^-(i) \cap \mathcal{A}} f_{ji} \right] \\
&= f(\{\mathcal{A}, \mathcal{A}^c\}) - f(\{\mathcal{A}^c, \mathcal{A}\}) + 0
\end{aligned}$$

Con lo que queda demostrado el resultado. ■

Por lo tanto, un flujo f será máximo si encontramos un corte $\{\mathcal{A}, \mathcal{A}^c\}$ de forma que el flujo neto a través del corte coincida con la capacidad del corte. Para que esto suceda, a la vista de la definición de flujo neto y capacidad del corte, necesitamos que se cumplan las dos condiciones siguientes:

- Dados $i \in \mathcal{A}, j \in \mathcal{A}^c$, se tiene que $f_{ij} = c_{ij}$.
- Dados $i \in \mathcal{A}^c, j \in \mathcal{A}$, se tiene que $f_{ij} = 0$.

El resultado en el que se basa el algoritmo de Ford-Fulkerson es el siguiente:

Teorema 3. *Sea f un flujo máximo y consideremos el conjunto $\mathcal{A}$ definido por:*

- $F \in \mathcal{A}$.
- *Si $i \in \mathcal{A}$ y $f_{ij} < c_{ij}$, entonces $j \in \mathcal{A}$.*
- *Si $j \in \mathcal{A}$ y $f_{ij} > 0$, entonces $i \in \mathcal{A}$.*

Entonces $\{\mathcal{A}, \mathcal{A}^c\}$ es un corte.

Demostración: Supongamos que $\{\mathcal{A}, \mathcal{A}^c\}$ no es un corte. Esto implica que $S \in \mathcal{A}$. Entonces, se puede llegar desde F a S de la forma

$$F \to U_1 \leftarrow U_2 \to U_3 ... U_r \to S.$$

Sea $\epsilon := \text{mín}\{c_{FU_1} - f_{FU_1}, f_{U_1U_2}, c_{U_2U_3} - f_{U_2U_3}, ..., c_{U_rS} - f_{U_rS}\} > 0$.
Consideremos entonces el siguiente flujo:

$$g_{ij} := \begin{cases} f_{U_iU_j} + \epsilon & \text{si U}_\text{i} \to \text{U}_\text{j} \\ f_{U_iU_j} - \epsilon & \text{si U}_\text{i} \leftarrow \text{U}_\text{j} \\ f_{U_iU_j} & \text{en otro caso} \end{cases}$$

Entonces g es un flujo factible que tiene de valor $v_f + \epsilon$, lo que contradice que f sea un flujo óptimo. ■

Además, para este flujo óptimo y para el corte construido en el teorema anterior, si $i \in \mathcal{A}, j \in \mathcal{A}^c$ se tiene que

- Si $(i, j) \in A$, entonces $f_{ij} = c_{ij}$.
- Si $(j, i) \in A$, entonces $f_{ji} = 0$.

Este resultado implica que el valor del flujo máximo coincide con la capacidad del corte mínimo.

Corolario 1.

$$\text{máx}\, v_f = \text{mín}\, c(\mathcal{A}, \mathcal{A}^c).$$

Veamos ahora cómo calcular un flujo en las condiciones anteriores.

Definición 10. *Sea una red capacitada y un flujo f a través de esta red. El* **grafo incremental** *es una red con capacidades definidas de la siguiente manera:*

- *Si $f_{ij} < c_{ij}$, entonces existe el arco entre i y j con capacidad $c_{ij} - f_{ij}$.*
- *Si $f_{ij} > 0$, entonces existe el arco entre j e i con capacidad f_{ij}.*

La primera condición nos indica que todavía es posible pasar flujo a través del arco. La segunda condición nos permite rectificar si hemos asignado un flujo a un arco y queremos cambiar la asignación.

Corolario 2. *Un flujo f es máximo si y sólo si no existe un camino entre F y S en el correspondiente grafo incremental.*

Veamos finalmente el algoritmo de Ford-Fulkerson para hallar el flujo máximo:

RESUMEN DEL ALGORITMO DE FORD-FULKERSON

- **INICIALIZACIÓN**: Se inicializa el flujo a $f_{ij} = 0, \forall (i,j) \in V$. Entonces el grafo incremental coincide con el grafo inicial.
- **PASO ITERATIVO**: Mientras sea posible
  - Determinar un camino entre F y S en el grafo incremental.
  - Aumentar el flujo la máxima cantidad posible en los arcos del camino, es decir, el mínimo de las capacidades de los arcos del camino.
  - Construir el grafo incremental para el nuevo flujo.

Ejemplo 18. *Consideremos el ejemplo con el que iniciamos esta sección partiendo del flujo nulo, tal y como aparece en la Figura 18. Aplicaremos el algoritmo anterior.*

Si consideramos el camino $F \to A_1 \to A_2 \to S$, entonces el valor del flujo pasa a ser 12 y el grafo incremental sería el que aparece en la Figura 19.

Consideremos ahora el camino $F \to B_1 \to B_2 \to C \to S$. Entonces el valor del flujo pasa a ser 22 y el grafo incremental sería el dado en la Figura 20.

Cogemos ahora el camino $F \to B_1 \to B_2 \to A_1 \to A_2 \to C \to S$. Entonces el valor del flujo pasa a ser 24 y el grafo incremental pasa a ser el dado en la Figura 21.

Finalmente, cogemos el camino $F \to A_1 \to A_2 \to C \to S$. Entonces el valor del flujo pasa a ser 27 y el grafo incremental pasa a ser el dado en la Figura 22.

Como ahora ya no tenemos ningún camino en el grafo incremental entre F y S, el algoritmo termina. El flujo final viene dado en la Figura 23, que se obtiene comparando el grafo incremental final con la red capacitada inicial.

El corte de capacidad mínima es $\{F, B_1\}, \{A_1, A_2, B_2, C, S\}$.

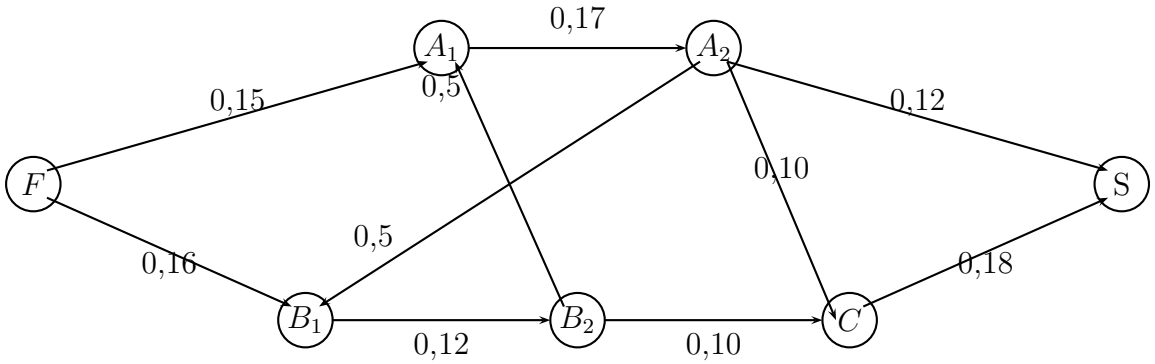


Figura 18: Iteración 0.

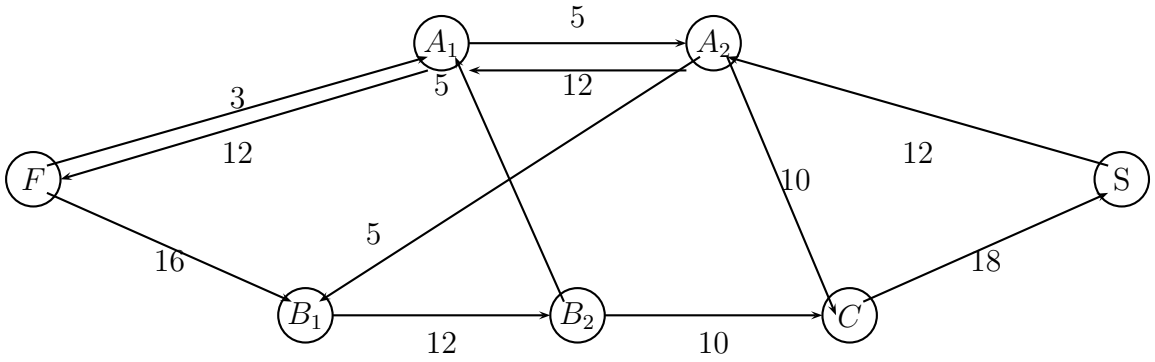


Figura 19: Iteración 1.

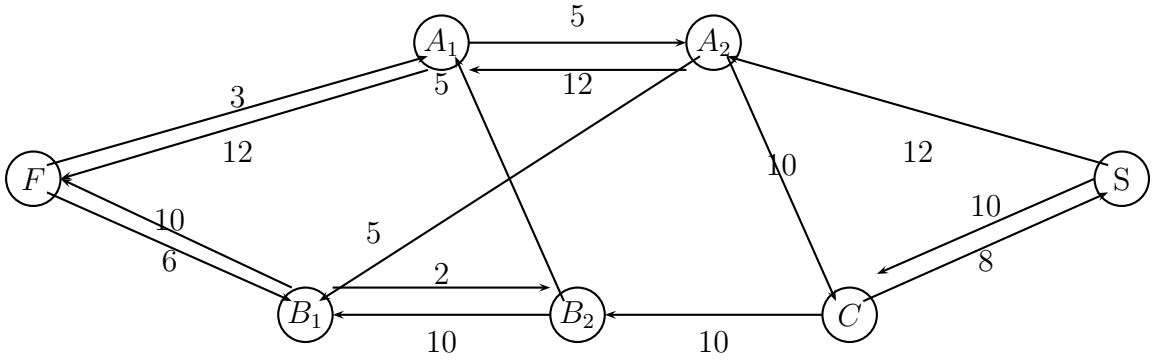


Figura 20: Iteración 2.

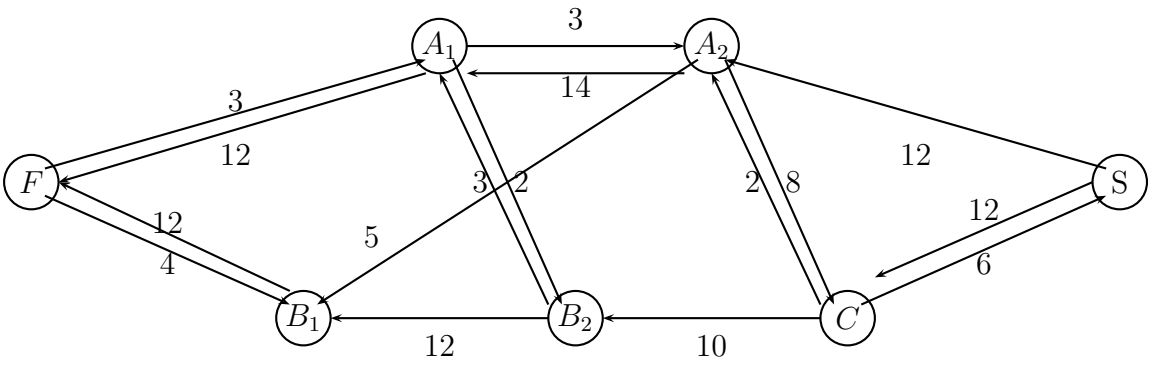


Figura 21: Iteración 3.

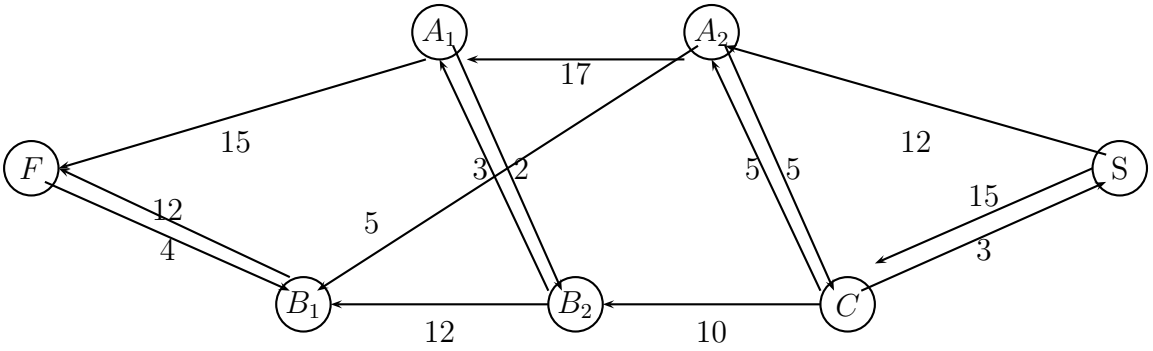


Figura 22: Iteración 4

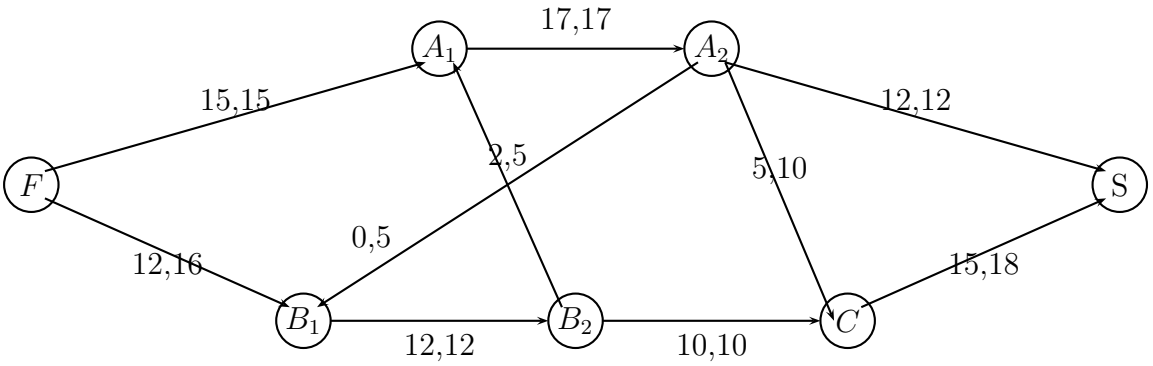


Figura 23: Flujo máximo.