Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 13

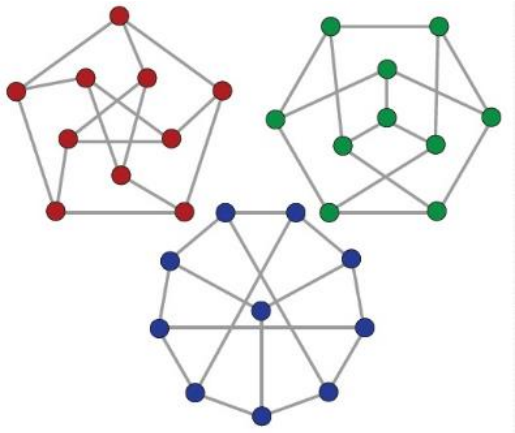
Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T3. Tipos Abstractos de Datos (TAD)

- **Grafos.**
  - Árbol de Recubrimiento o Expansión
  - Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM
  - Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL
  - Cierre Transitivo: ALGORITMO DE WARSHALL
  - Caminos Mínimos: ALGORITMO DE FLOYD
  - Caminos Mínimos: ALGORITMO DE DIJKSTRA



UNIVERSIDAD NEBRIJA

ÍNDICE

Árbol de Recubrimiento o Expansión

- Sea G (V, E) un Grafo Conexo: $E(G) = T(G) \cup R(G)$

  donde: $T(G)$ = {arcos de un recorrido del grafo}

  $R(G)$ = {arcos del grafo pero no del recorrido }

- Los vértices V(G) junto con los arcos de T(G), constituyen un ÁRBOL:

Árbol de Recubrimiento/Expansión del Grafo G *(Spanning Tree)*

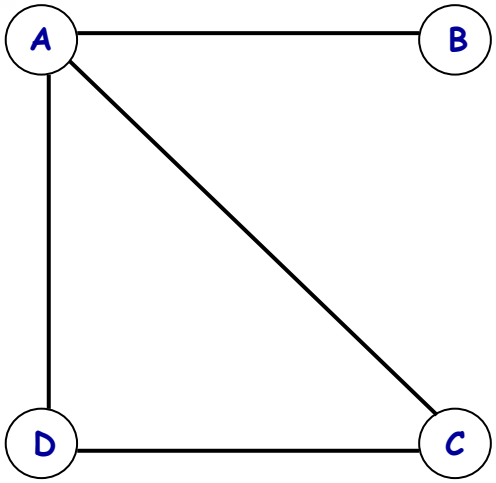
- Un árbol de recubrimiento es un subgrafo, G', de G tal que:

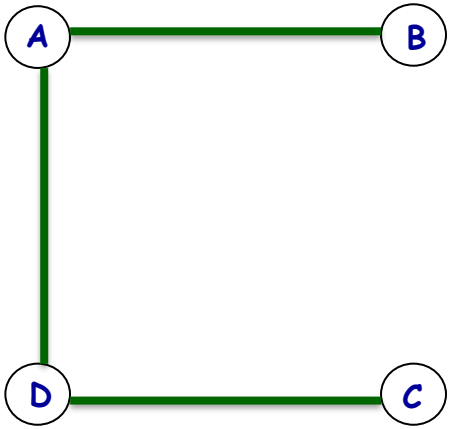
  V(G') = V(G) E(G') = T(G) y G' es conexo (sin ciclos).

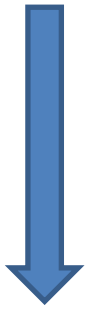
- Todos los **grafos conexos** con **N vértices** y **(N-1) arcos** son **ÁRBOLES**.

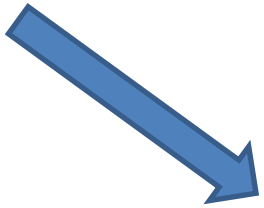
UNIVERSIDAD NEBRIJA

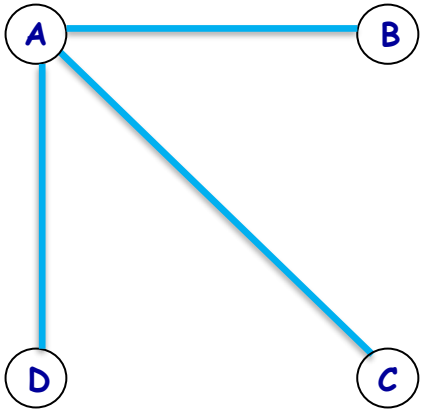
Árbol de Recubrimiento o Expansión

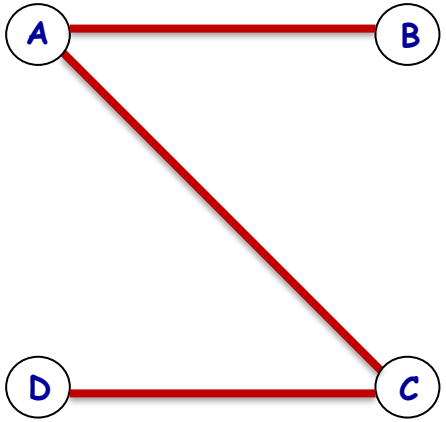






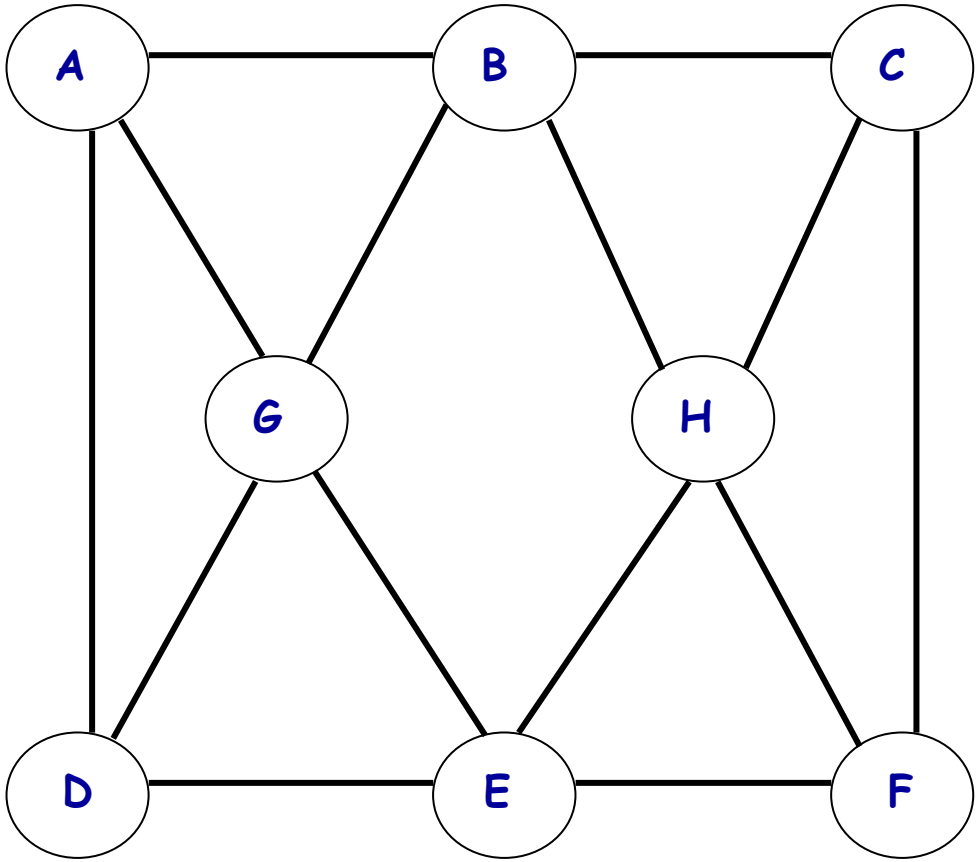






Supongamos el siguiente grafo conexo:

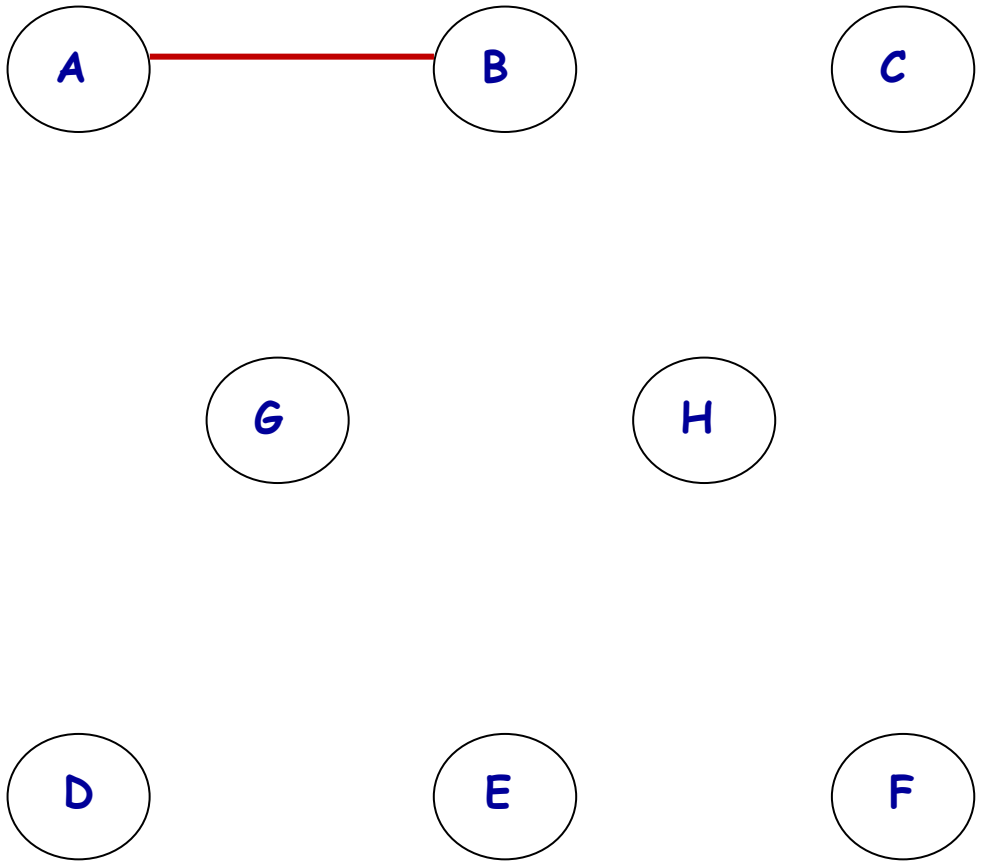
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

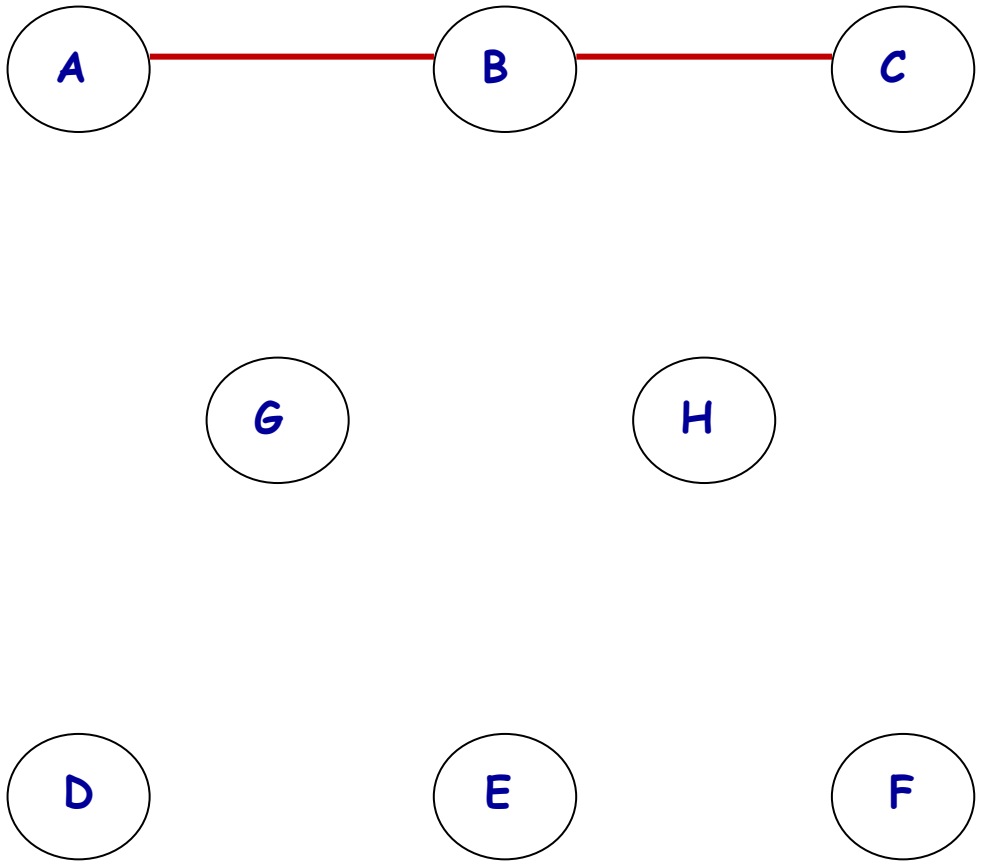
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

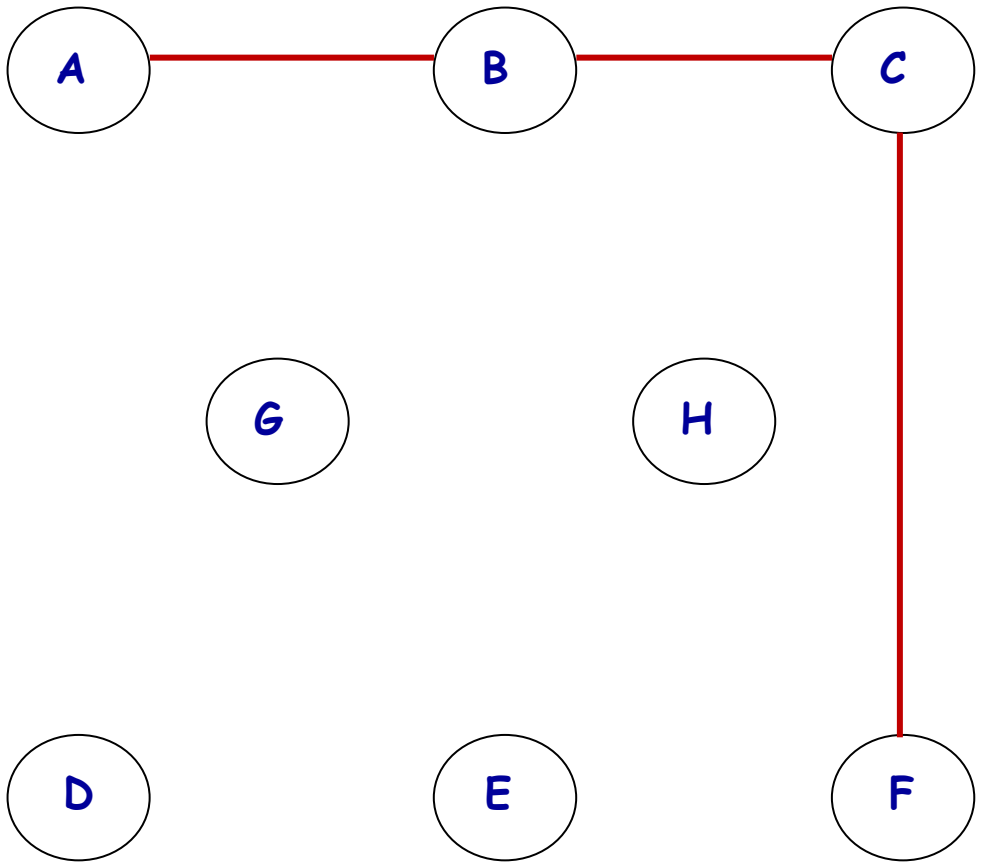
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

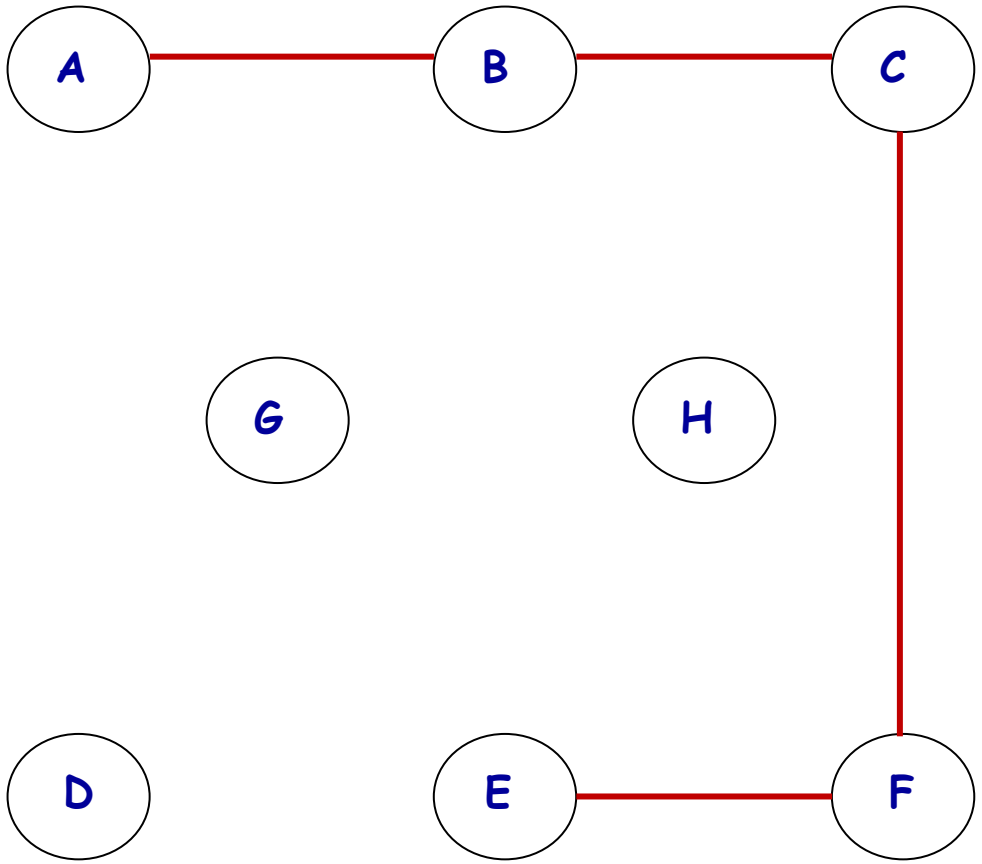
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

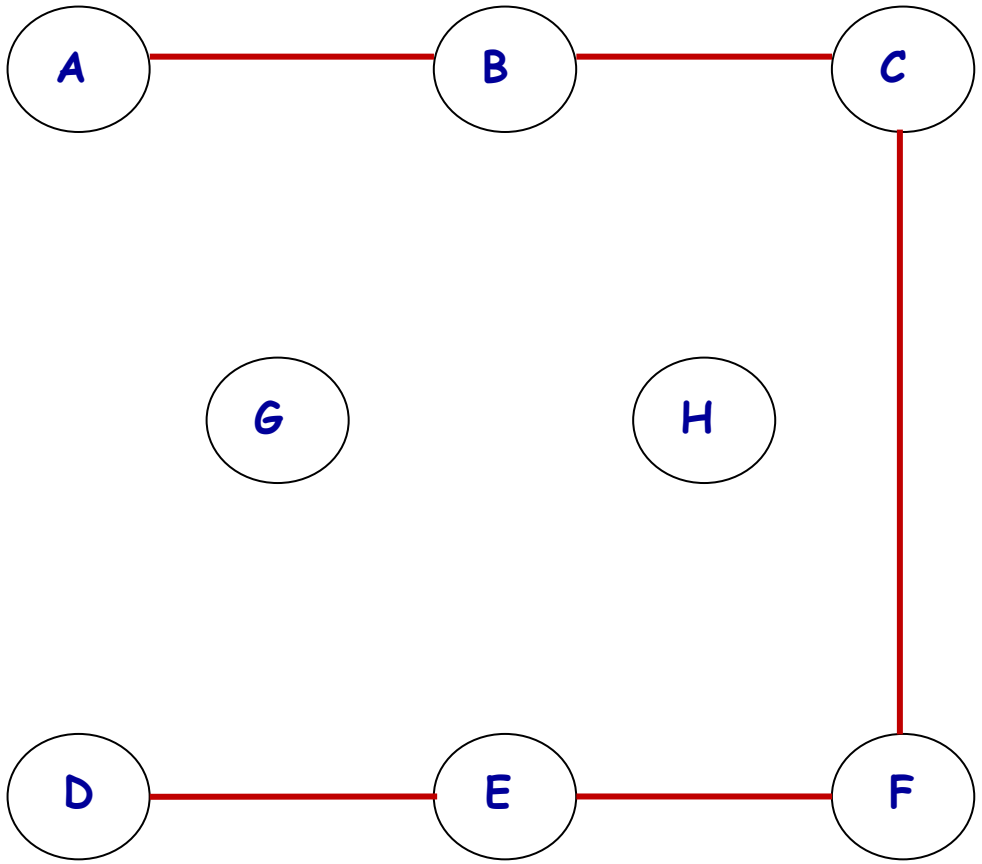
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

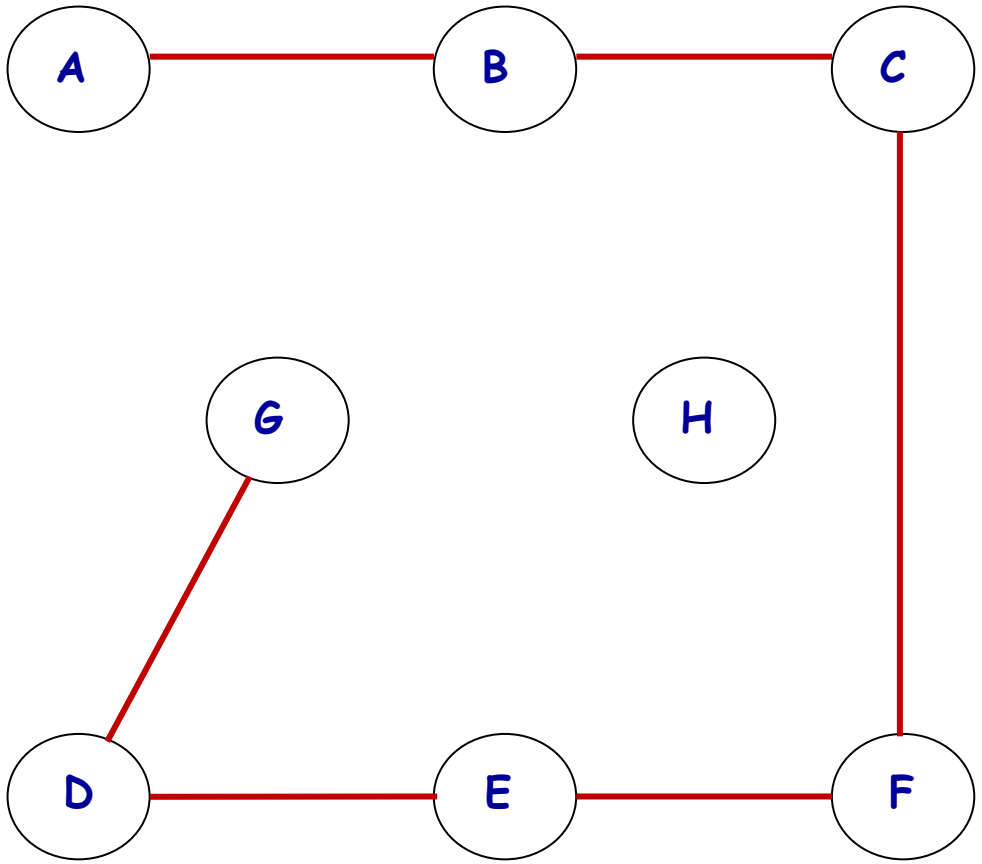
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

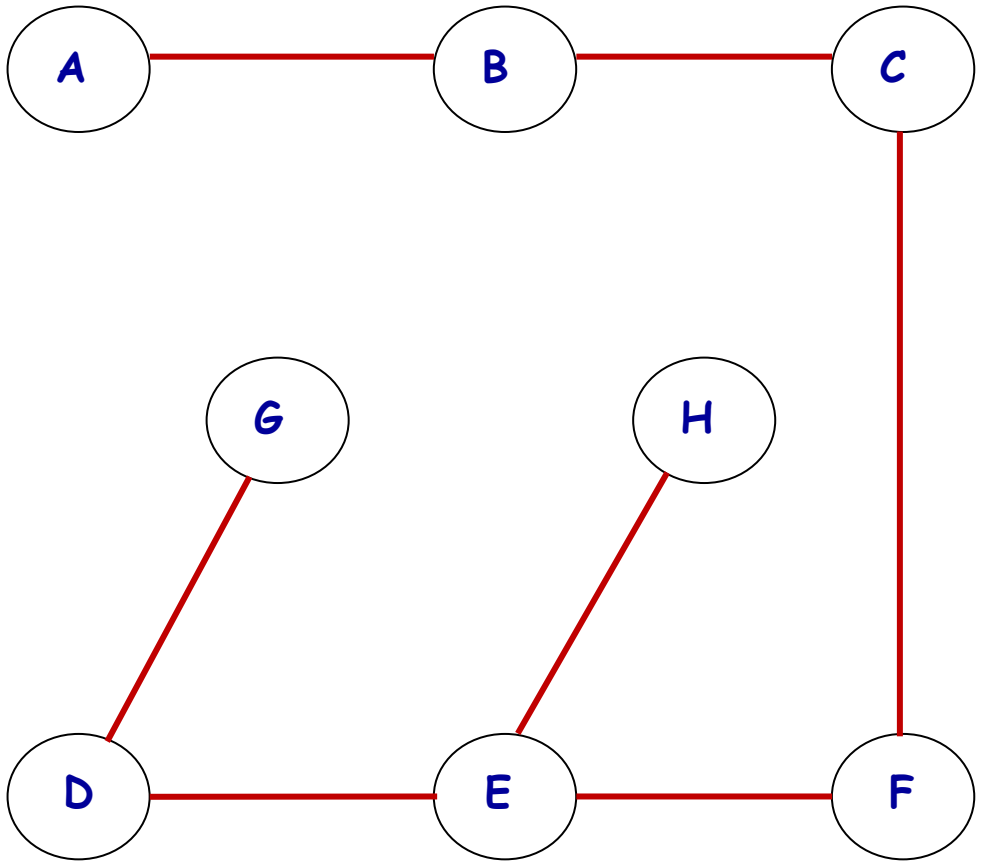
A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento o Expansión

Supongamos el siguiente grafo conexo:

A → B → D → G
↓
B → A → C → G → H
↓
C → B → F → H
↓
D → A → E → G
↓
E → D → F → G → H
↓
F → C → E → H
↓
G → A → B → D → E
↓
H → B → C → E → F



Árbol de Recubrimiento Mínimo

Sea **G es un grafo valorado y conexo**.

Su **árbol de recubrimiento mínimo** es un **árbol de recubrimiento que cumple que la suma de las etiquetas de sus aristas es la menor posible**.

Se obtiene a través de los algoritmos: **Kruskal** y **Prim**

Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM

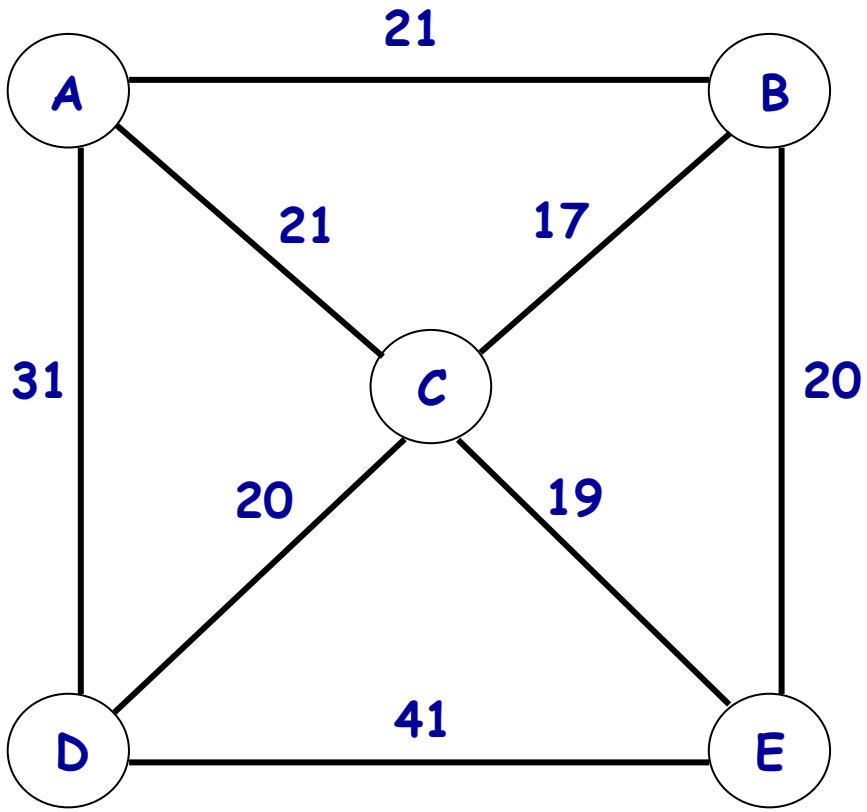
Consiste en **añadir, en cada paso, una arista de peso mínimo** a un **árbol** previamente construido y siempre haciendo que el **subgrafo** que se va formando sea **conexo**.

Pasos:

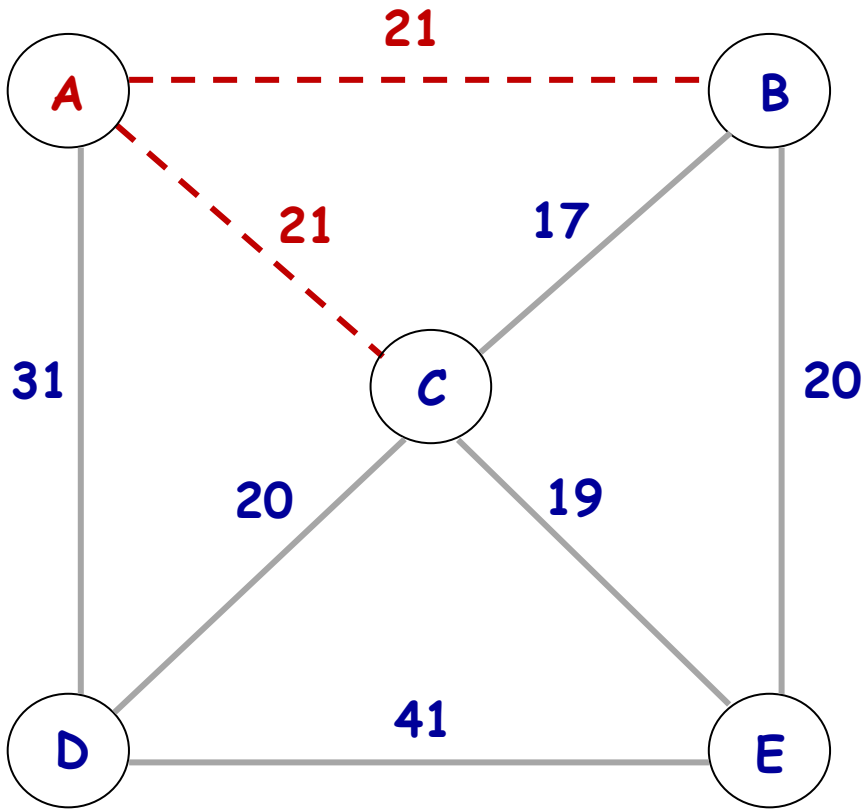
1. Empezar en **un vértice cualquiera v**. El árbol consta inicialmente sólo del **nodo v**.
2. Del resto de vértices, buscar el que esté **más próximo a v** (la arista (v, w)) de **coste mínimo**. Añadir w y la arista (v, w) al árbol.
3. Buscar el **vértice más próximo a cualquiera de estos dos**. Añadir ese vértice y la arista al árbol de recubrimiento.
4. **Repetir sucesivamente hasta añadir los n vértices y (n-1) aristas**.

Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM

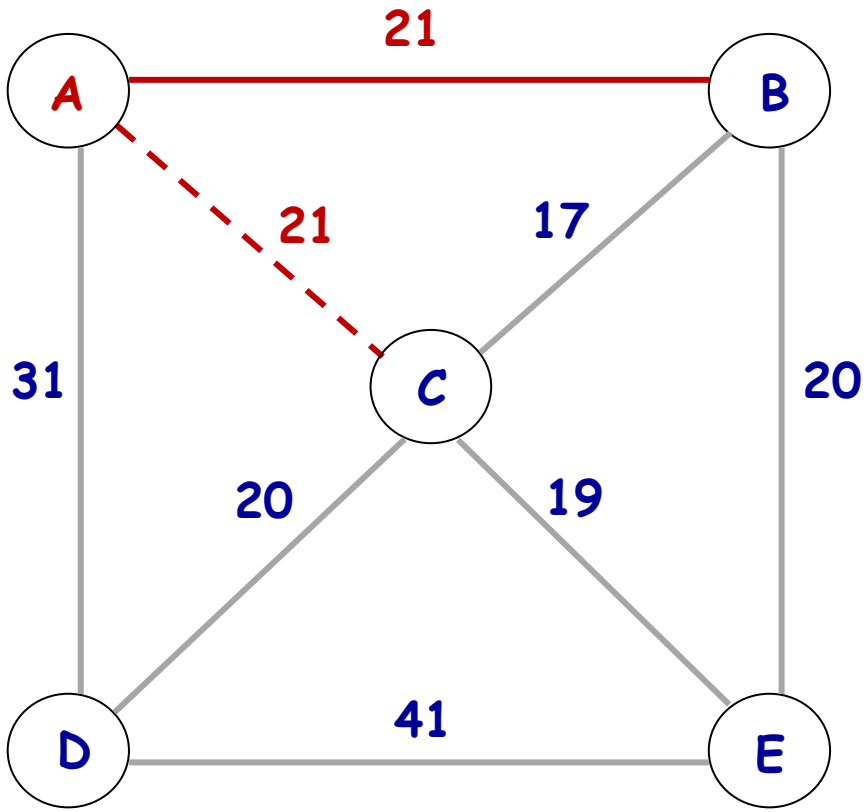
Supongamos el siguiente **grafo conexo valorado**



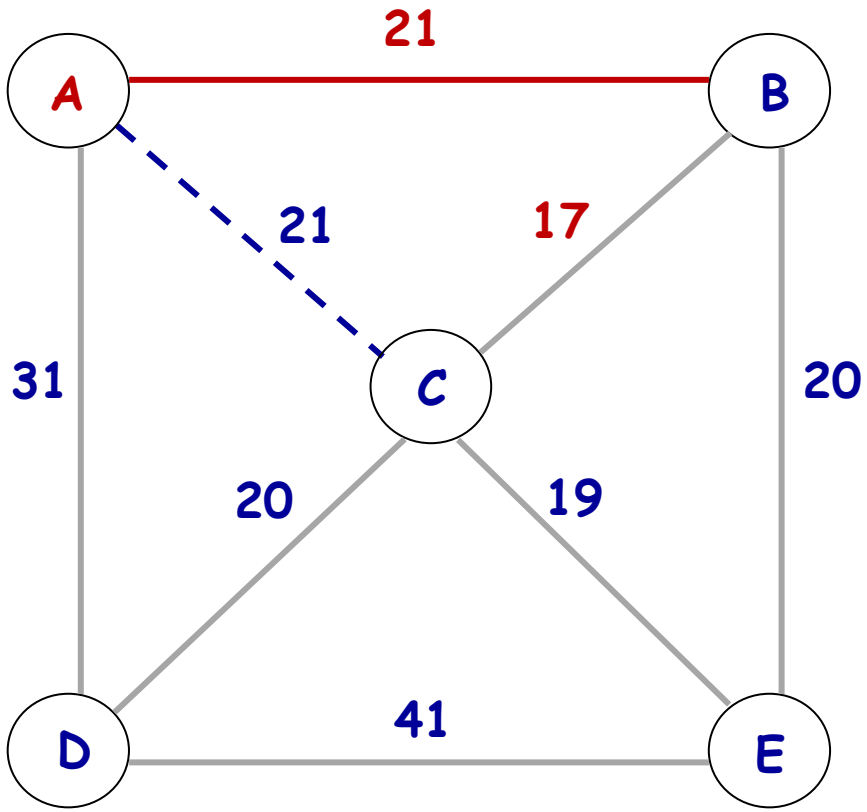
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



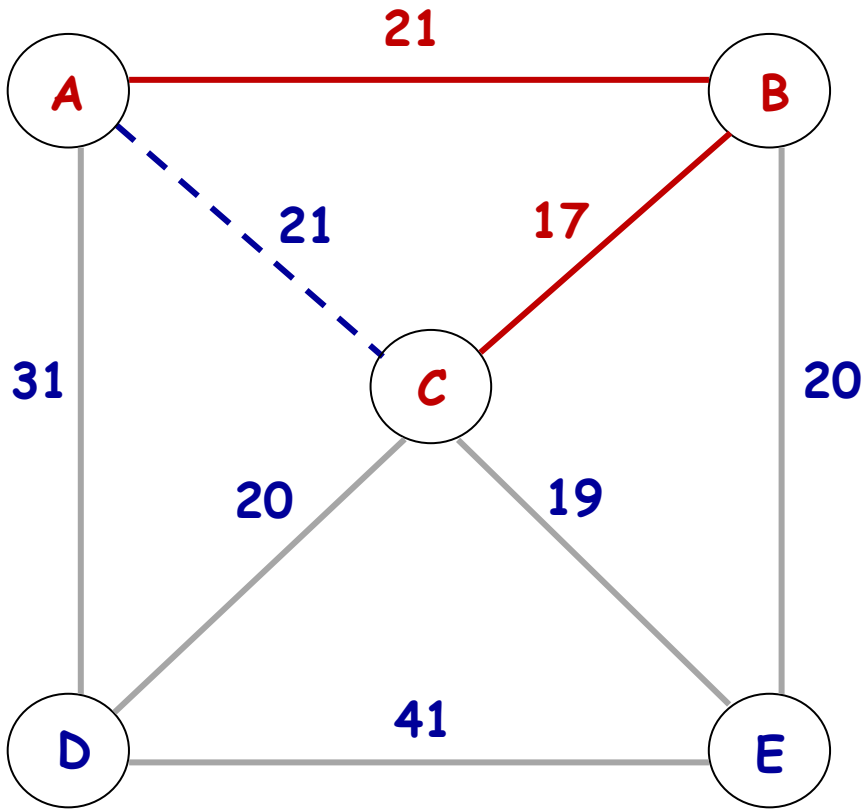
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



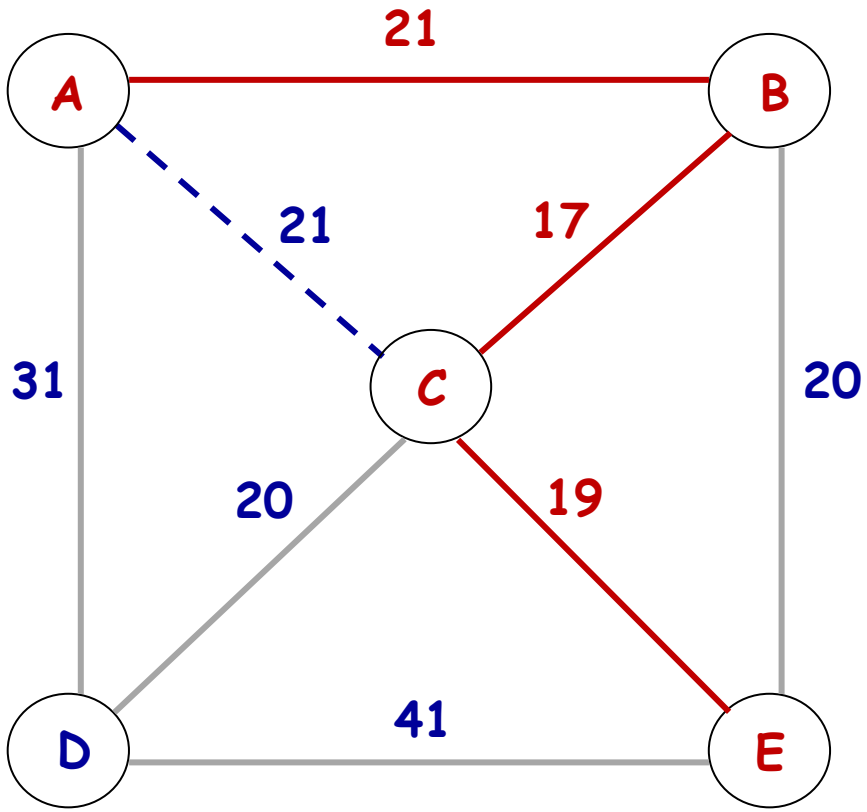
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



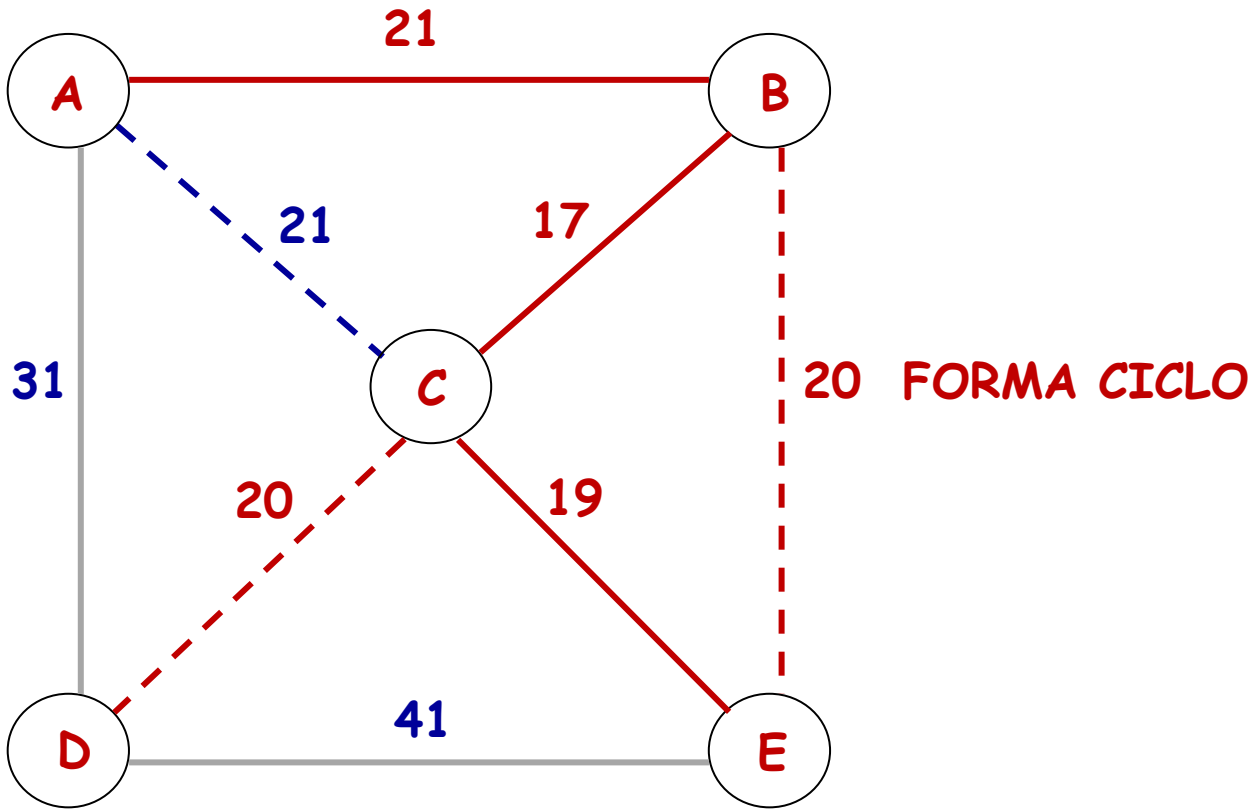
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



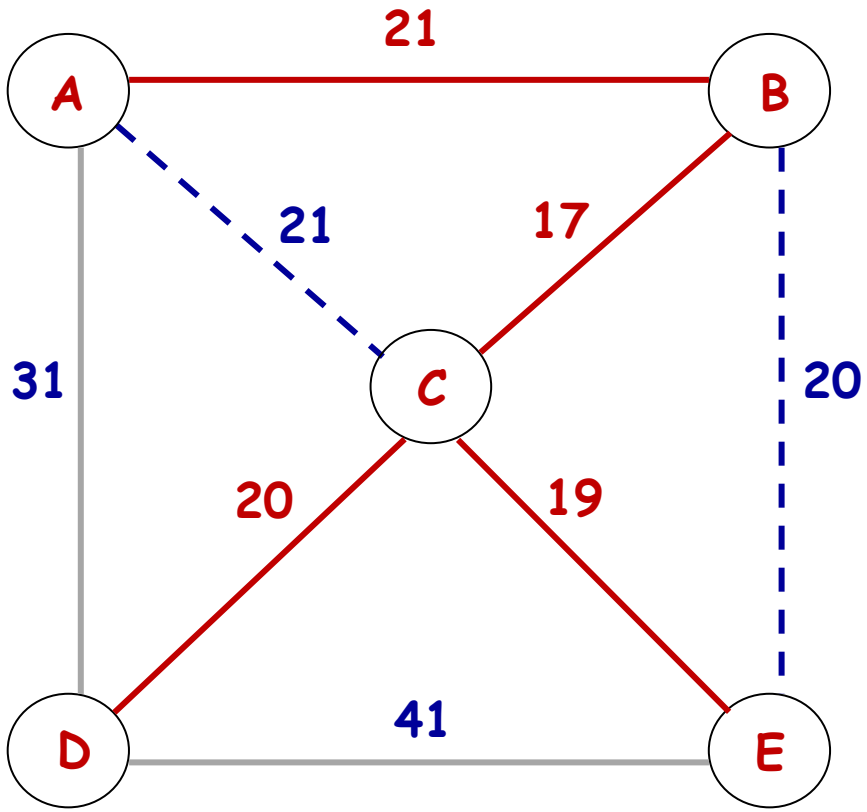
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



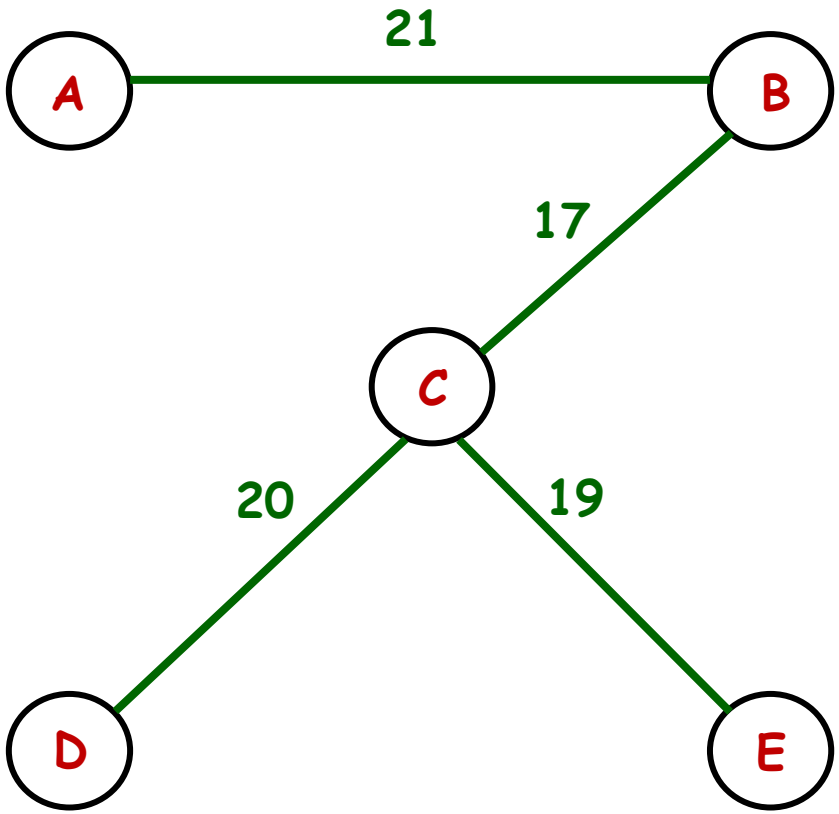
Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM



Árbol de Recubrimiento Mínimo: ALGORITMO DE PRIM

Coste total = 21 + 17 + 19 + 20 = 77

5 vértices y 4 aristas



Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL

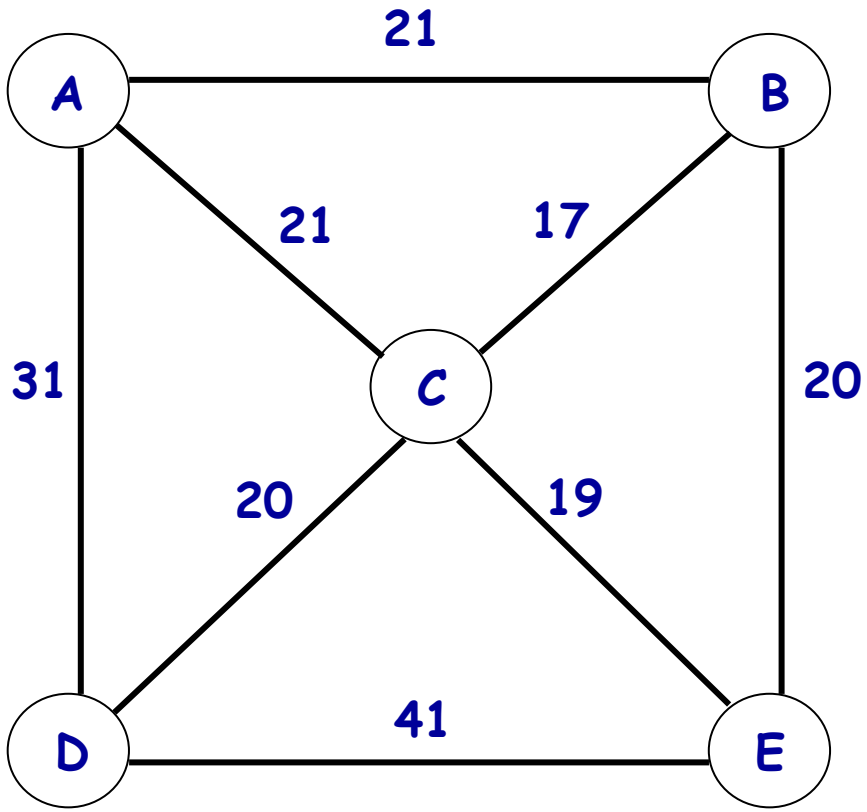
Consiste en elegir las **aristas de menor peso que no forman ciclos**. Para poder elegir dichas aristas es necesario **ordenarlas de menor a mayor peso**.

Pasos:

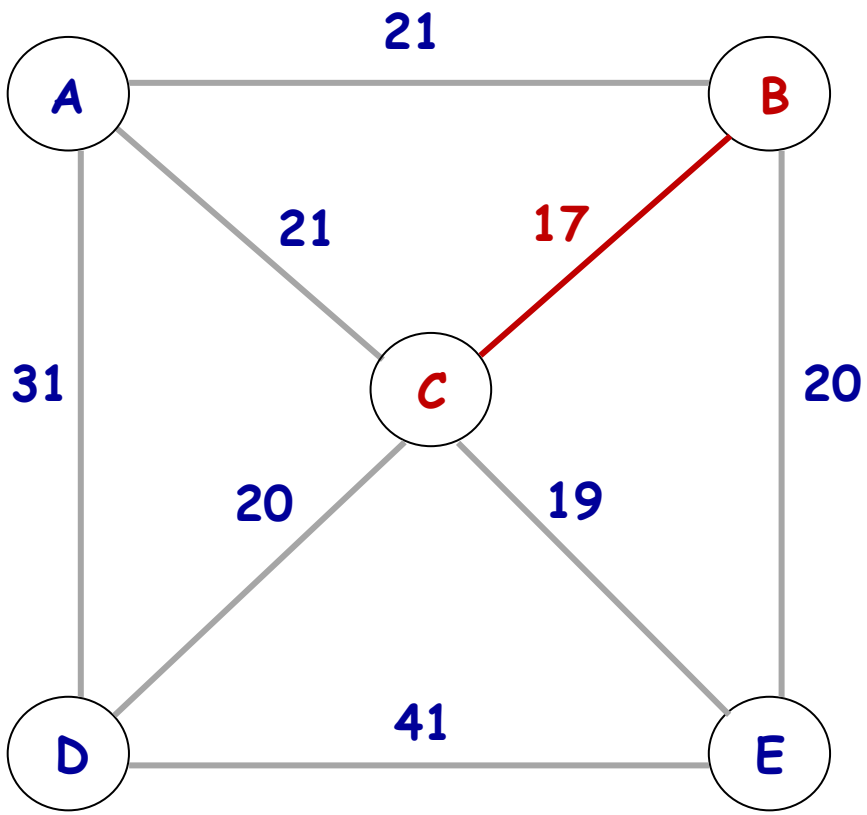
1. Se **marca la arista con menor valor**. Si hay más de una, se elige cualquiera de ellas.
2. De las aristas restantes, se **marca la que tenga menor valor**, si hay más de una, se elige cualquiera de ellas.
3. **Repetir el paso 2 siempre que la arista elegida no forme un ciclo** con las ya marcadas.
4. El proceso termina cuando tenemos **todos los nodos del grafo y (n-1) arcos**, siendo n el número de nodos del grafo.

Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL

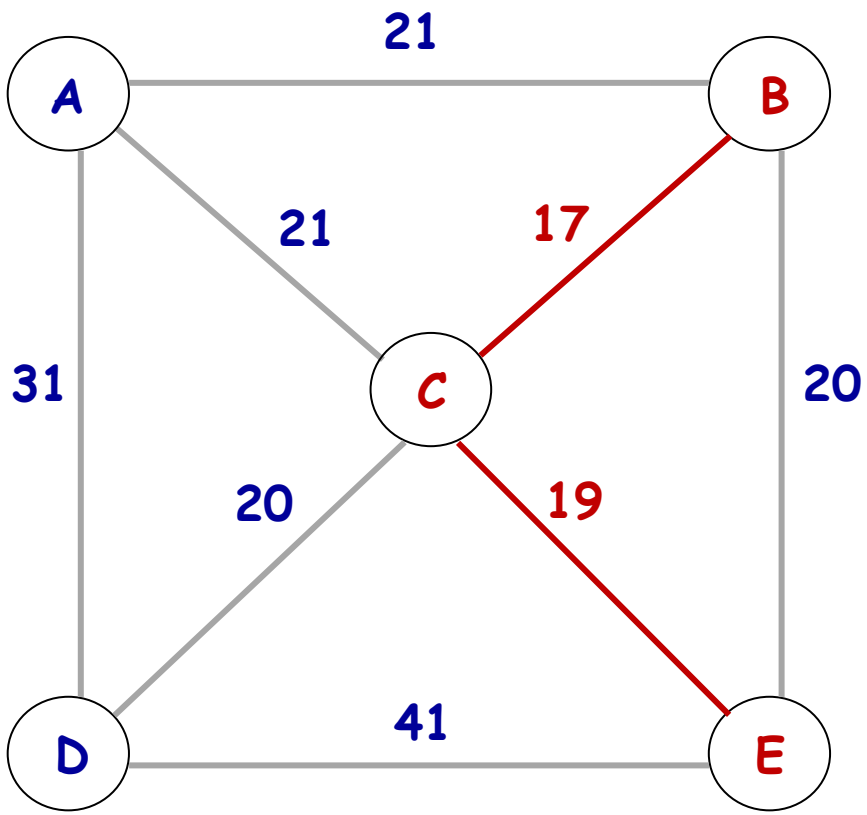
Supongamos el siguiente **grafo conexo valorado**



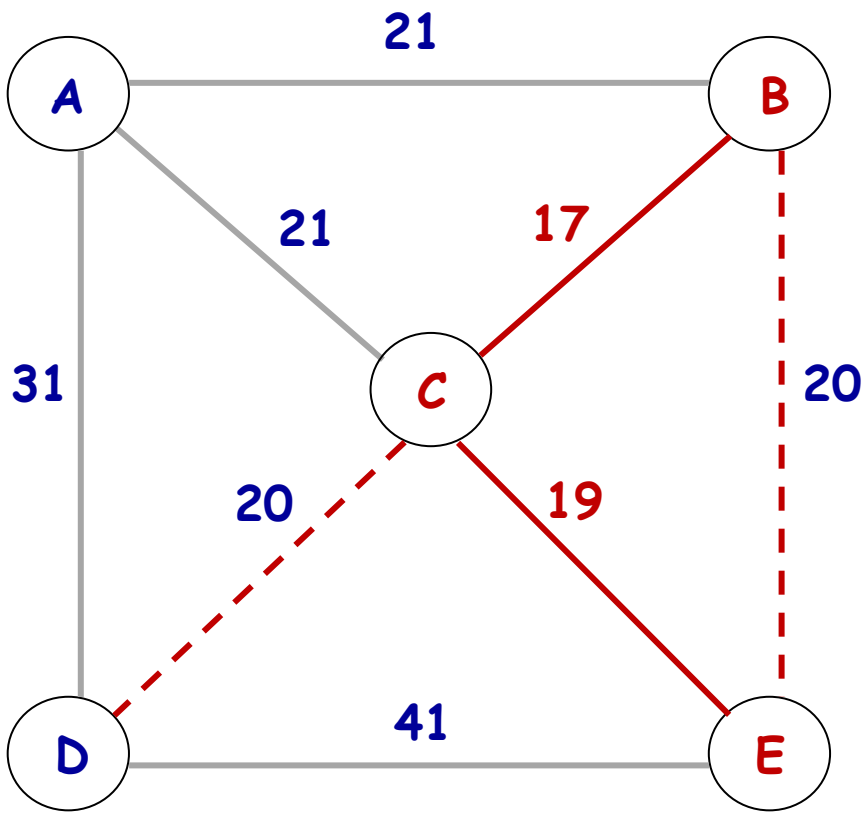
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



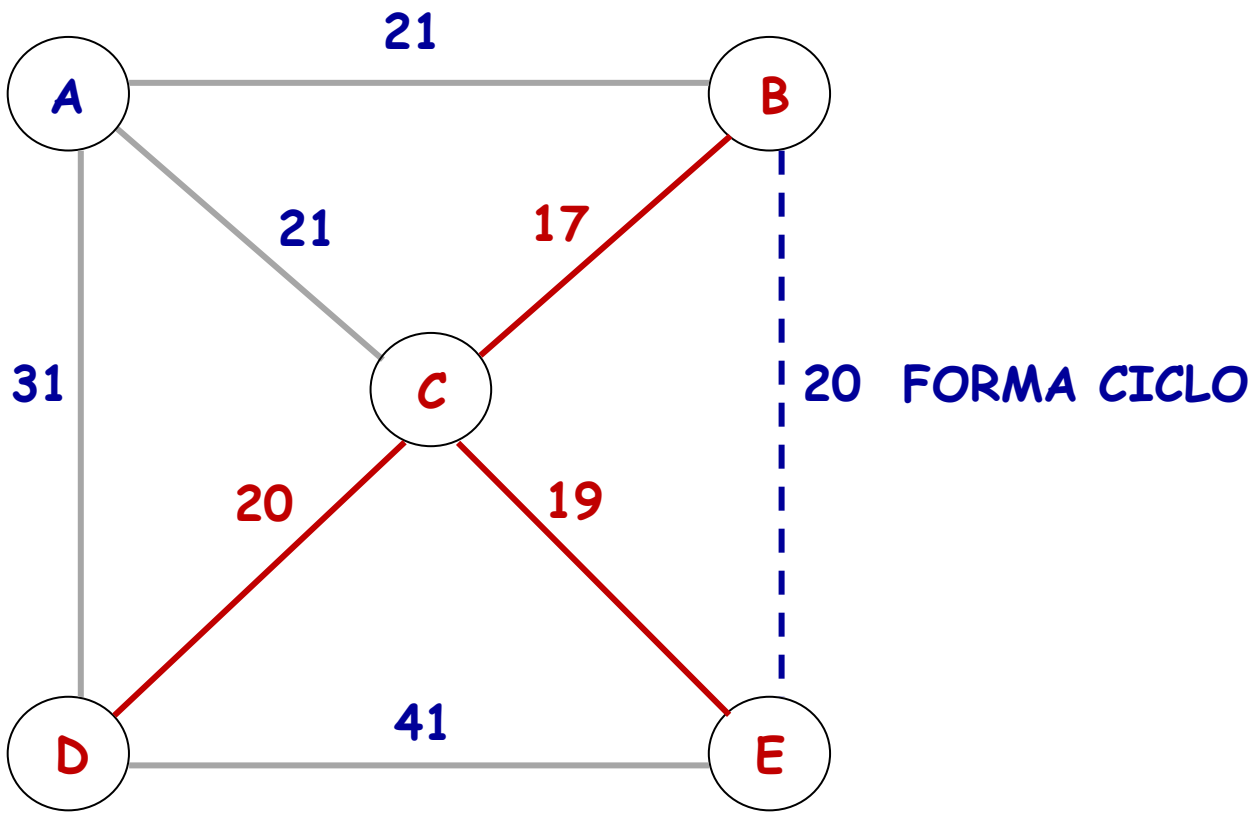
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



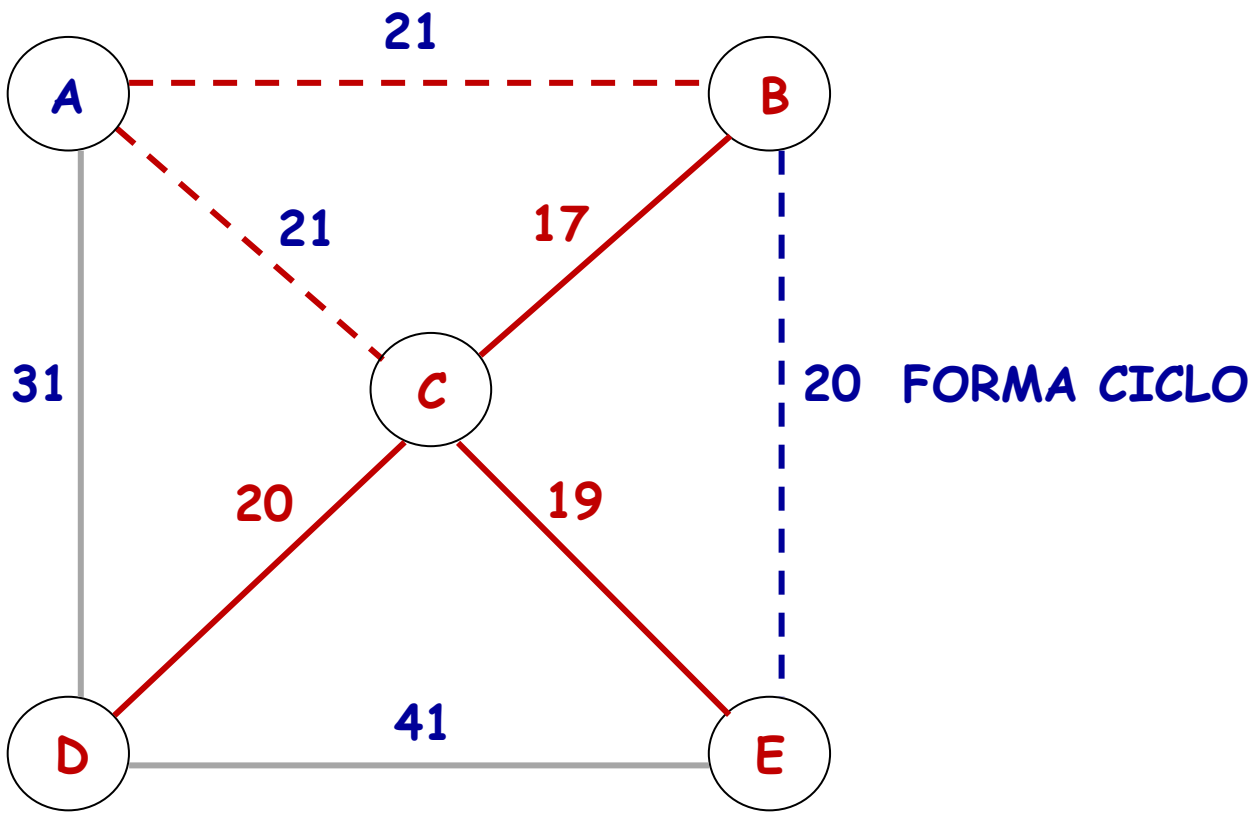
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



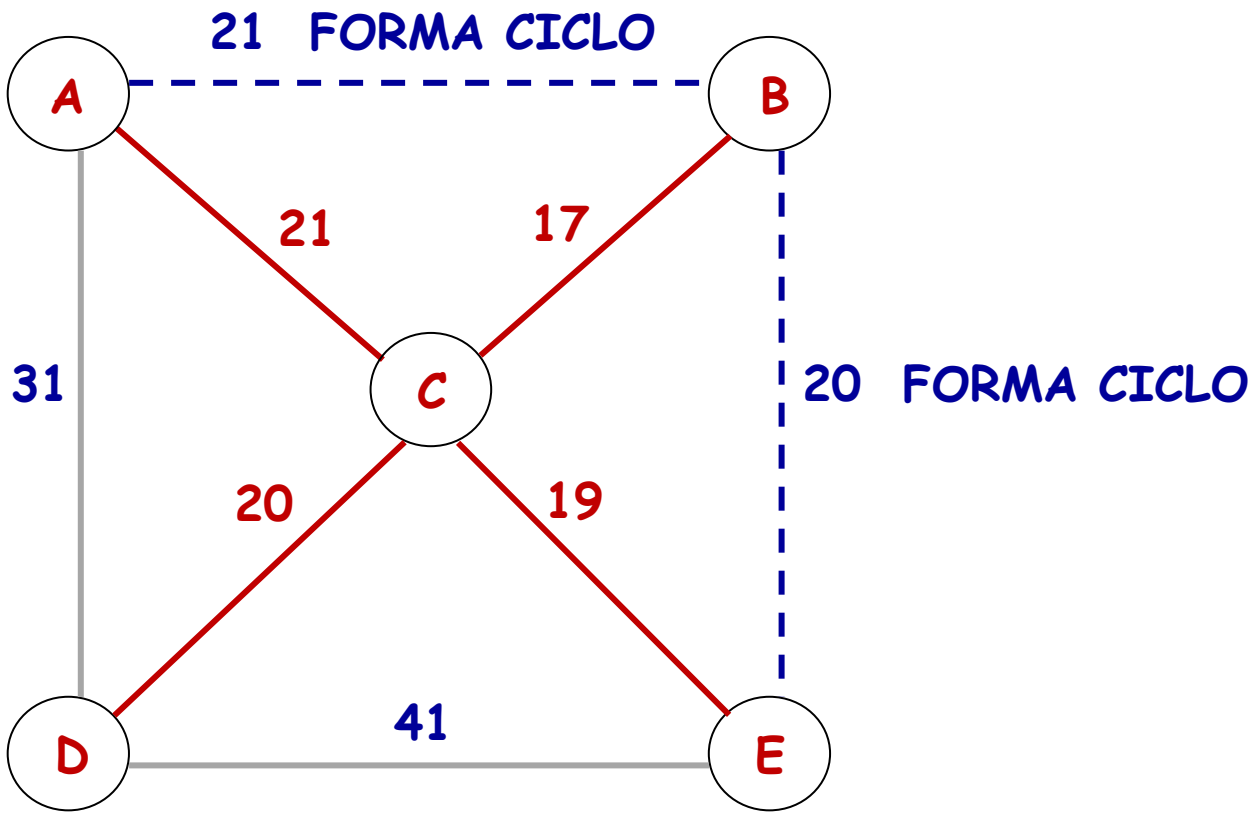
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



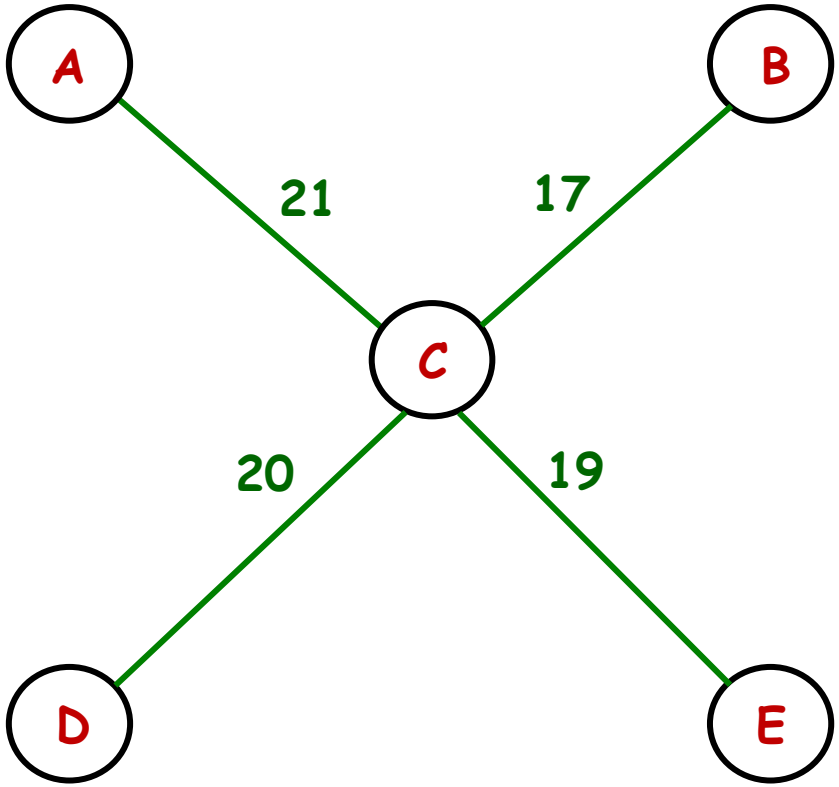
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL



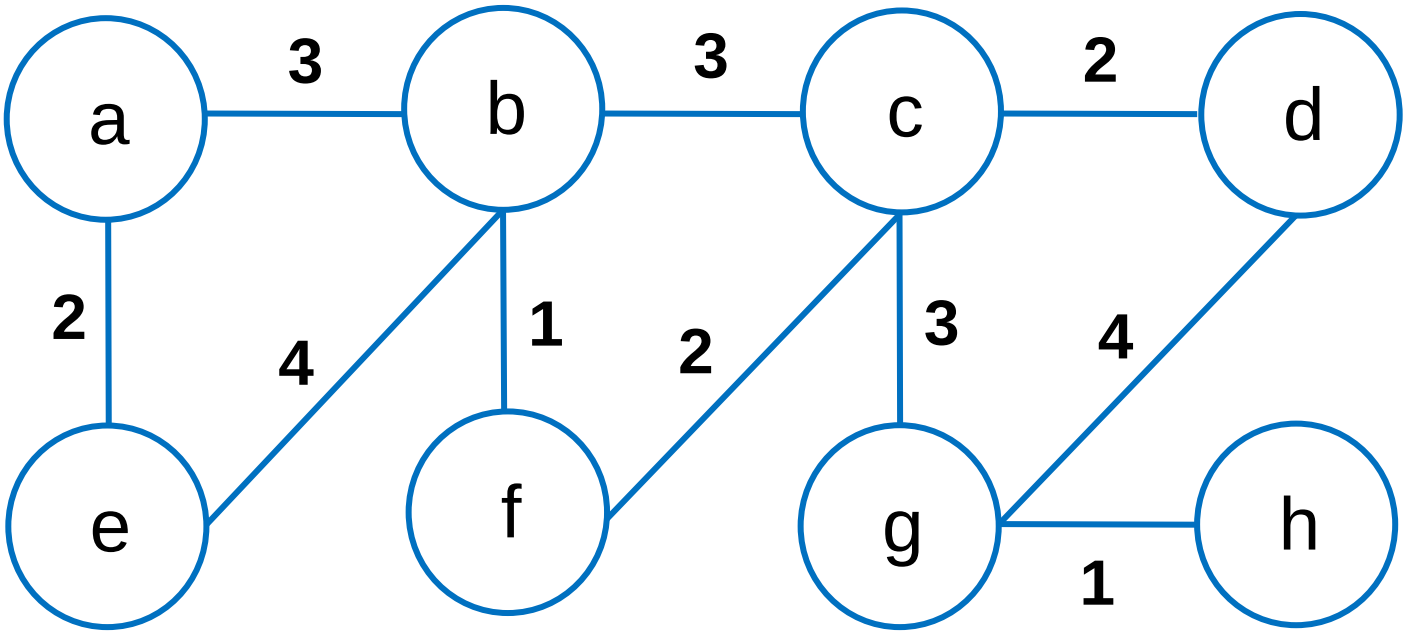
Árbol de Recubrimiento Mínimo: ALGORITMO DE KRUSKAL

Coste total = 21 + 17 + 19 + 20 = 77

5 vértices y 4 aristas



Árbol de Recubrimiento Mínimo: Algoritmos de Prim y Kruskal



Cierre Transitivo: ALGORITMO DE WARSHALL

El problema consiste en dados dos vértices V_i y V_j de un grafo G, **determinar si existe algún camino que conecte ambos vértices, independientemente de su coste**.

El **algoritmo comprobará si existe conexión** entre V_i y V_j si:

- Inicialmente existía, o bien
- Si dado un nodo V_k, existe conexión entre V_i y V_k y también existe conexión entre V_k y V_j.

Cierre Transitivo: ALGORITMO DE WARSHALL

1.- Construir la **Matriz de Adyacencia**:

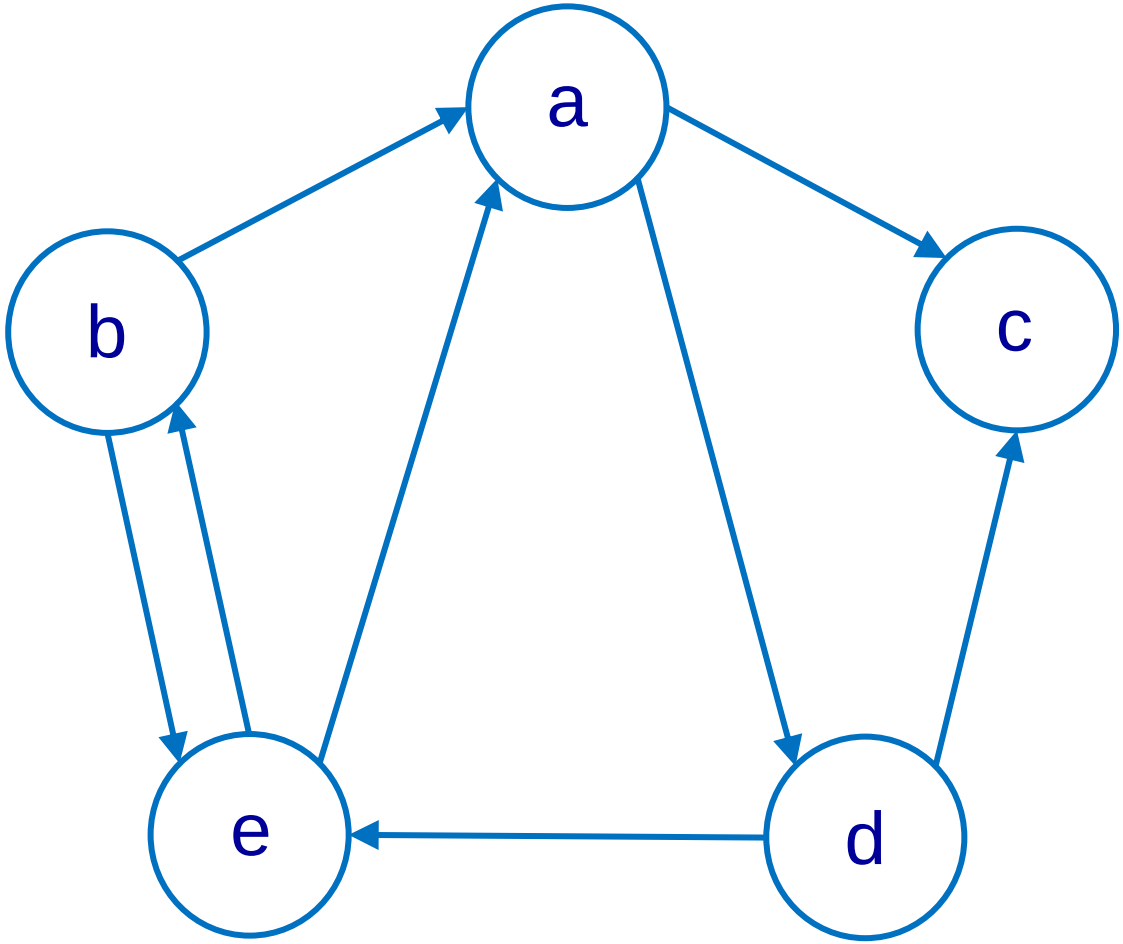
$$M(V_i, V_j) = \begin{cases} 1 & \text{si } [V_i, V_j] \in E(G) \\ 0 & \text{en caso contrario} \end{cases}$$

2.- Iterar con los diferentes vértices hasta obtener la **Matriz de Cierre Transitivo**:

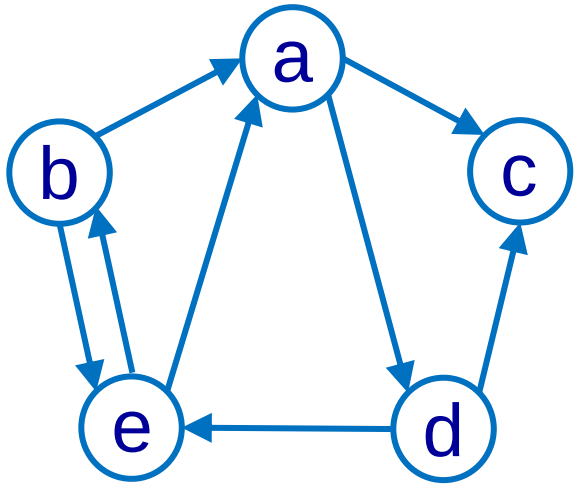
$$A[V_i, V_j] = \begin{cases} 1 & \text{si existe un camino de Vi a Vj en G} \\ 0 & \text{en caso contrario} \end{cases}$$

$$A_k[i, j] = (A_{k-1}[i, j]) \text{ OR } (A_{k-1}[i, k] \text{ AND } A_{k-1}[k, j])$$

Cierre Transitivo: ALGORITMO DE WARSHALL



Cierre Transitivo: ALGORITMO DE WARSHALL



	a	b	c	d	e
a	0	0	1	1	0
b	1	0	0	0	1
c	0	0	0	0	0
d	0	0	1	0	1
e	1	1	0	0	0

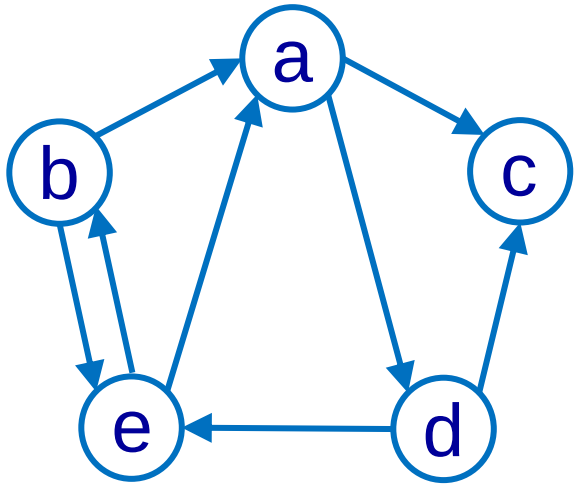
	a	b	c	d	e
a	0	0	1	1	0
b	1	0	**1**	**1**	1
c	0	0	0	0	0
d	0	0	1	0	1
e	1	1	**1**	**1**	0

Vértice intermedio: a

	a	b	c	d	e
a	0	0	1	1	0
b	1	0	**1**	**1**	1
c	0	0	0	0	0
d	0	0	1	0	1
e	1	1	**1**	**1**	0

Vértice intermedio: b

Cierre Transitivo: ALGORITMO DE WARSHALL



Vértice intermedio: c

	a	b	c	d	e
a	0	0	1	1	0
b	1	0	1	1	1
c	0	0	0	0	0
d	0	0	1	0	1
e	1	1	1	1	0

	a	b	c	d	e
a	0	0	1	1	1
b	1	0	1	1	1
c	0	0	0	0	0
d	0	0	1	0	1
e	1	1	1	1	0

Vértice intermedio: d

	a	b	c	d	e
a	0	1	1	1	1
b	1	0	1	1	1
c	0	0	0	0	0
d	1	1	1	0	1
e	1	1	1	1	0

Vértice intermedio: e

Caminos Mínimos: ALGORITMO DE FLOYD

El problema es determinar **todos los caminos mínimos** dentro de un grafo. Es decir, **todos los caminos mínimos que toman como partida cualquier vértice del grafo y toman como llegada cualquier otro vértice del grafo**.

Pasos:

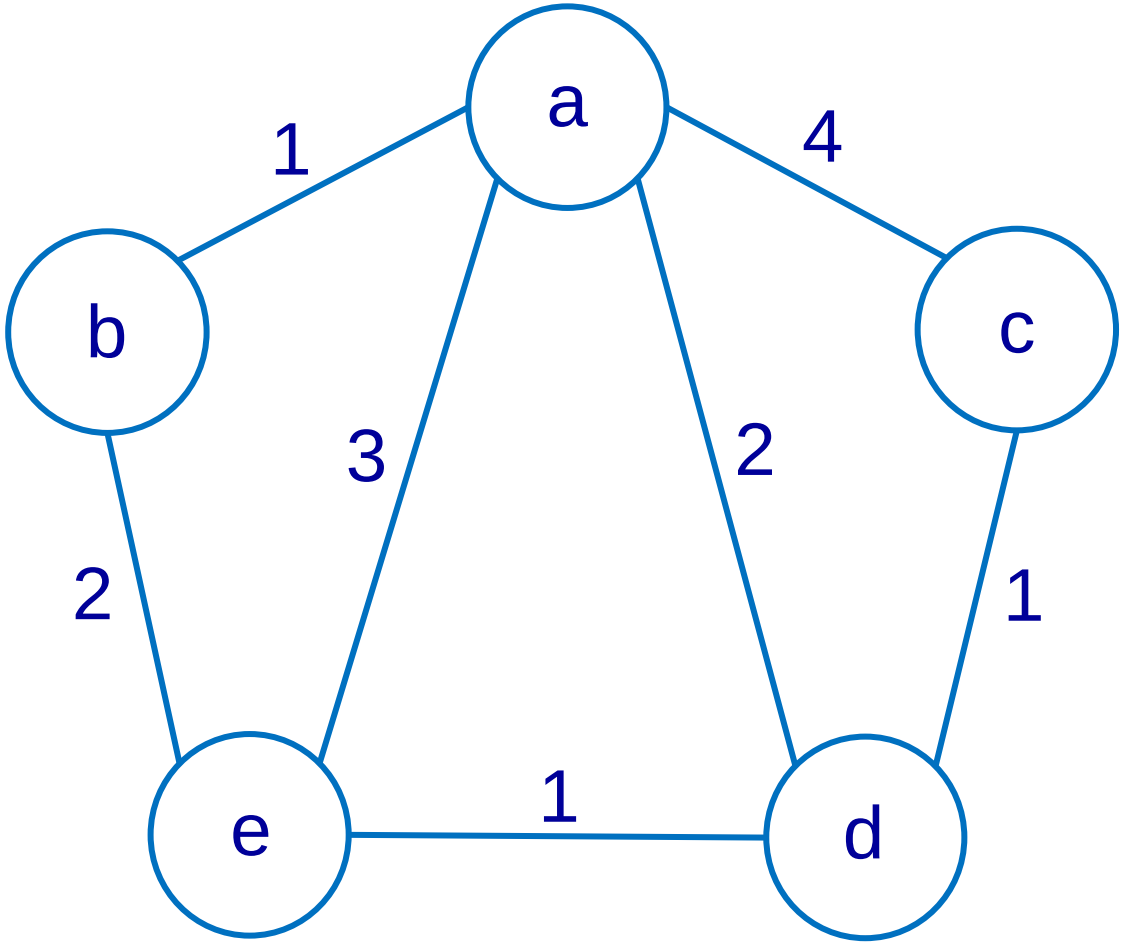
1. Construimos **la matriz A**, con los costes asociados a los arcos del grafo dado.
2. Realizamos **n *iteraciones*** (número de vértices del grafo), según se indica posteriormente.
3. Después de la ***iteración k-ésima***, A contiene los costes de **los caminos mínimos** que sólo utilizan como vértices intermedios el conjunto {0, 1,..., k}.
4. Finalmente, tras la ***iteración n-ésima***, se obtiene la matriz A con las **longitudes de todos los caminos mínimos posibles del grafo**.

Para la ***iteración k-ésima*** para **cada par de vértices** (***i, j***), hay que comprobar si **existe un camino que pasa** por el ***vértice k*** que sea **más corto** que el mejor camino que se tenga y que únicamente pasa por {0, 1,..., k-1}, es decir:

$$A_k[i, j] = \text{mínimo } \{(A_{k-1}[i, j]); A_{k-1}[i, k] + A_{k-1}[k, j]\}$$

- Un camino optimo que pasa por ***k***, **no lo hace 2 veces**.
- En la ***k-ésima*** iteración **no cambian los índices de la fila k ni los de la columna k**, puesto que consideramos que **A[k, k] = 0**.
- Cuando **dos vértices V_i y V_j no estén conectados** el coste $[V_i, V_j]$ será **infinito (∞)**.

Caminos Mínimos: ALGORITMO DE FLOYD



Permite calcular **caminos de longitud mínima** en **grafos no dirigidos, conexos y ponderados**.

Pasos:

Paso 1.- Etiquetado de los vértices con L(v).

$$L(v) = \begin{cases} 0 \ si & v = x \\ \infty \ si & v \neq x \end{cases}$$

donde:

- x el **vértice de inicio del camino**
- ∞ un **número positivo arbitrariamente grande**.

Paso 2.- Selección de la Etiqueta Mínima.

Se busca un vértice cuya etiqueta **L(v)** sea la mínima. Si hay varias iguales, entonces se elige un vértice cualquiera.

- Si $v = y$, siendo y el último vértice del camino, la longitud del camino más corto entre el vértice x de inicio y el vértice final y es **L(y)**. En esta situación se **acabó** el algoritmo.
- Si $v \neq y$, se pasa al siguiente paso del algoritmo.

Paso 3.- Nuevo etiquetado de los vértices.

Para cada vértice **w**, adyacente con el vértice **v** (cuya **L(v)** es mínima), se calcula el valor de una nueva etiqueta $\mathbf{L_i(w)}$ con el siguiente criterio:

$$L_i(w) = \text{mín. } \{L(w), L(v) + p(v, w)\}$$

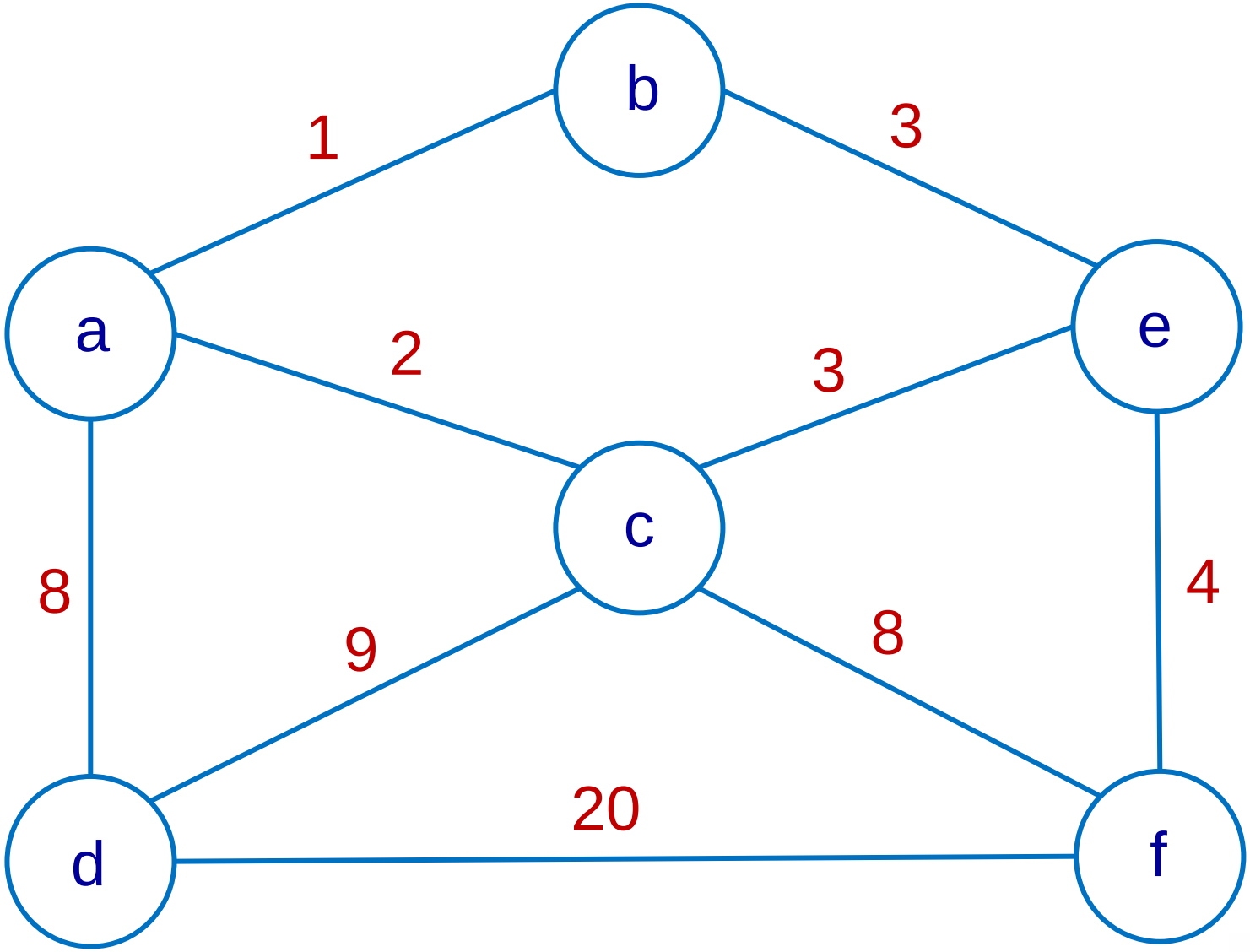
donde **p(v, w)** es la etiqueta o peso de la arista que tiene como extremos los vértices **v** y **w**.

Paso 4.- Simplificación del grafo.

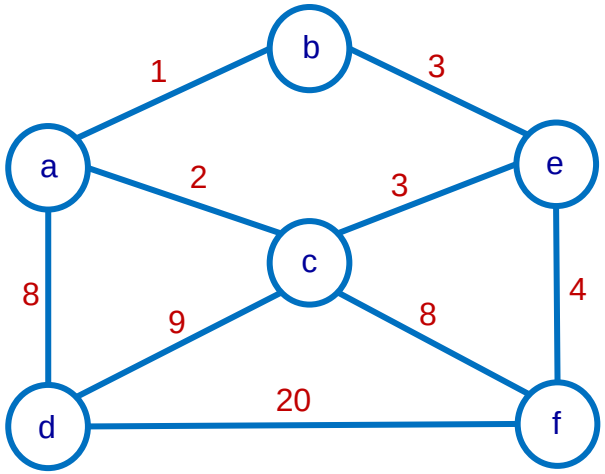
Se eliminan del grafo el vértice **v** (cuya L(v) es mínima) y las aristas que contienen a **v**.

Volver al **paso 2** hasta el final del algoritmo.

Caminos mínimos: ALGORITMO DE DIJKSTRA



Caminos mínimos: ALGORITMO DE DIJKSTRA



Vértice Mínimo	a	b	c	d	e	f	Arista del vértice mínimo