Optimización Combinatoria

Maestría en Cómputo Estadístico

Centro de Investigación en Matemáticas A.C.

Qué es optimización combinatoria? I

- En términos generales, es un problema de optimización donde el dominio / espacio de búsqueda es discreto.
- El espacio de búsqueda suele ser más grande, por lo que la fuerza bruta se vuelve ineficiente computacionalmente
- Los problemas de optimización combinatoria tienen varias aplicaciones que surgen en diferentes áreas.

Qué es optimización combinatoria? II

Algunos ejemplos son:

- Problema del agente viajero (TSP).
- Problema de ruteo de vehículos (VRP).
- Problema de planificación (calendarización).
- Ploblemas de cadena de suministro.
- Problemas de agrupamiento de datos.

Métodos

Algunos métodos para resolver este tipo de problemas son:

- Ruta más corta.
- Árboles de expansión.
- Flujo en redes.

CIMAT

Ruta más corta I

- Uno de los problemas de optimización combinatoria más conocidos es encontrar la ruta más corta entre dos vértices.
- El problema de la ruta más corta se puede definir para un grafo dirigido o no dirigido.
- Existen varias variantes, así como un conjunto diverso de algoritmos para su solución.

Ruta más corta II

Descripción del problema

Considerar un grafo dirigido $G = (V, A)$, donde V es un conjunto de vértices o nodos y $A = \{(i, j) : i, j \in V\}$ es un conjunto de arcos (pares ordenados de vértices). Para todo $(i, j) \in A$, sea c_{ij} el peso del arco (i, j). Dados $s, t \in V$, el problema de caminos mínimos consiste en encontrar el camino dirigido de peso mínimo que conecta el nodo s con el nodo t.

Ruta más corta III

Las **variables de decisión** están asociadas a los arcos del grafo, es decir, $\forall (i,j) \in A$,

$$x_{ij} = \begin{cases} 1, & \text{Si el arco } (i,j) \text{ pertenece al camino.} \\ 0, & \text{en otro caso.} \end{cases}$$

CIMAT

Ruta más corta IV

Función objetivo: se requiere encontrar el camino de peso total mínimo, es decir, la suma de los pesos de los arcos del camino debe ser lo más pequeña posible:

$$\text{minimizar } z = \sum_{(i,j)\in A} c_{ij} x_{ij}$$

CIMAT

Ruta más corta V

Las **restricciones** son:

- Se debe salir del nodo origen s:

$$\sum_{j \in V:(s,j) \in A} x_{sj} = 1$$

- Se debe llegar al nodo destino t:

$$\sum_{j \in V:(j,t) \in A} x_{jt} = 1$$

- Para los nodos internos del camino, $i \in V \setminus (\{s\} \cup \{t\})$. Es decir, En los nodos internos del camino el flujo que entra es igual al que sale:

$$\sum_{j \in V:(j,i) \in A} x_{ji} = \sum_{j \in V:(i,j) \in A} x_{ij}$$

- Dominio de las variables x: $x_{ij} \in \{0,1\} \quad \forall (i,j) \in A$

Ruta más corta VI

- Las variantes más comunes del problema de ruta más corta son:
  - Problema de ruta más corta con un origen.
  - Problema de ruta más corta con un destino.
  - Problema de ruta más corta con un solo par.
  - Problema de ruta más corta con todos los pares.

Ruta más corta VII

- Los algoritmos más conocidos para resolver el problema de la ruta más corta son:
  - Dijkstra.
  - Jhonson.
  - A^*.
  - Bellman-Ford.
  - Seidel.

Algoritmo Dijkstra I

- Es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.
- Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo concibió en 1956 y lo publicó por primera vez en 1959.
- Realiza $O(n^2)$ operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo con n vértices.

CIMAT

Algoritmo Dijkstra I

Algoritmo Dijkstra II

Pseudocódigo

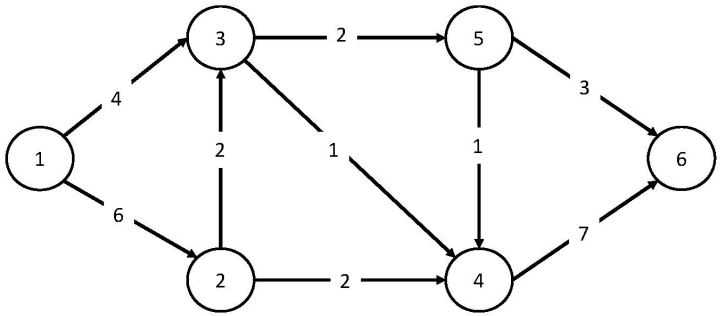
$S \leftarrow \emptyset$
$\bar{S} \leftarrow V$
$d(s) \leftarrow 0$
$pred(s) \leftarrow 0$
$d(i) \leftarrow \infty, \quad \forall i \in V \setminus \{s\}$
while $|S| < |V|$ **do**
  $i \in \arg\min_{j \in \bar{S}} \{d(j)\}$
  $S \leftarrow S \cup \{i\}$
  $\bar{S} \leftarrow \bar{S} \setminus \{i\}$
  for each $(i, j) \in A$ **do**
    if $d(j) > d(i) + c_{ij}$ **then**
      $d(j) \leftarrow d(i) + c_{ij}$
      $pred(j) \leftarrow i$
    end if
  end for
end while

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Instancia del problema

Considerar la instancia del problema de caminos mínimos que se muestra en el grafo de la figura.



Paso Iterativo

CIMAT

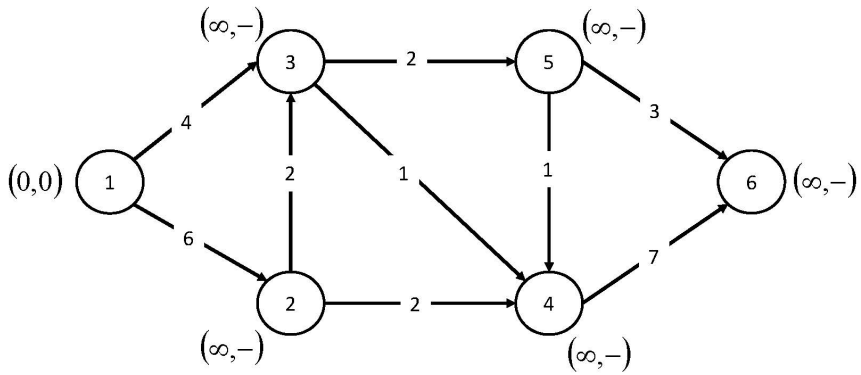
Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Inicialización

- Hacer 0 la etiqueta de distancia del nodo S
- Hacer 0 la etiqueta de predecesor del nodo t
- Hacer ∞ la etiqueta de distancia de $i \in V \setminus (\{s\})$

Paso Iterativo



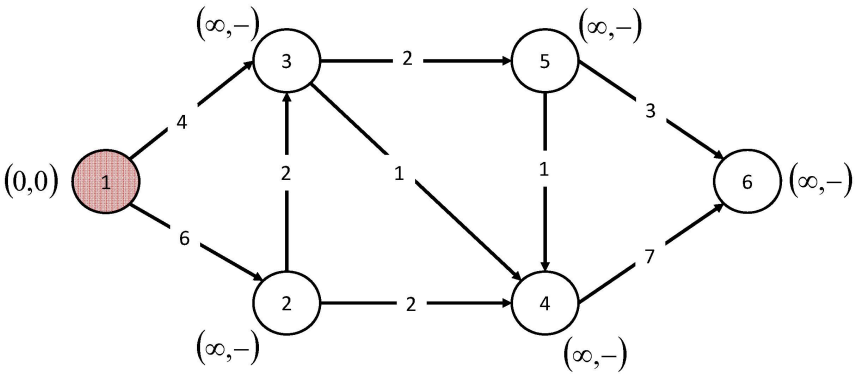
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia: Nodo 1



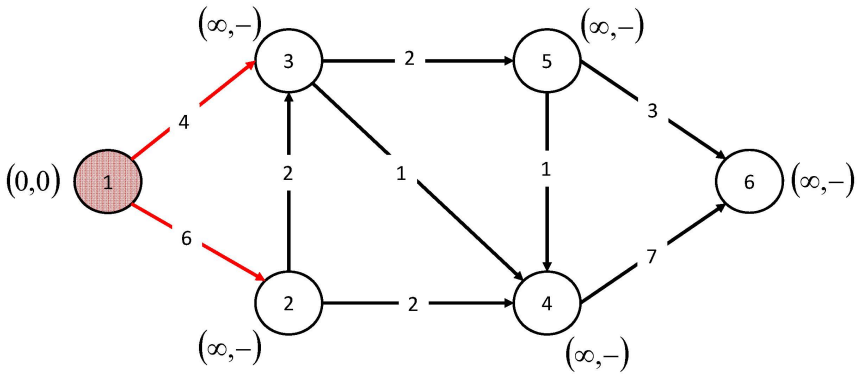
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 1:



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 1:
Arco $(1, 2)$:
dado que

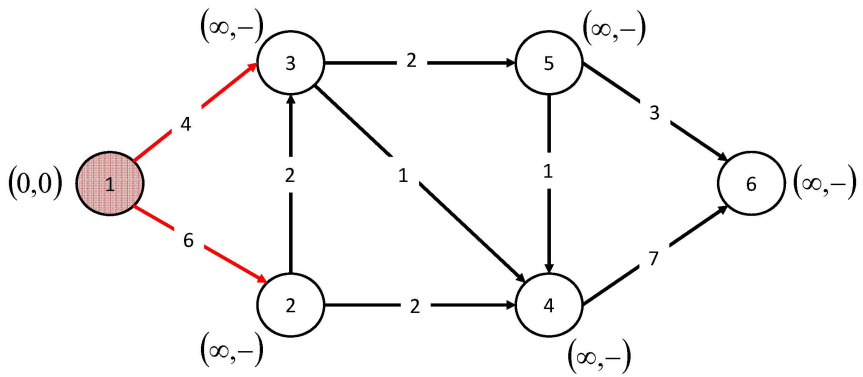
$$d_1 + c_{12} < d_2,$$

hacer

$$d_2 := d_1 + c_{12} = 0 + 6 = 6,$$

hacer

$$pred_2 := 1$$



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 1:
Arco $(1, 3)$:
dado que

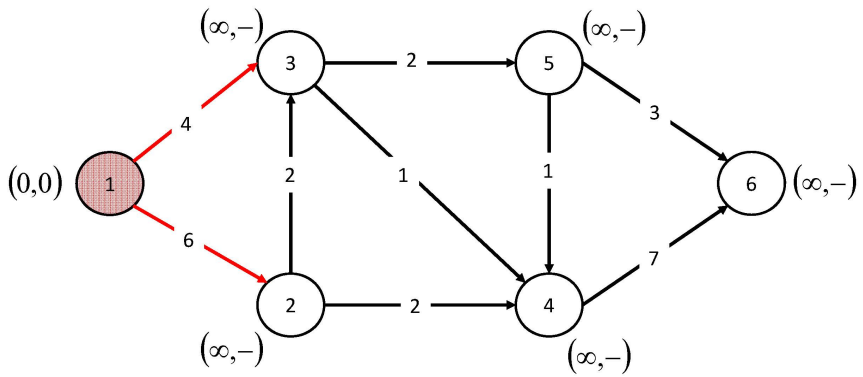
$$d_1 + c_{13} < d_3,$$

hacer

$$d_3 := d_1 + c_{13} = 0 + 4 = 4,$$

hacer

$$pred_3 := 1$$



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 1:
Arco $(1, 3)$:
dado que

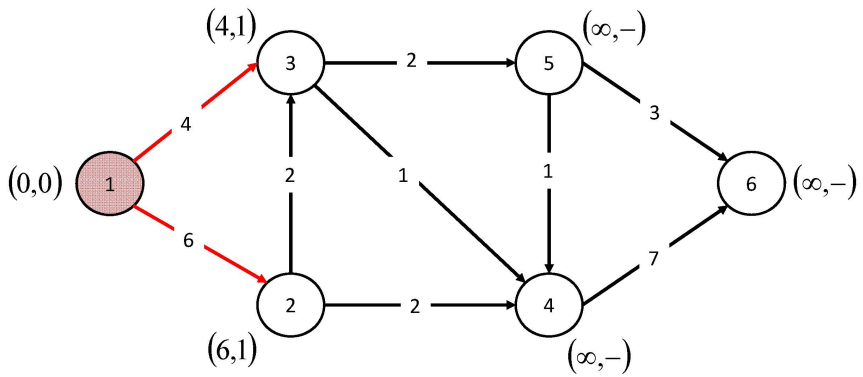
$$d_1 + c_{13} < d_3,$$

hacer

$$d_3 := d_1 + c_{13} = 0 + 4 = 4,$$

hacer

$$pred_3 := 1$$



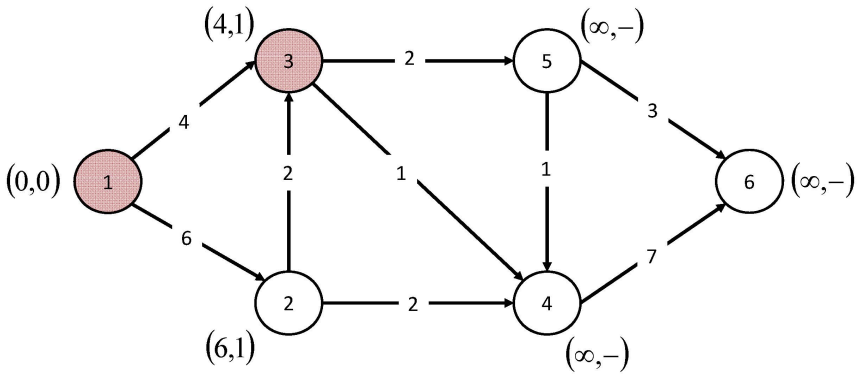
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia aún no procesado: Nodo 3



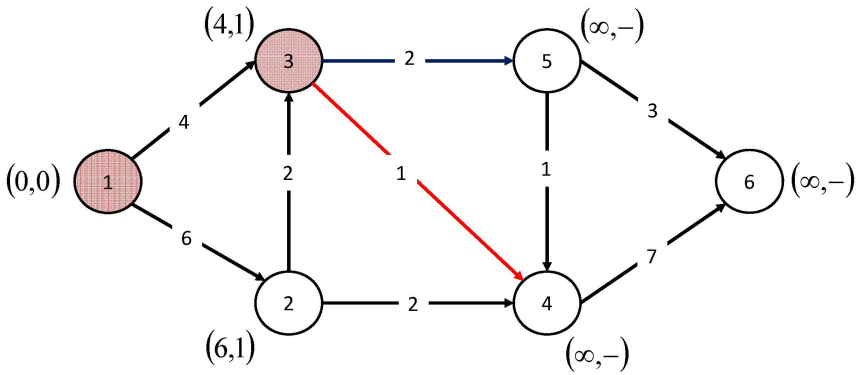
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 3:



Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 3:
Arco $(3, 4)$:
dado que

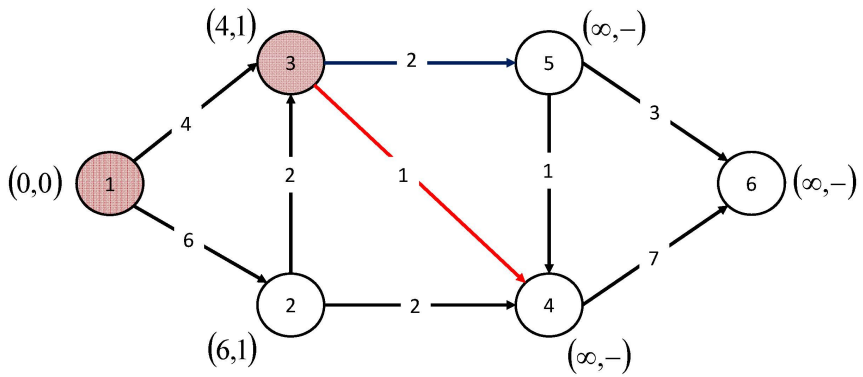
$$d_3 + c_{34} < d_4,$$

hacer

$$d_3 := d_3 + c_{34} = 4 + 1 = 5,$$

hacer

$$pred_5 := 3$$



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 3:
Arco $(3, 4)$:
dado que

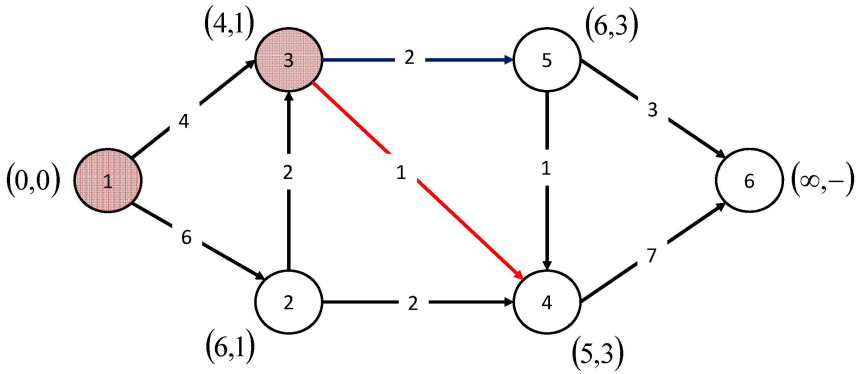
$$d_3 + c_{34} < d_4,$$

hacer

$$d_3 := d_3 + c_{34} = 4 + 1 = 5,$$

hacer

$$pred_5 := 3$$



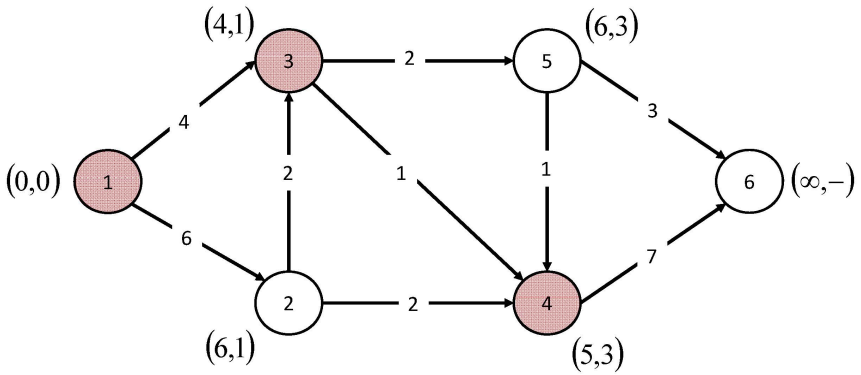
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia aún no procesado: Nodo 4



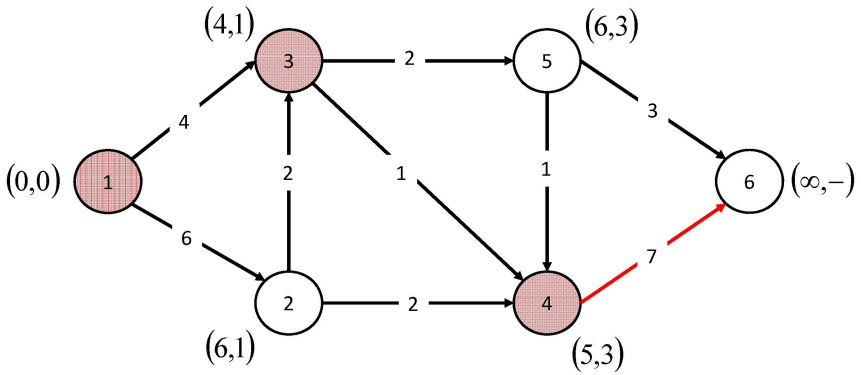
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 4:



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 4:
Arco $(4, 6)$:
dado que

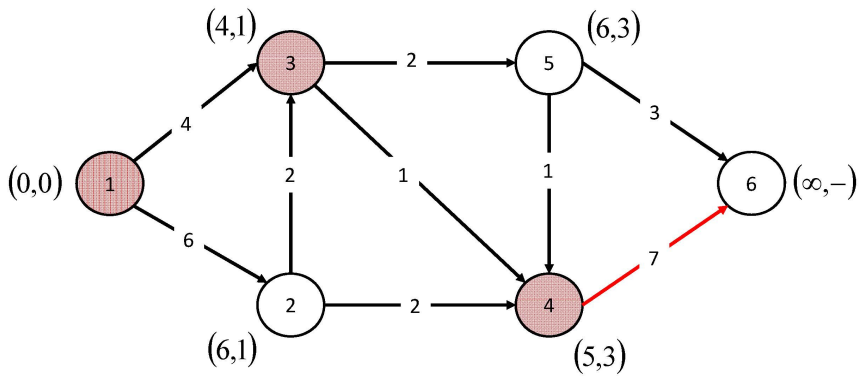
$$d_4 + c_{46} < d_6,$$

hacer

$$d_6 := d_4 + c_{46} = 5 + 7 = 12,$$

hacer

$$pred_6 := 4$$



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 4:
Arco $(4, 6)$:
dado que

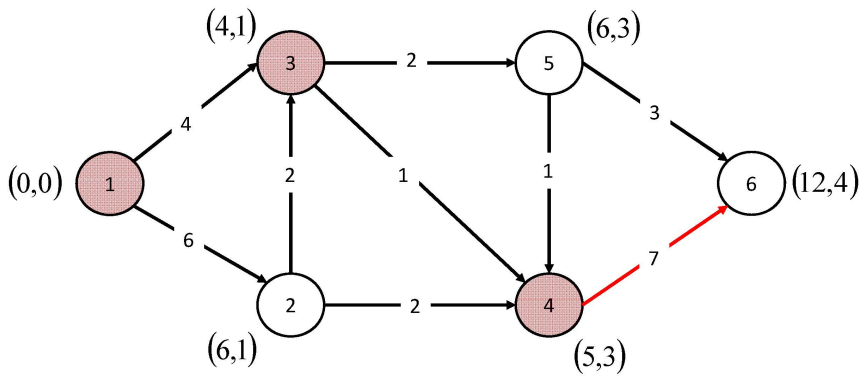
$$d_4 + c_{46} < d_6,$$

hacer

$$d_6 := d_4 + c_{46} = 5 + 7 = 12,$$

hacer

$$pred_6 := 4$$



CIMAT

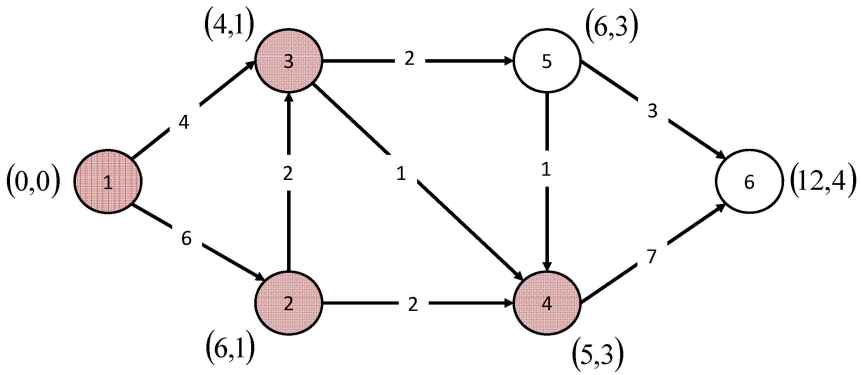
Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia aún no procesado: Nodo 2

Los empates se rompen de manera arbitraria



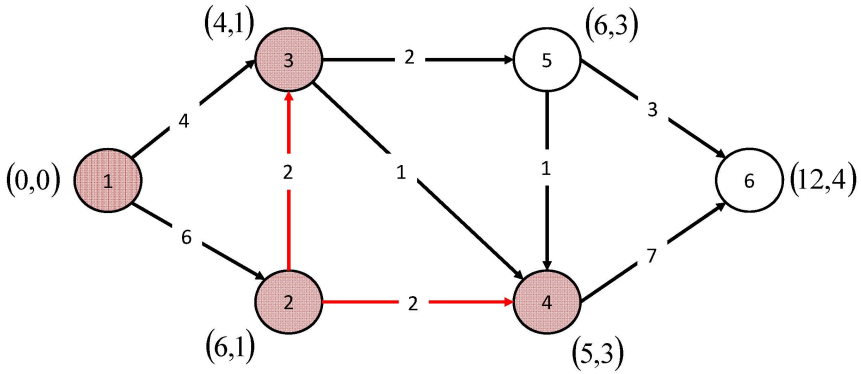
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 2:



CIMAT

Algoritmo Dijkstra: Ejemplo I

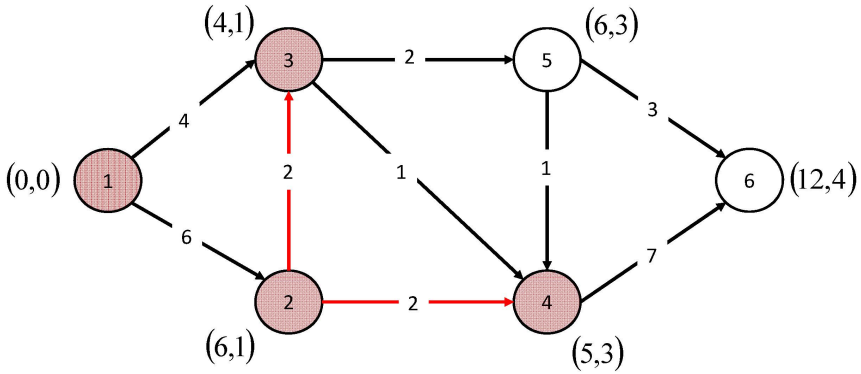
Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 2:
Arco $(2, 3)$:
dado que

$$d_2 + c_{23} > d_3,$$

No se actualizan las etiquetas del nodo 3



CIMAT

Algoritmo Dijkstra: Ejemplo I

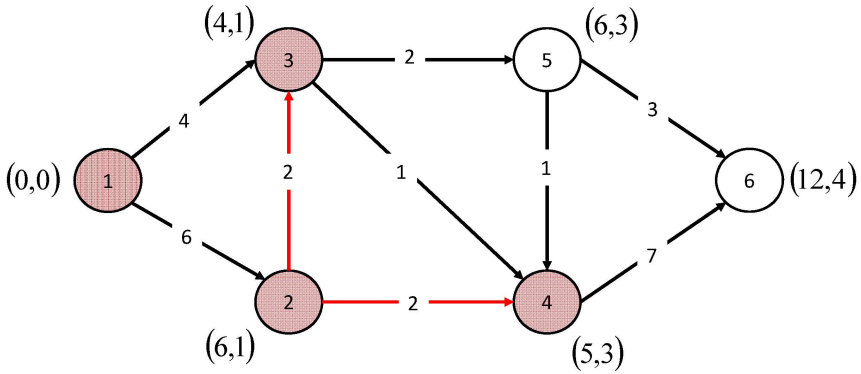
Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 2:
Arco $(2, 4)$:
dado que

$$d_2 + c_{24} > d_4,$$

No se actualizan las etiquetas del nodo 4



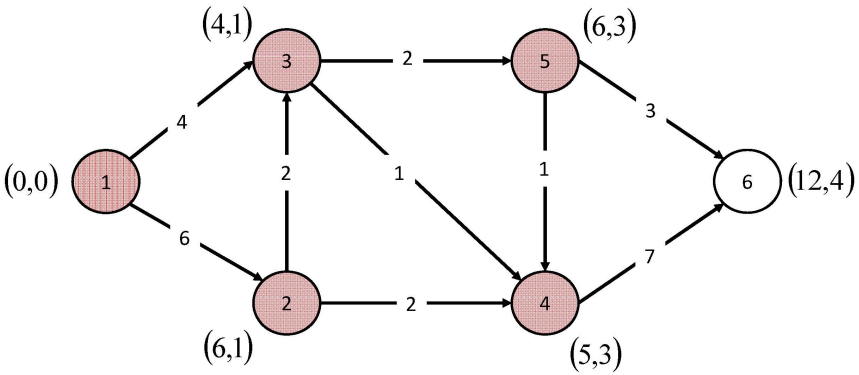
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia aún no procesado: Nodo 5



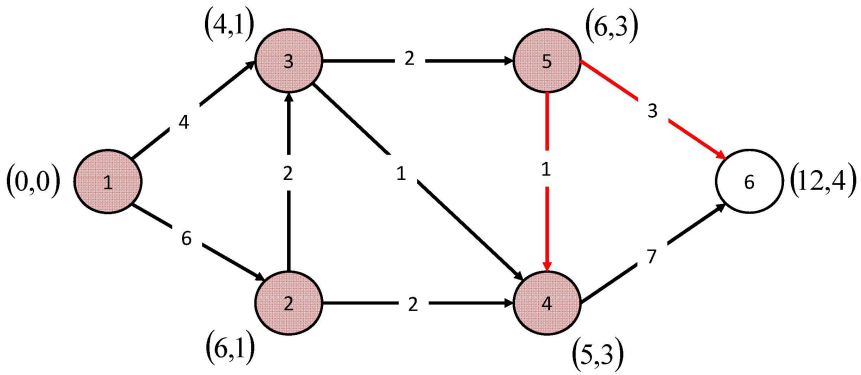
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 5:



CIMAT

Algoritmo Dijkstra: Ejemplo I

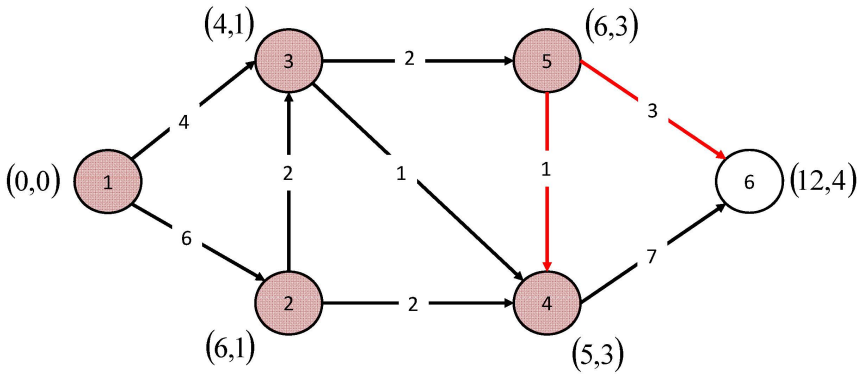
Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 5:
Arco $(5, 4)$:
dado que

$$d_5 + c_{54} > d_4,$$

No se actualizan las etiquetas del nodo 4



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 5:
Arco $(5, 6)$:
dado que

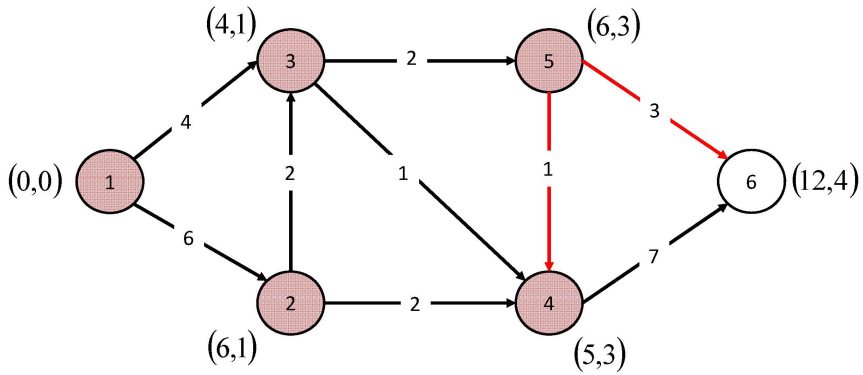
$$d_5 + c_{56} < d_6,$$

hacer

$$d_6 := d_5 + c_{56} = 6 + 3 = 9,$$

hacer

$$pred_6 := 5$$



CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Para todos los arcos que emanan del nodo 5:
Arco $(5, 6)$:
dado que

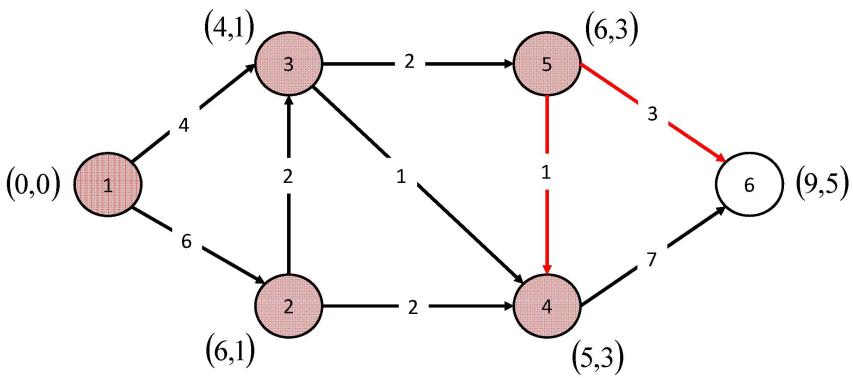
$$d_5 + c_{56} < d_6,$$

hacer

$$d_6 := d_5 + c_{56} = 6 + 3 = 9,$$

hacer

$$pred_6 := 5$$



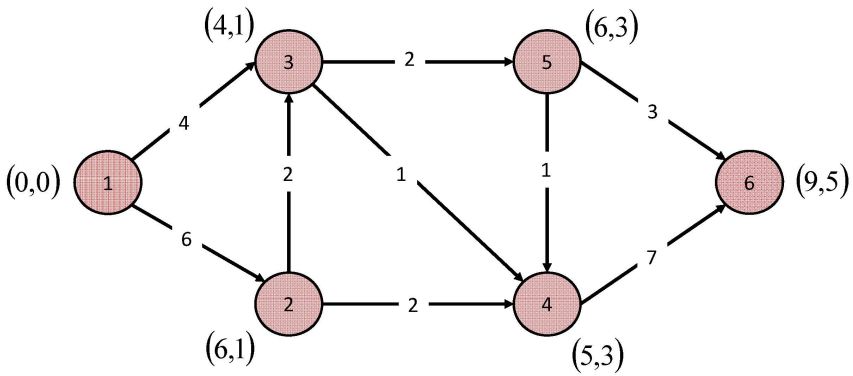
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

Seleccionar el nodo con menor etiqueta de distancia aún no procesado: Nodo 6



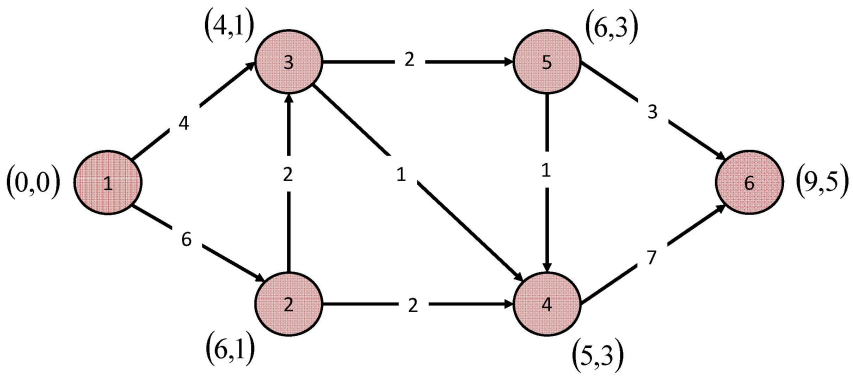
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

No hay nodos que emanen del nodo 6



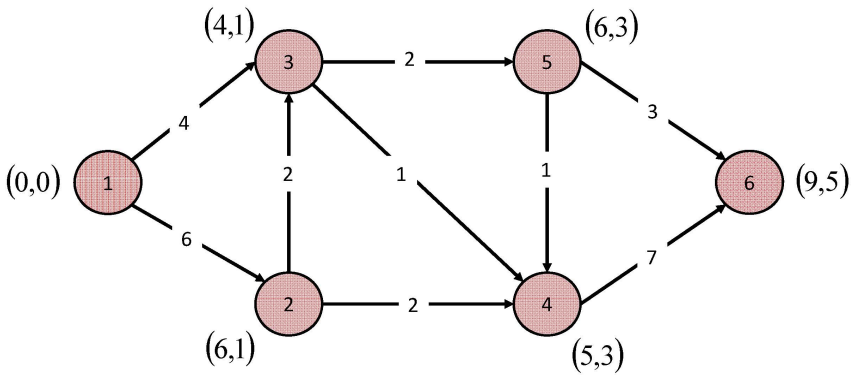
CIMAT

Algoritmo Dijkstra: Ejemplo I

Encontrar el camino más corto entre nodo 1 y el nodo 6.

Paso Iterativo

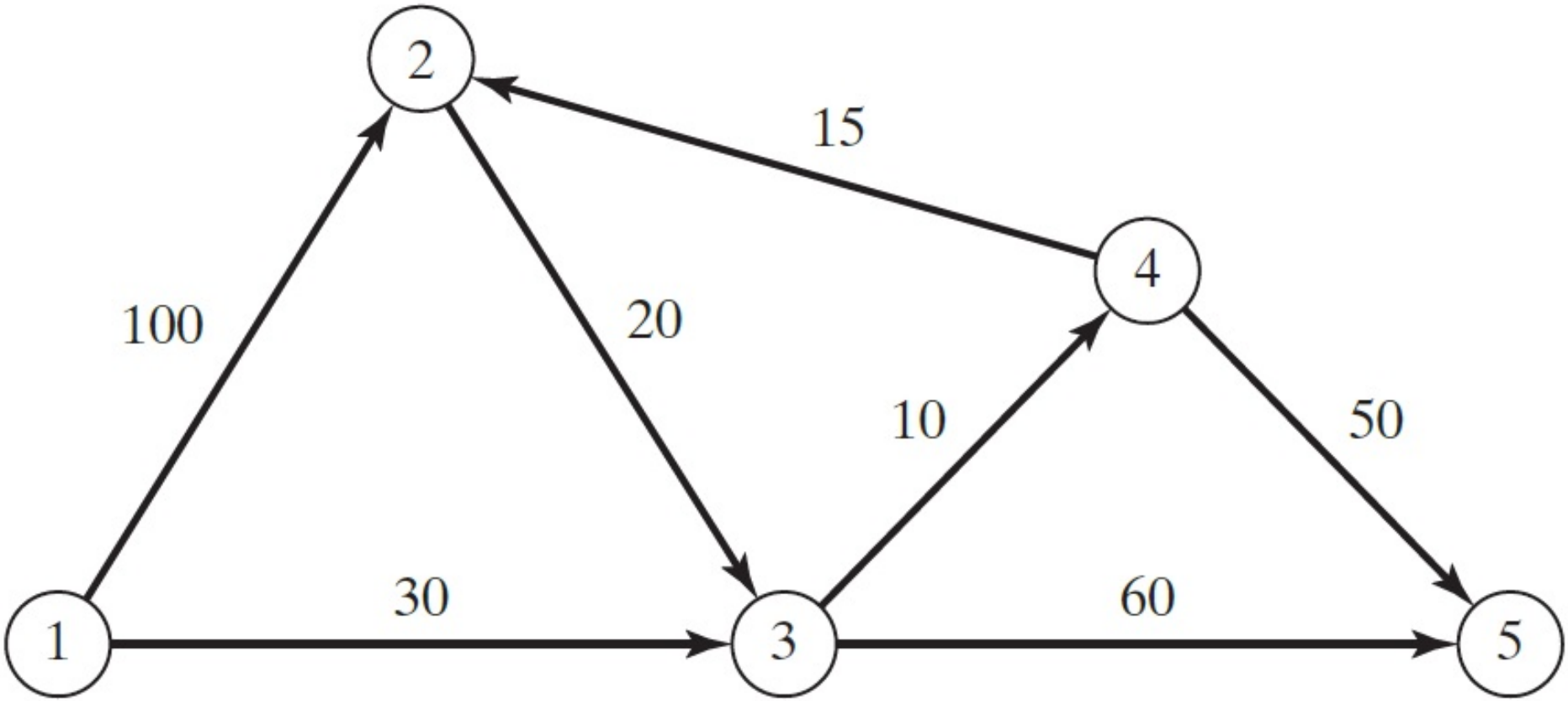
Todos los nodos han sido procesados, se satisface el criterio de terminación



CIMAT

Algoritmo Dijkstra: Ejemplo II

La red de la siguiente figura da las rutas permisibles y sus longitudes en millas entre la ciudad 1 (nodo 1) y las otras cuatro ciudades (nodos 2 a 5). Determine las rutas más cortas entre la ciudad 1 y cada una de las cuatro ciudades restantes.



Ejercicio Seervada Park

En fecha reciente se reservó el área de Seervada Park para paseos y campamentos. No se permite la entrada de automóviles, pero existe un sistema de caminos angostos y sinuosos para tranvías y para "jeeps" conducidos por los guardabosques. En la siguiente figura se muestra este sistema de caminos —sin las curvas—, en donde O es la entrada al parque; las otras letras representan la localización de las casetas de los guardabosques y otras instalaciones de servicio. Los números son las distancias en millas de estos caminos accidentados.

CIMAT

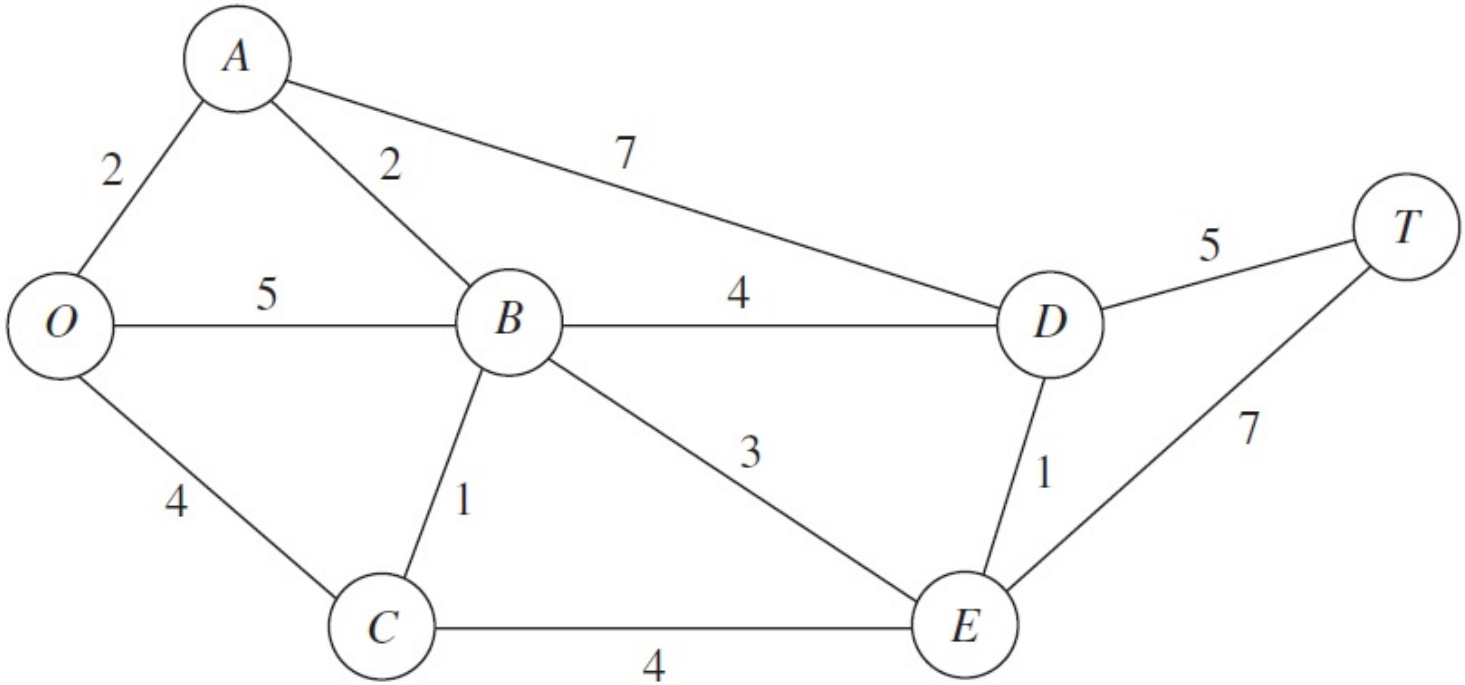
Ejercicio Seervada Park

El parque contiene un mirador a un hermoso paisaje en la estación T. Unas cuantas camionetas transportan a los visitantes desde la entrada a la estación T y viceversa.

En este momento la administración del parque se enfrenta al problema de determinar qué ruta, desde la entrada del parque a la estación T, es la que representa la distancia total más corta para la operación de los tranvías.

CIMAT

Ejercicio Seervada Park



CIMAT

Algoritmo Ford-Moore-Bellman I

- Este algoritmo permite encontrar la ruta más corta con un origen.
- Es una implementación más eficiente del algoritmo de Ford (1956), descubierta independientemente por Moore (1957) y por Bellman (1958).
- Aunque puede ser más lento que el algoritmo de Dijkstra, es más versátil.
- La complejidad computacional es $O(nm)$ donde n son las iteraciones y m arcos.

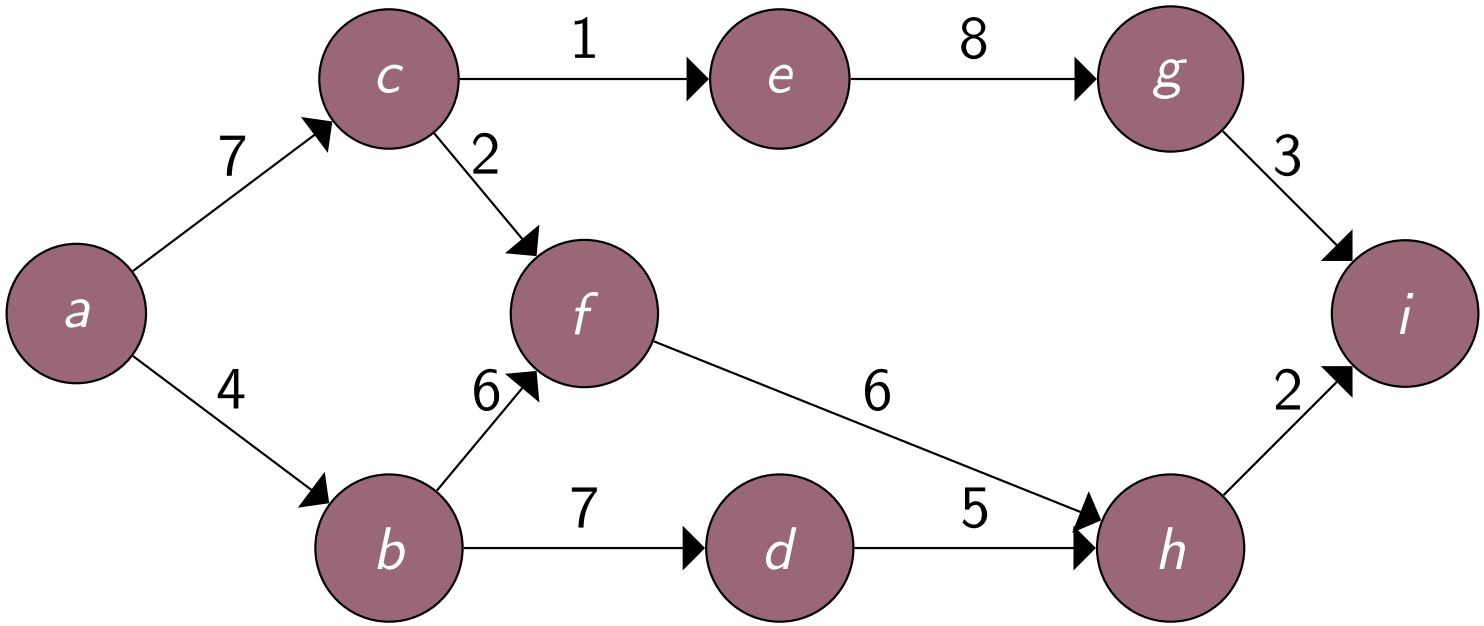
Algoritmo Ford-Moore-Bellman II

Algorithm Ford-Moore-Bellman algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, s the source vertex, and d the destination vertex
2: Set $l(s) \leftarrow 0$
3: Set $l(v) \leftarrow \infty : \forall v \in V \setminus \{s\}$
4: **for** $i = 1 to \mid V \mid -1$ **do**
5: **for** each pair $(u, v) \in E$ **do**
6: **if** $l(v) > l(u) + c(u, v)$ **then**
7: Set $l(v) \leftarrow l(u) + c(u, v)$
8: Set $p(v) \leftarrow u$
9: **end if**
10: **end for**
11: **end for**
12: **return** The path list from s to each $v \in V \setminus \{s\}$

Algoritmo Ford-Moore-Bellman: Ejemplo

Dado el siguiente grafo, encuentre la ruta más corta de a a todos los vértices.



CIMAT

Algoritmo Floyd-Warshall I

- El algoritmo Floyd-Warshall (1962) está diseñado para manejar el caso con todos los pares y desde todos los nodos del grafo.
- Multiples ejecuciones de los algoritmos Dijkstra o Ford-Moore-Bellman puede realizarse con una mayor complejidad computacional.
- Tiene complejidad computacional de $O(n^3)$.

CIMAT

Algoritmo Floyd-Warshall II

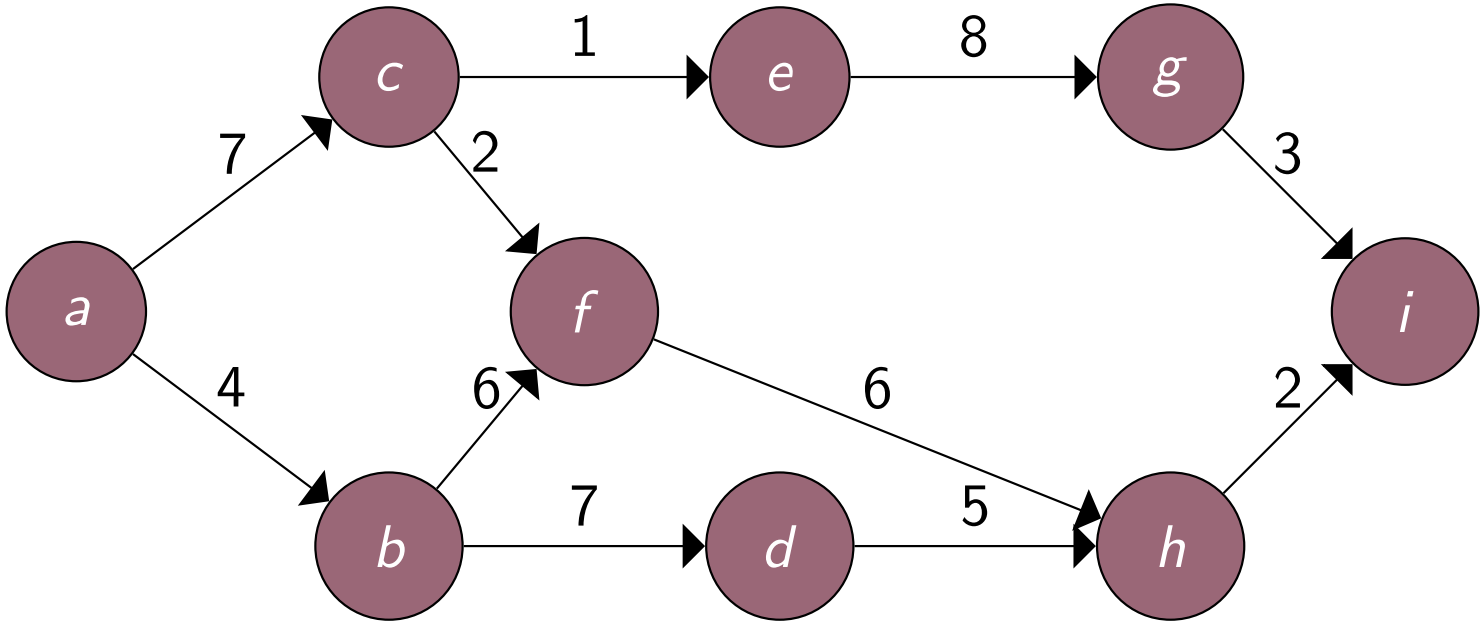
Algorithm Floyd-Warshall algorithm

Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
Set $l(i,j) \leftarrow c(i,j) : (i,j) \in E(G)$
Set $l(i,j) \leftarrow \infty : \forall (i,j) \in (V(G) \times V(G)) \setminus E(G),$ and $i \neq j$
Set $l(i,i) \leftarrow 0$
Set $p(i,i) \leftarrow i : (i,j) \in V(G)$
for $i = 1to \mid V \mid$ **do**
    for $j = 1to \mid V \mid$ **do**
        for $k = 1to \mid V \mid$ **do**
            if $l(j,k) > l(j,i) + l(i,k)$ **then**
                Set $l(j,k) \leftarrow l(j,i) + l(i,k)$
                Set $p(j,k) \leftarrow p(i,k)$
            end if
        end for
    end for
end for
return The path list from all pairs of vertices

CIMAT

Algoritmo Floyd-Warshall: Ejemplo

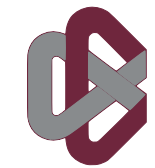
Dado el siguiente grafo, encontrar el camino más corto para todos los pares de vértices.



CIMAT

Árboles de mínima expansión I

- Árboles de mínima expansión (MSTs) son un tipo de problema de optimización que tiene similitudes con el problema de la ruta más corta.
- A diferencia del problema de la ruta más corta, en el problema MST queremos encontrar el conjunto de aristas que conecta a todos los vértices con el mínimo costo.



Árboles de mínima expansión II

- El problema MST tiene muchas aplicaciones, por ejemplo:
  - Diseño de redes.
  - Problema del agente viajero.
  - Análisis de agrupaciones.
  - Segmentación de imágenes.

Árboles de mínima expansión III

- Algunos de los algoritmos clásicos para resolver el problema MST son:
  - Kruskal.
  - Prim.
  - Eliminación-Inversa.

Árboles de mínima expansión IV

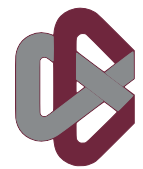
Los dos problemas involucran también el hecho de seleccionar un conjunto de aristas con la longitud total más corta entre todos los conjuntos de aristas que satisfacen cierta propiedad. En el caso del problema de la ruta más corta, esta propiedad es que la ruta seleccionada debe proporcionar una trayectoria entre el origen y el destino.

Mientras que para el árbol de expansión mínima la propiedad que se requiere es que la ruta seleccionada debe proporcionar una trayectoria entre cada par de nodos.

Árboles de mínima expansión V

El problema del árbol de expansión mínima se puede resumir de la siguiente manera:

- Se tienen los nodos de una red pero no las aristas. En su lugar se proporcionan las aristas potenciales y la longitud positiva de cada una si se insertan en la red. (Las medidas alternativas para la longitud de una arista incluyen distancia, costo y tiempo.)
- Se desea diseñar la red con suficientes aristas para satisfacer el requisito de que haya un camino entre cada par de nodos.
- El objetivo es satisfacer este requisito de manera que se minimice la longitud total de las aristas insertadas en la red.



CIMAT

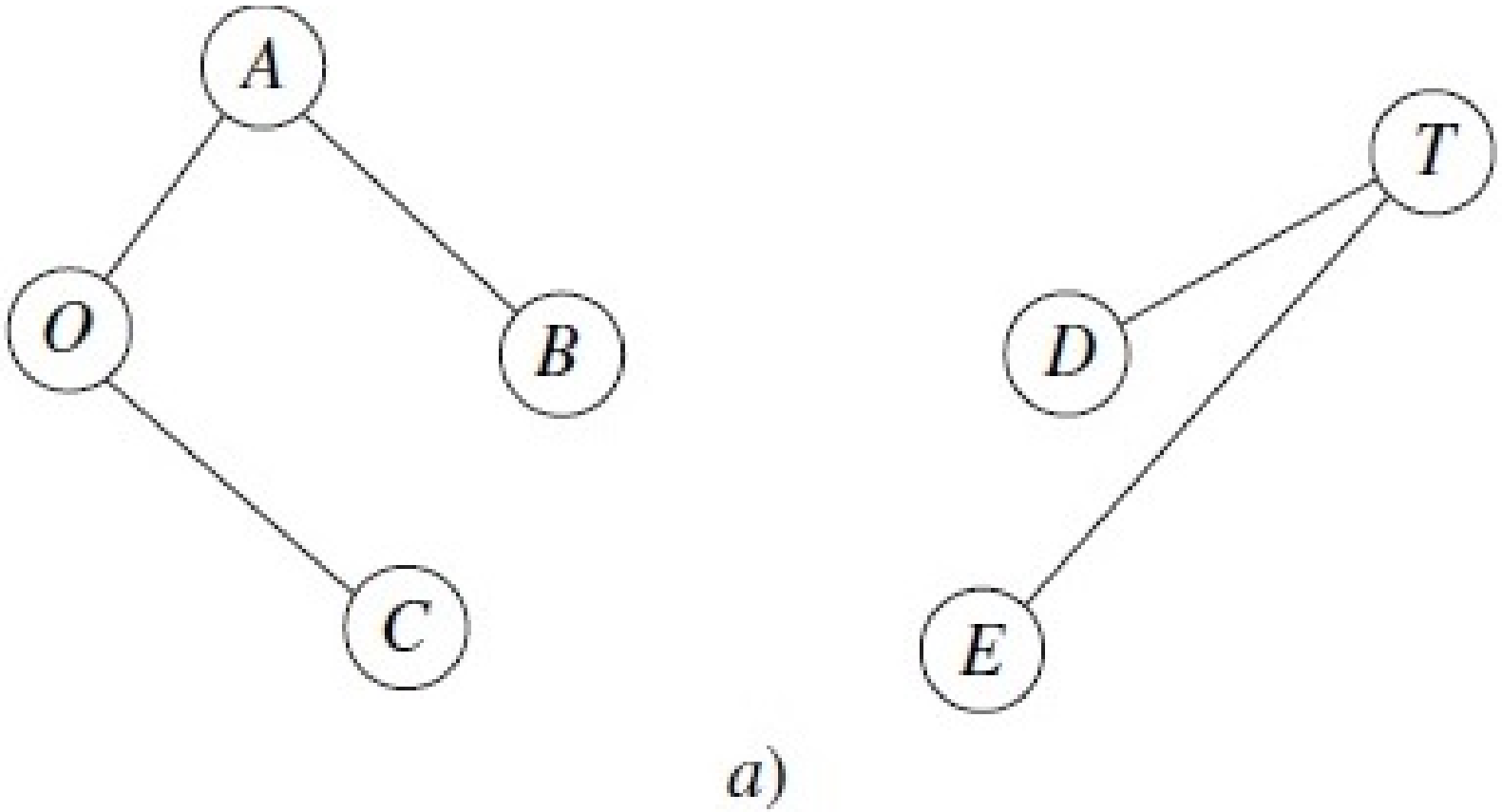
Árboles de mínima expansión VI

Una red con n nodos requiere de sólo (n–1) aristas para proporcionar una trayectoria entre cada par de nodos. No deben usarse más aristas puesto que ello aumentaría, sin necesidad, la longitud total de las aristas seleccionadas.

Las (n–1) aristas deben elegirse de tal manera que la red resultante —con sólo las aristas seleccionadas— forme un árbol de expansión.

Árboles de mínima expansión VII

¿Es un árbol de expansión?

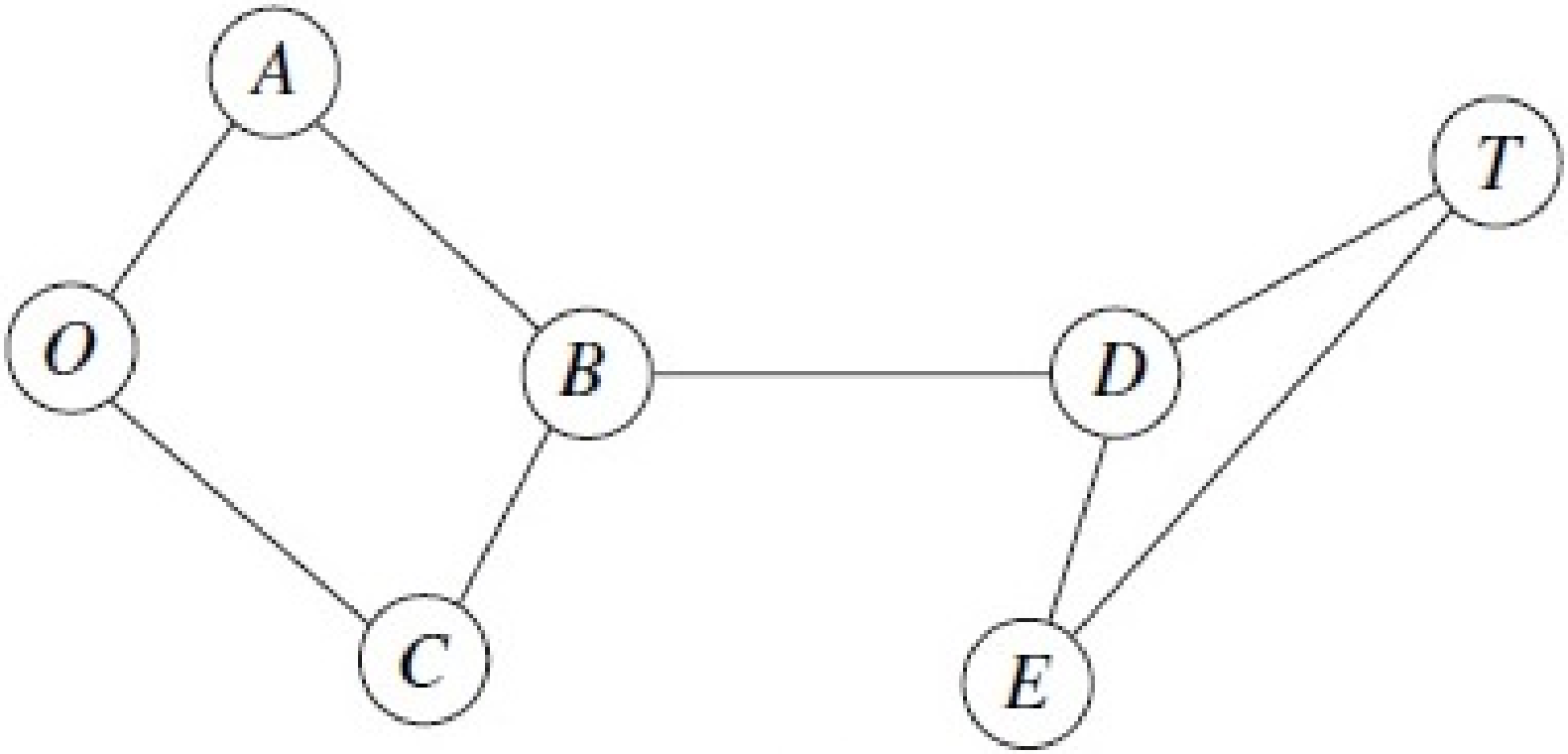


a)

CIMAT

Árboles de mínima expansión VIII

¿Es un árbol de expansión?

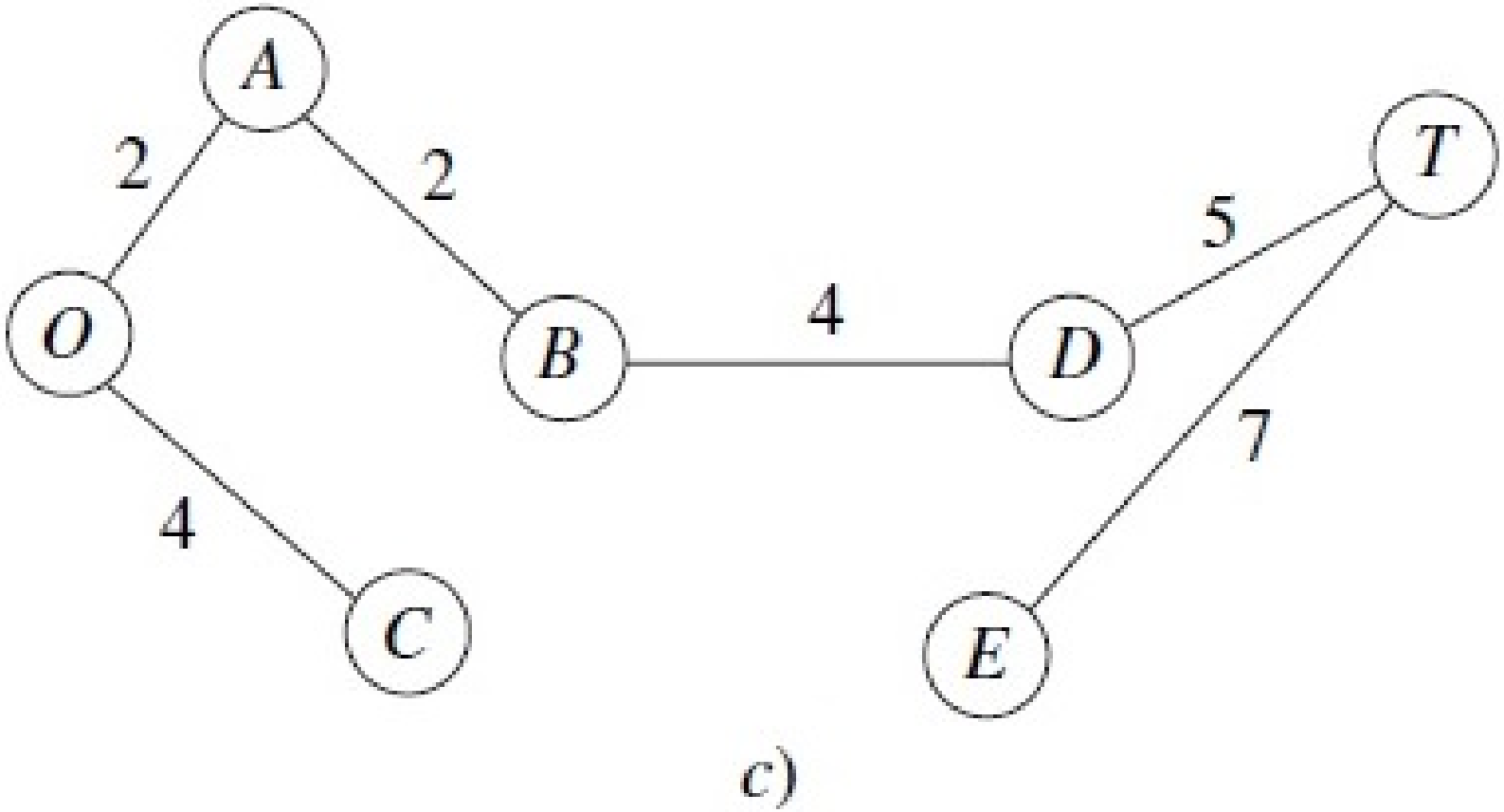


b)

CIMAT

Árboles de mínima expansión IX

¿Es un árbol de expansión?



c)

CIMAT

Árboles de mínima expansión I

Descripción del problema

Consideremos dado un grafo no dirigido $G = (N, E)$ donde cada arco $e \in E$ tiene asociado un costo c_e. Se llama *problema del árbol generador con costo mínimo* (del inglés Shortest Spanning Tree, SST) al problema de determinar un árbol generador $G' = (N, T)$ en G tal que $\sum_{e \in T} c_e$ es mínimo. Si G no es conexo, entonces el problema no es factible, si bien tiene sentido entonces plantearlo sobre cada una de la componentes conexas. Asumiremos por ello que G es conexo.

CIMAT

Árboles de mínima expansión II

Proposición:
Las siguientes afirmaciones son equivalentes:

- (N, T) es un árbol generador;
- cada par de nodos en (N, T) están unidos por un único camino;
- (N, T) es conexo y $|T| = n - 1$;
- (N, T) no tiene ciclos y $|T| = n - 1$;
- (N, T) es conexo pero $(N, T - \{e\})$ no lo es, para cualquier $e \in T$;
- (N, T) no tiene ciclos pero $(N, T \cup \{e\})$ tiene un ciclo, para cualquier $e \notin T$.

Árboles de mínima expansión III

Un modelo matemático de programación lineal entera es:

$$\min \quad \sum_{e \in E} c_e x_e \tag{1}$$

$$\text{sujeto a:} \quad \sum_{e \in E} x_e = n - 1 \tag{2}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \text{para todo } S \subset N \tag{3}$$

$$0 \leq x_e \leq 1 \quad \text{para todo } e \in E \tag{4}$$

$$x_e \in \mathbb{Z} \quad \text{para todo } e \in E \tag{5}$$

Donde la variable de decisión binaria x_e es igual a 1 si la arista e pertenece al árbol y 0 en otro caso.

CIMAT

Algoritmo Kruskal I

- Es un algoritmo voraz (greedy) que funciona agregando un peso creciente en cada arista.
- El costo computacional es $\mathcal{O}(m \log n)$ donde n son los vertices y m las aristas.

Algoritmo Kruskal II

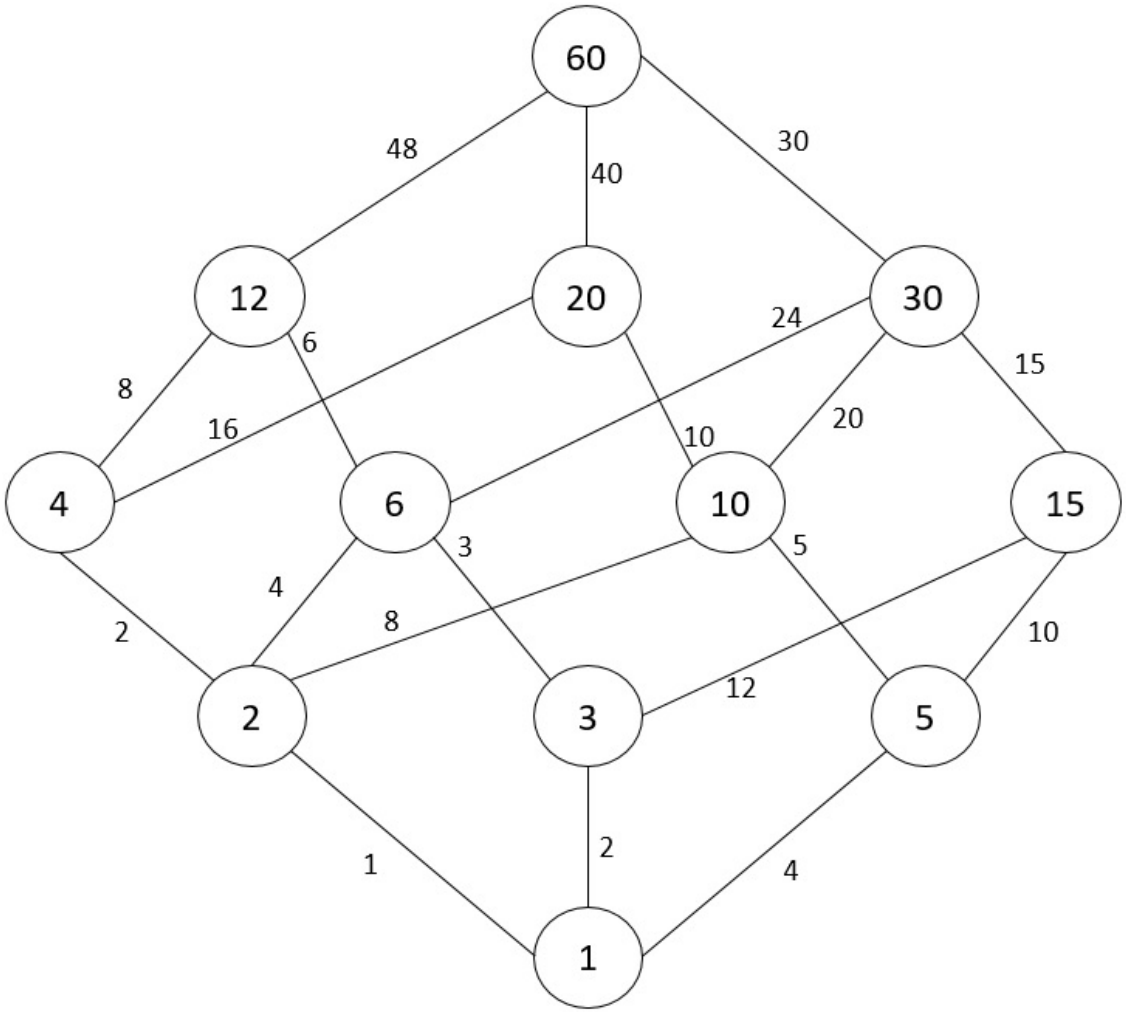
Algorithm Kruskal's algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Sort the edges $e \in E$ in ascending order
3: Set $T \leftarrow (V(G), \emptyset)$
4: **while** $T \neq V$ **do**
5: **if** $T + e_i$ contains no circuit **then**
6: $T \leftarrow T + e_i$
7: **end if**
8: **end while**
9: **return** The MST

CIMAT

Algoritmo Kruskal: Ejemplo I

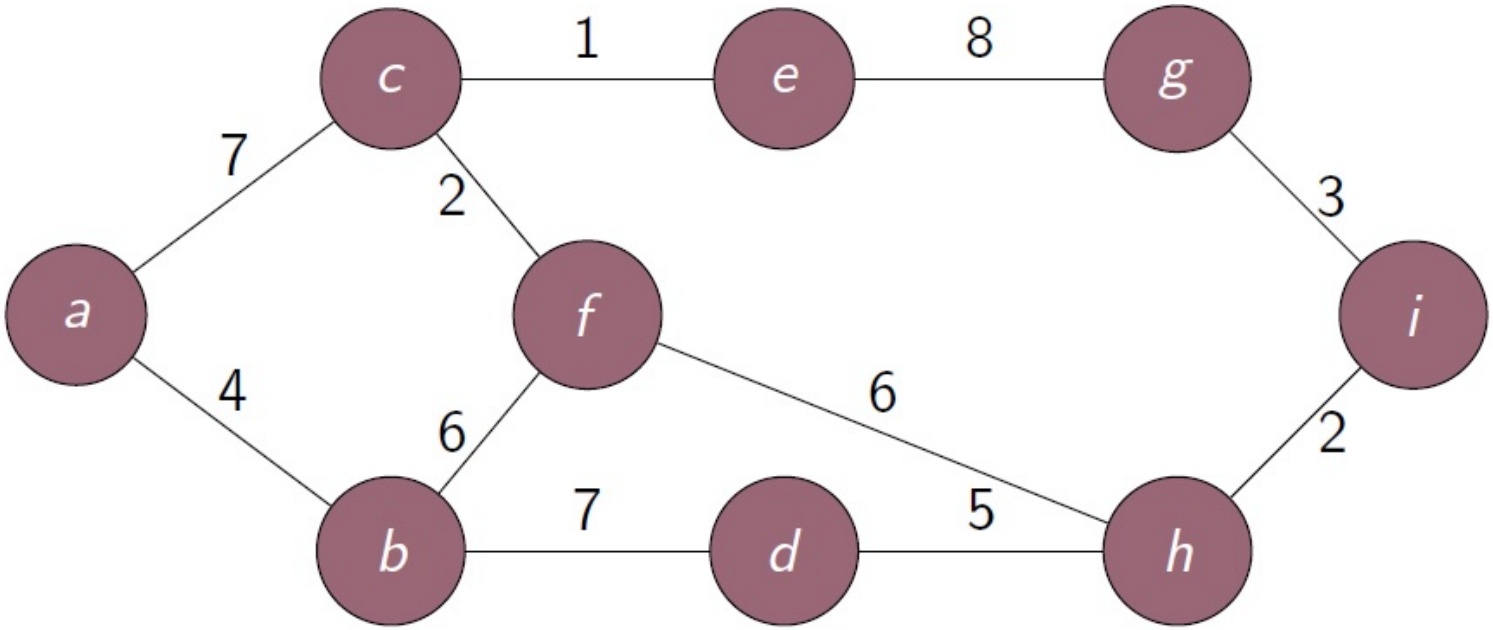
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Kruskal: Ejemplo II

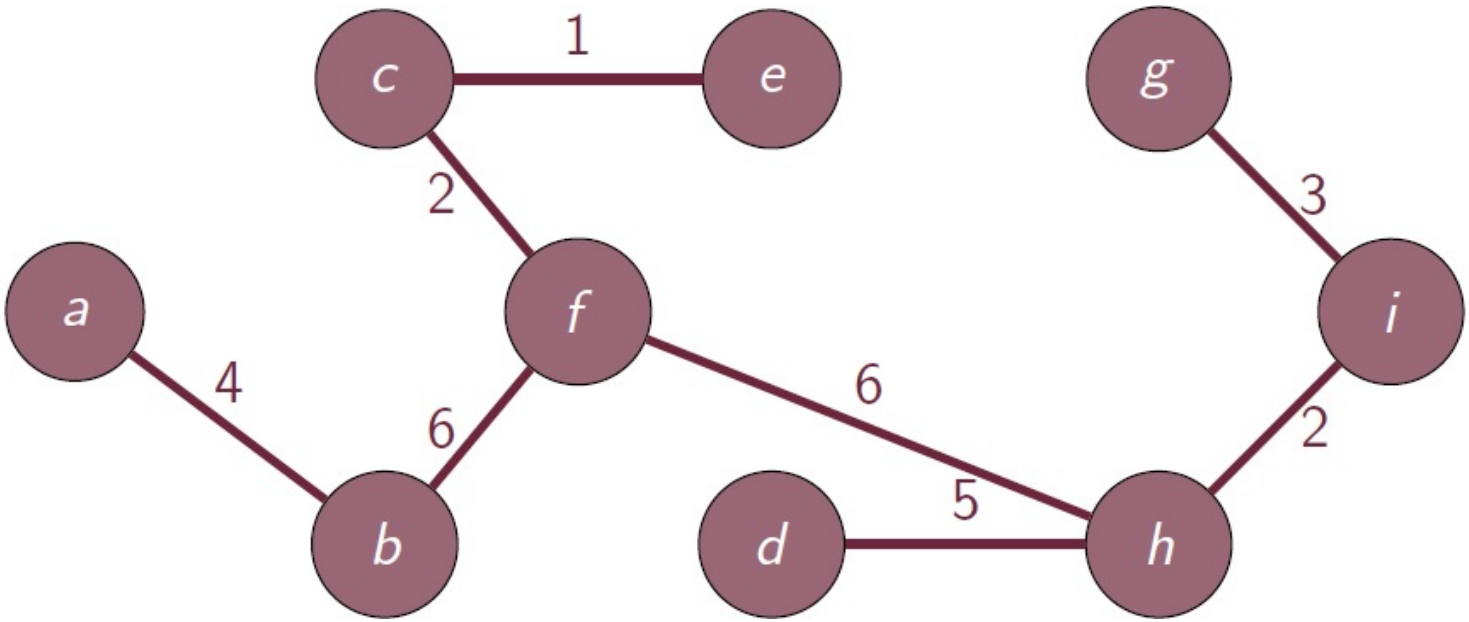
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Kruskal: Ejemplo II

Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim I

- Al igual que el algoritmo Kruskal, el algoritmo Prim (1957) es un algoritmo voraz para resolver problemas MST.
- La idea principal es mantener los conjuntos, uno para los vértices ya incluidos en el árbol y otro para los vértices restantes.
- Existe una conexión entre los vértices de cada conjunto.
- La complejidad es de $O(n^2)$.

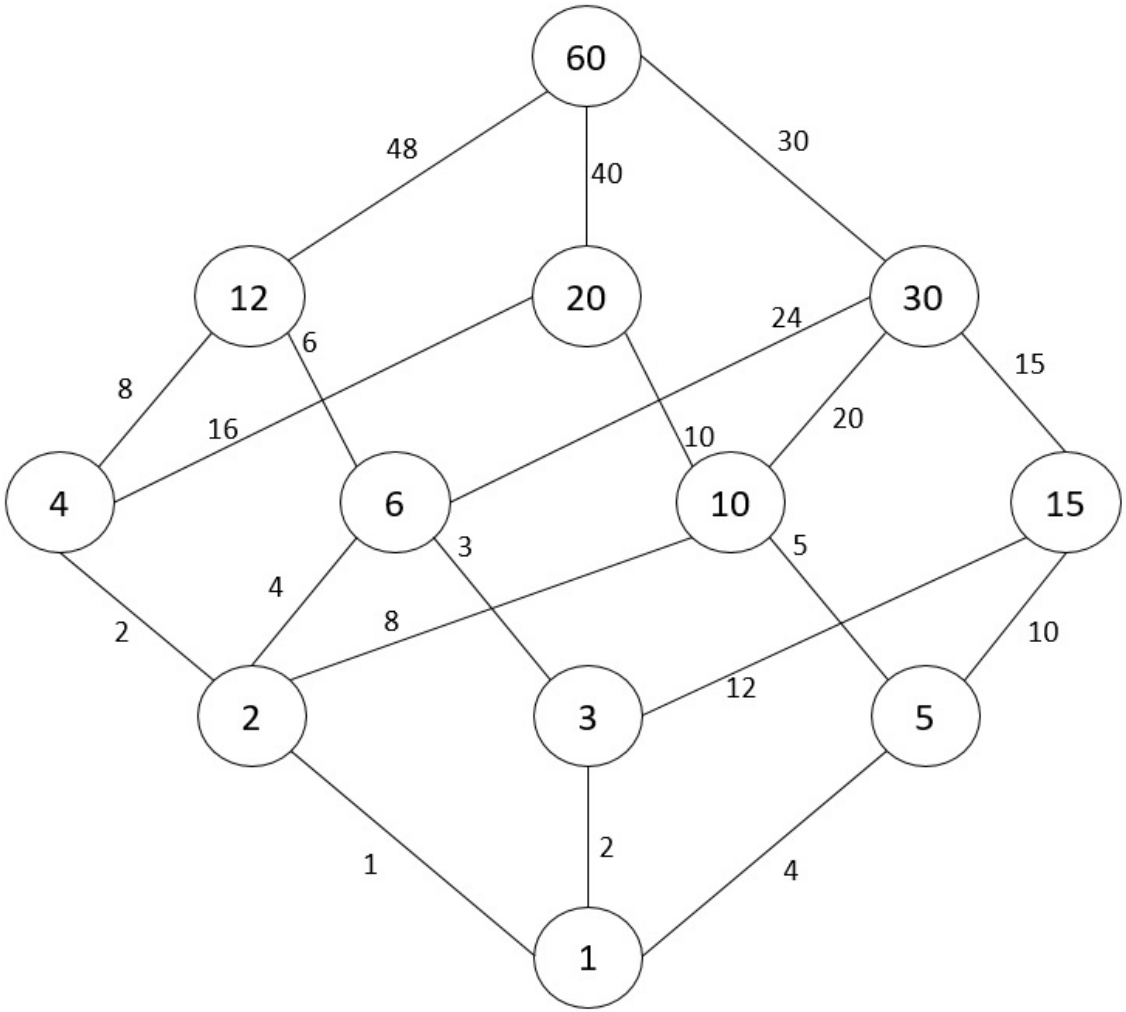
Algoritmo Prim II

Algorithm Prim's algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Choose $v \in V(G)$
3: Set $T \leftarrow v$
4: Set $l_w \leftarrow \infty : w \in V(G) \setminus v$
5: **while** $V(T) \neq V(G)$ **do**
6: **for** $e = \{v, w\} \in E(\{v\}, V(G) \setminus V(T))$ **do**
7: **if** $c(e) < l_w < \infty$ **then**
8: Set $l_w \leftarrow c(e)$ and update the edge
9: **else if** l_w is ∞ **then**
10: $l_w \leftarrow c(e)$ and insert the edge
11: **end if**
12: **end for**
13: **end while**
14: **return** The MST

Algoritmo Prim: Ejemplo I

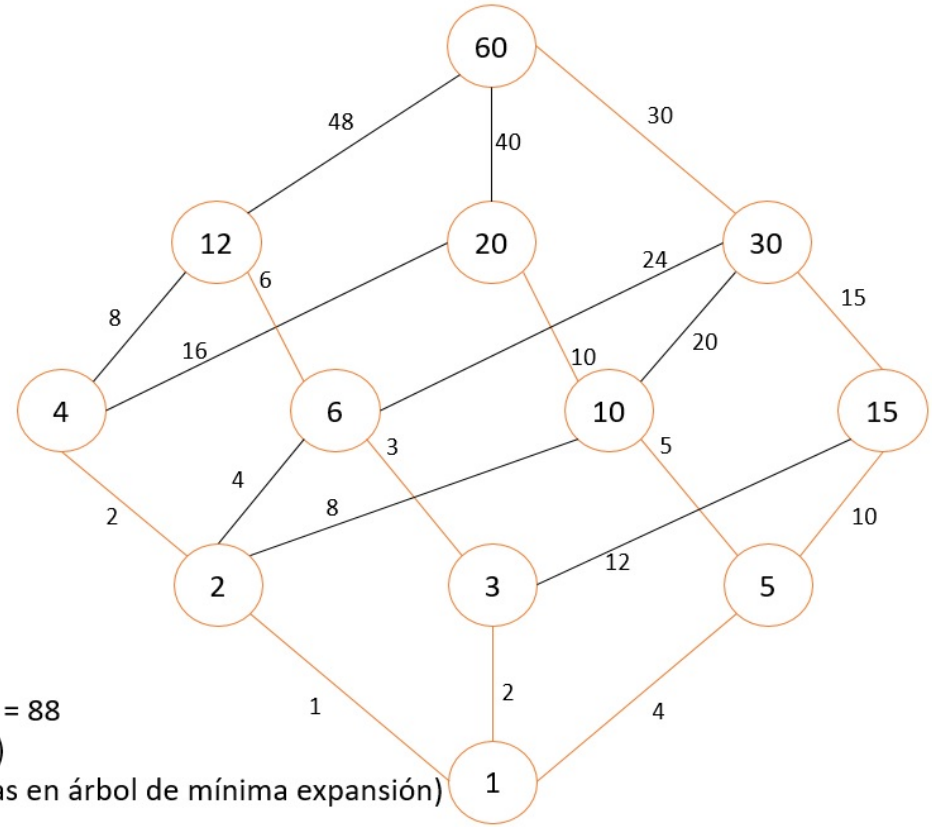
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo I

Obtener el árbol de mínima expransión del siguiente grafo.

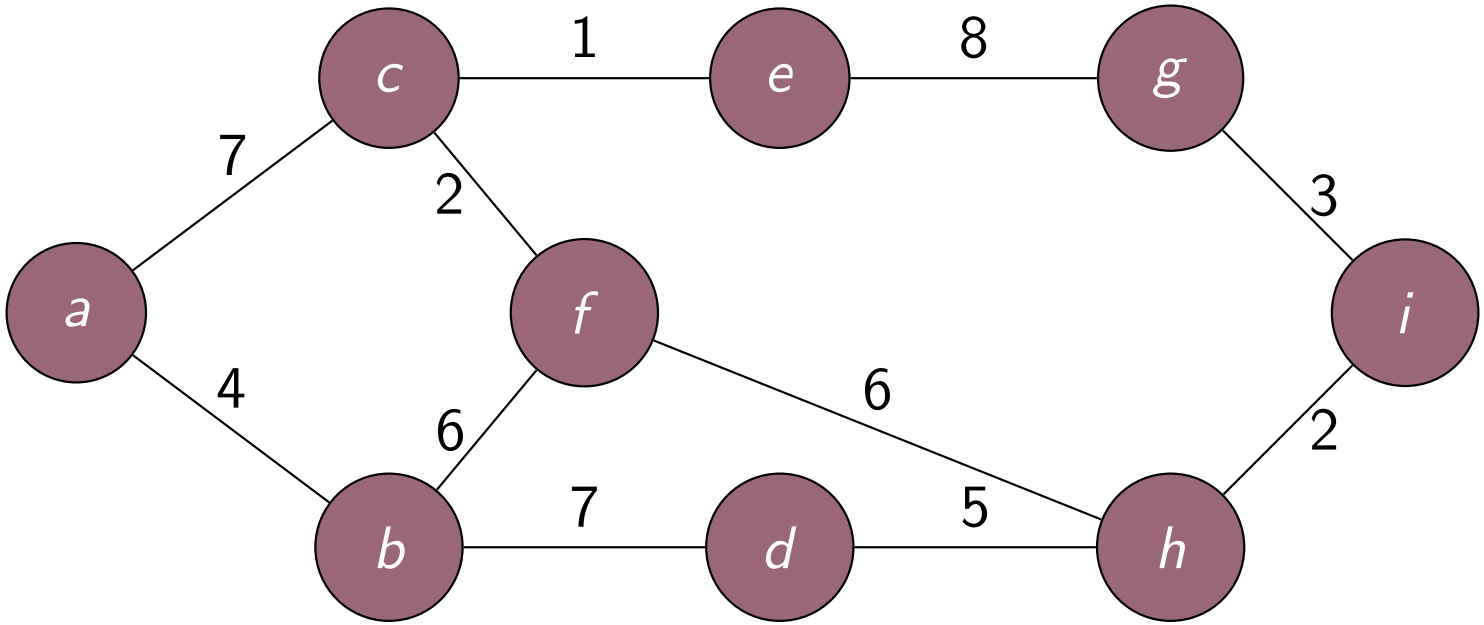


Flujo mínimo = 88
n= 12 (nodos)
n-1=11 (aristas en árbol de mínima expansión)

CIMAT

Algoritmo Prim: Ejemplo II

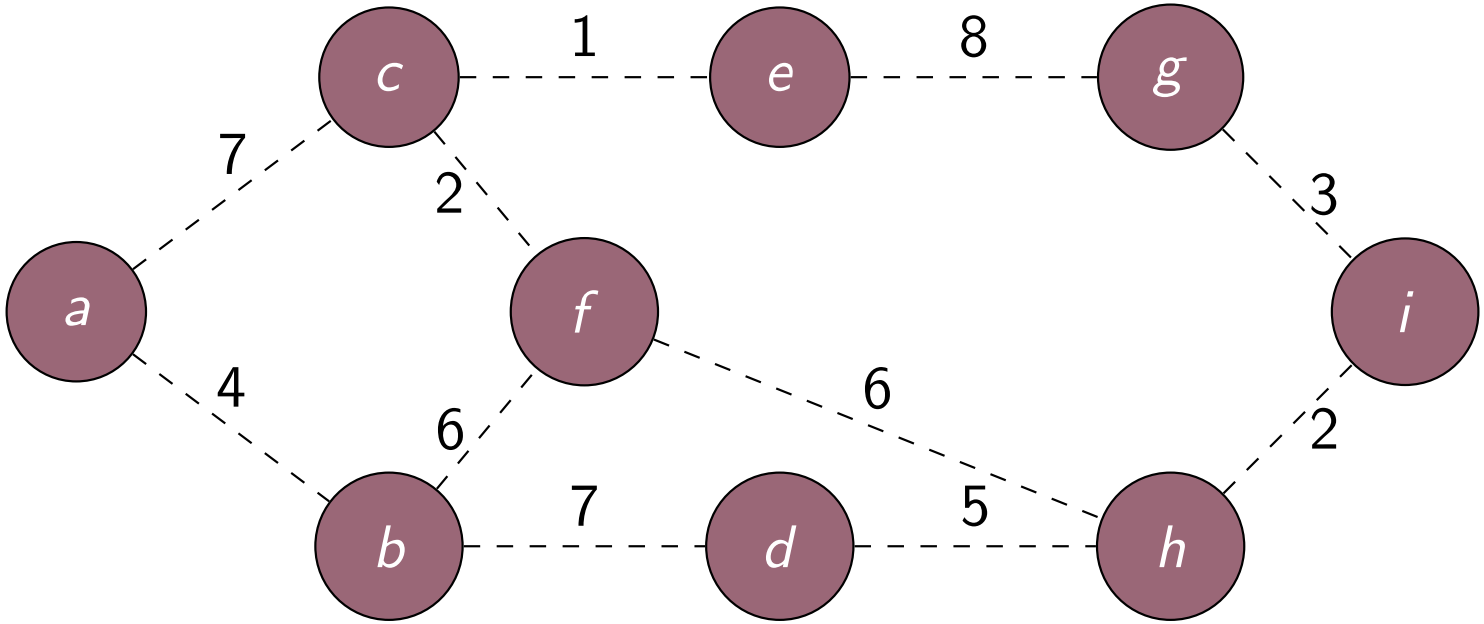
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

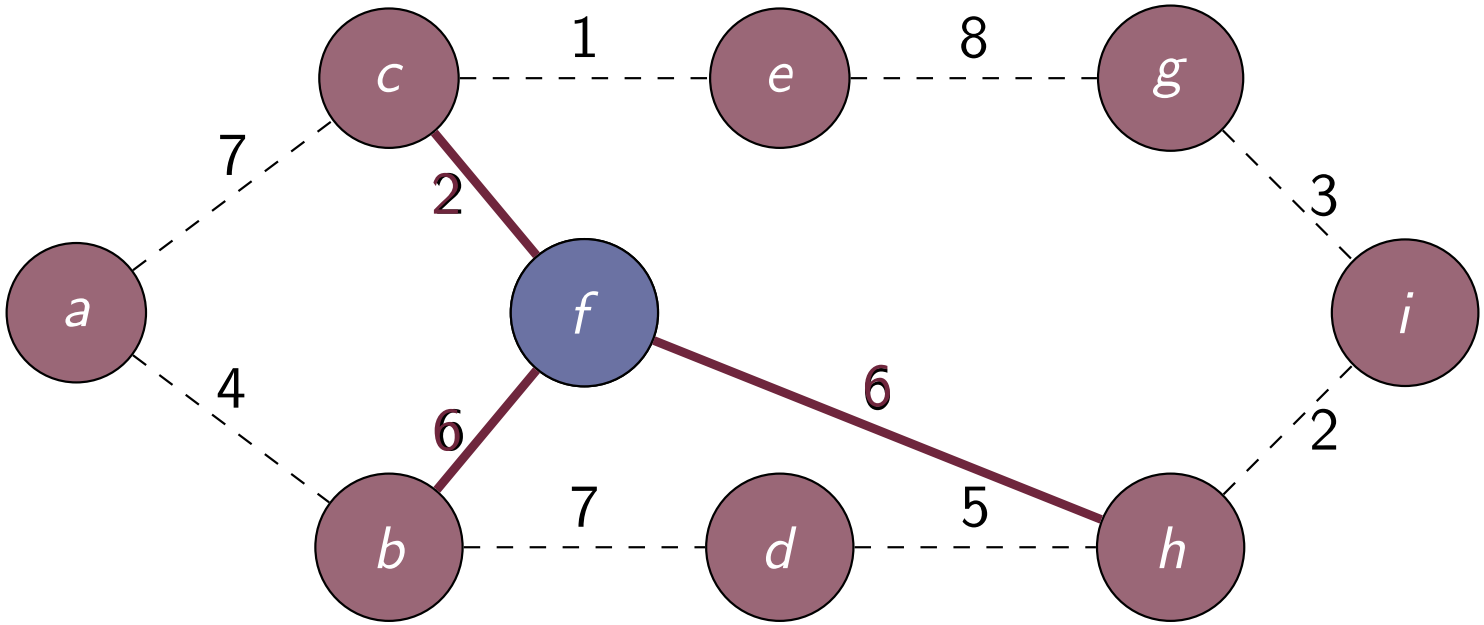
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

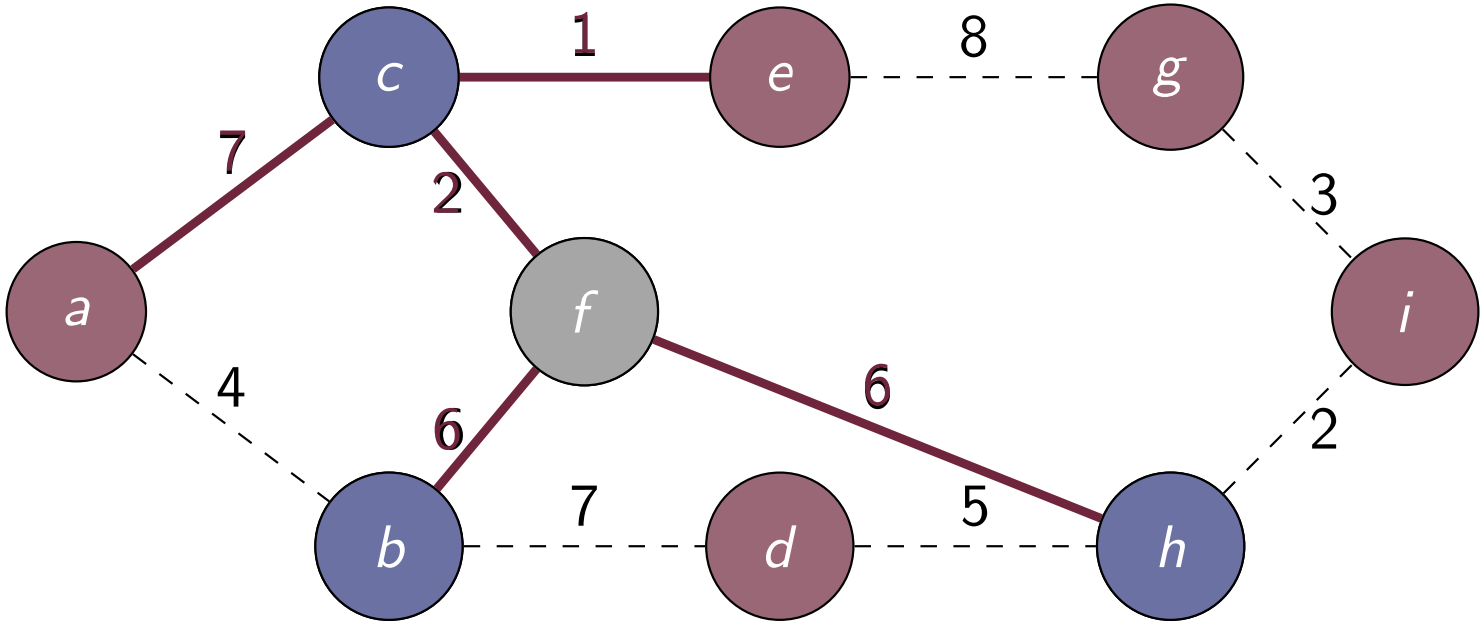
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

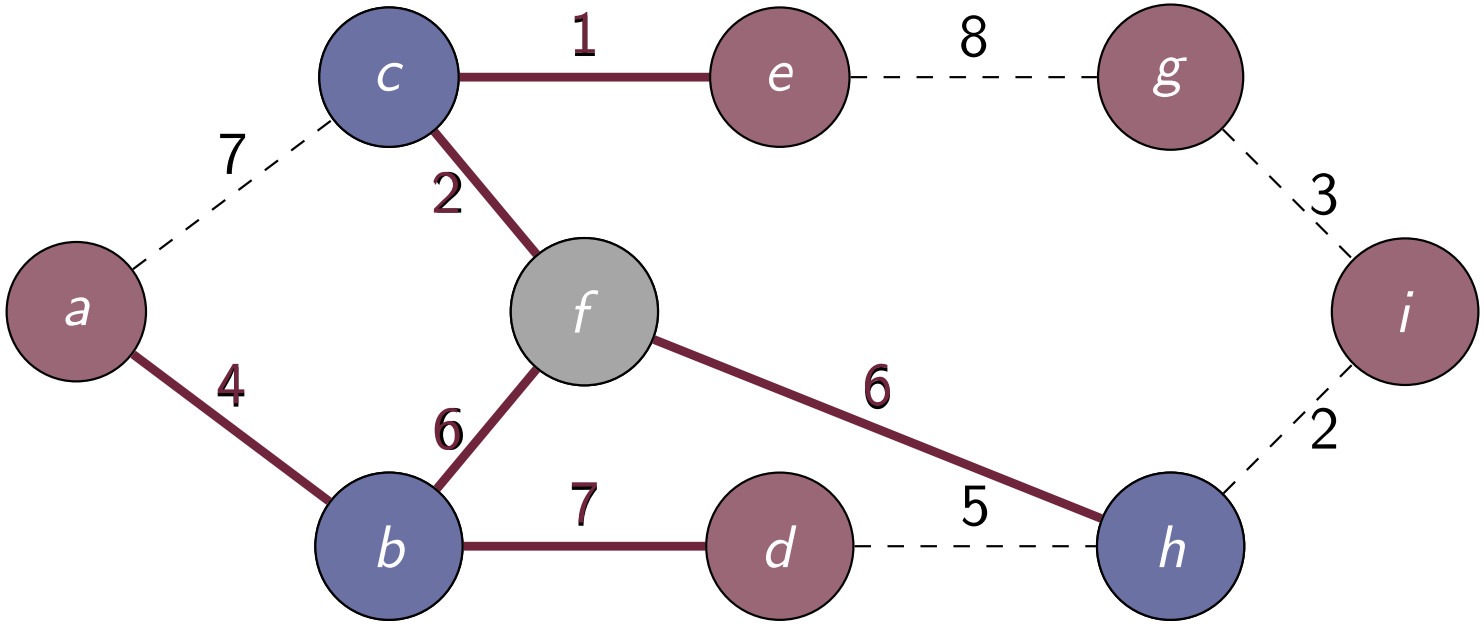
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

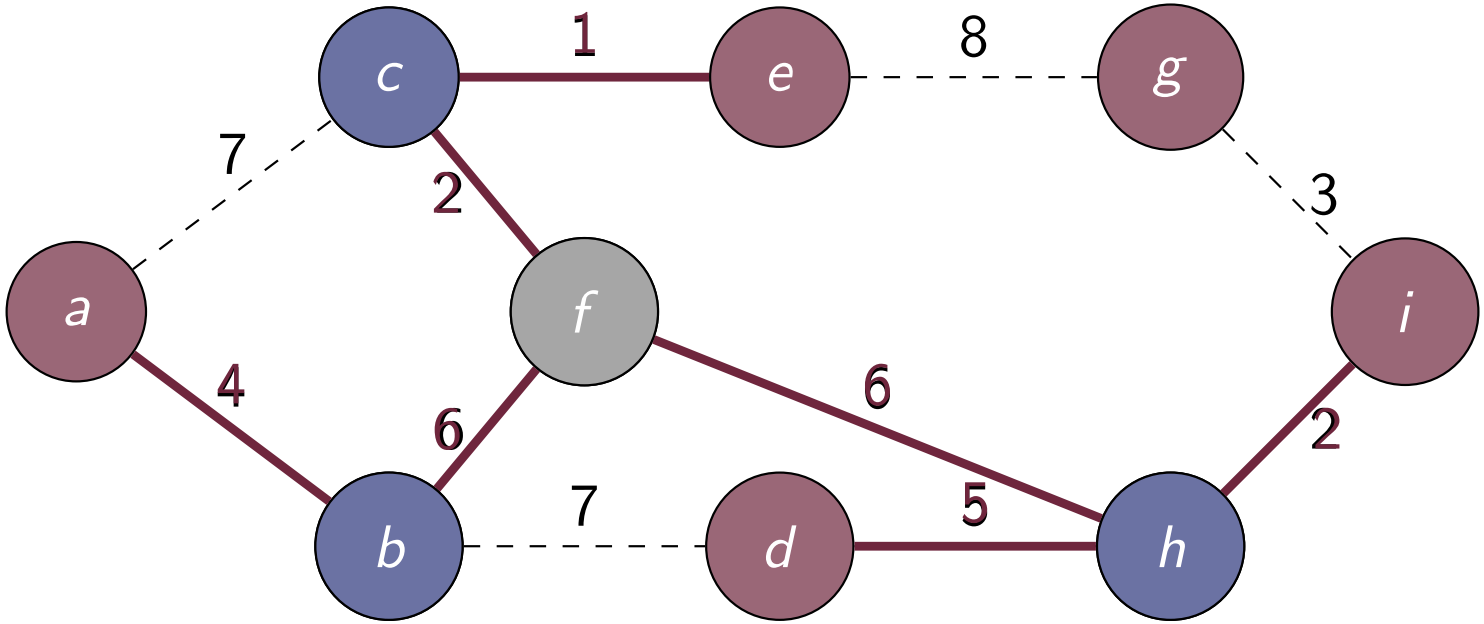
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

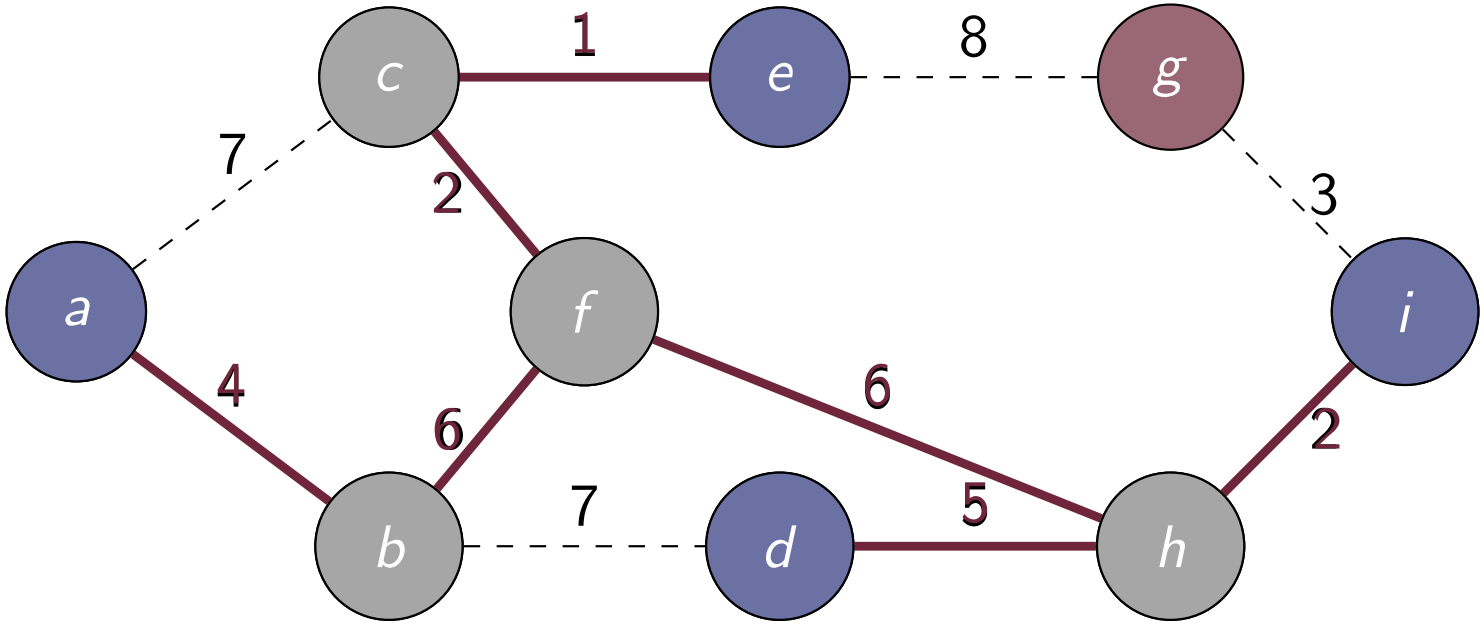
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

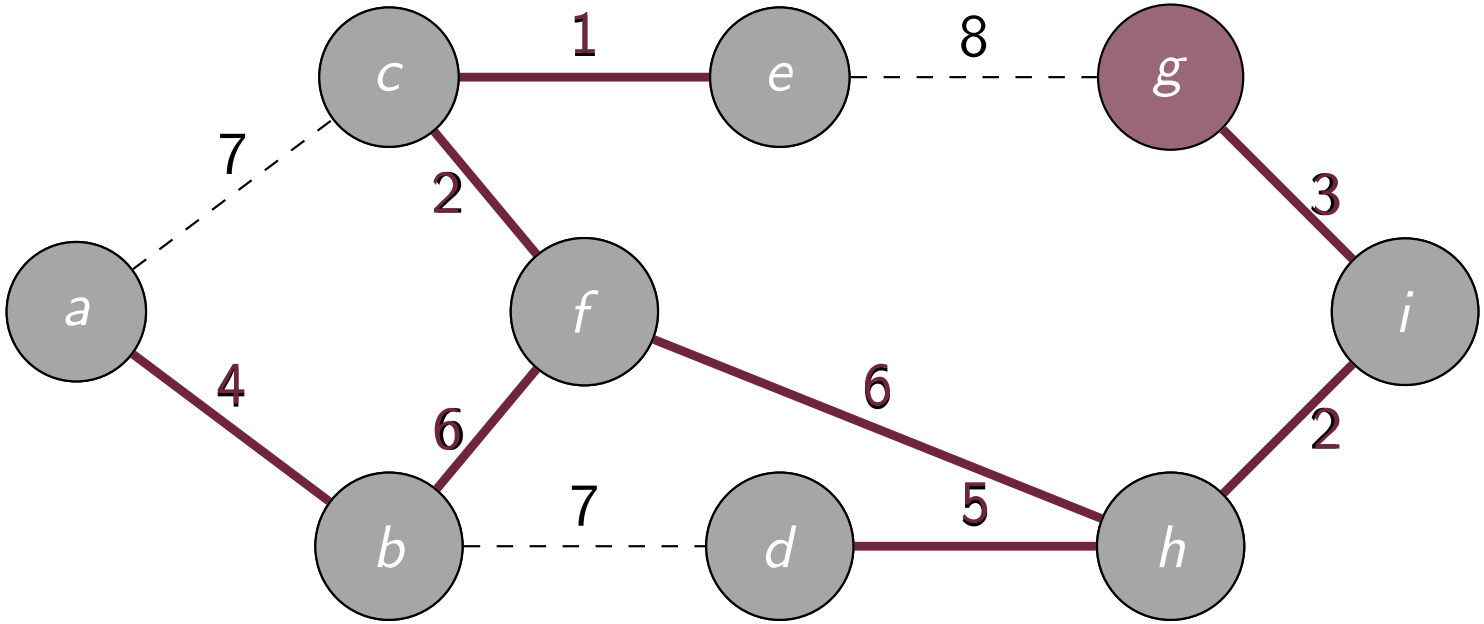
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

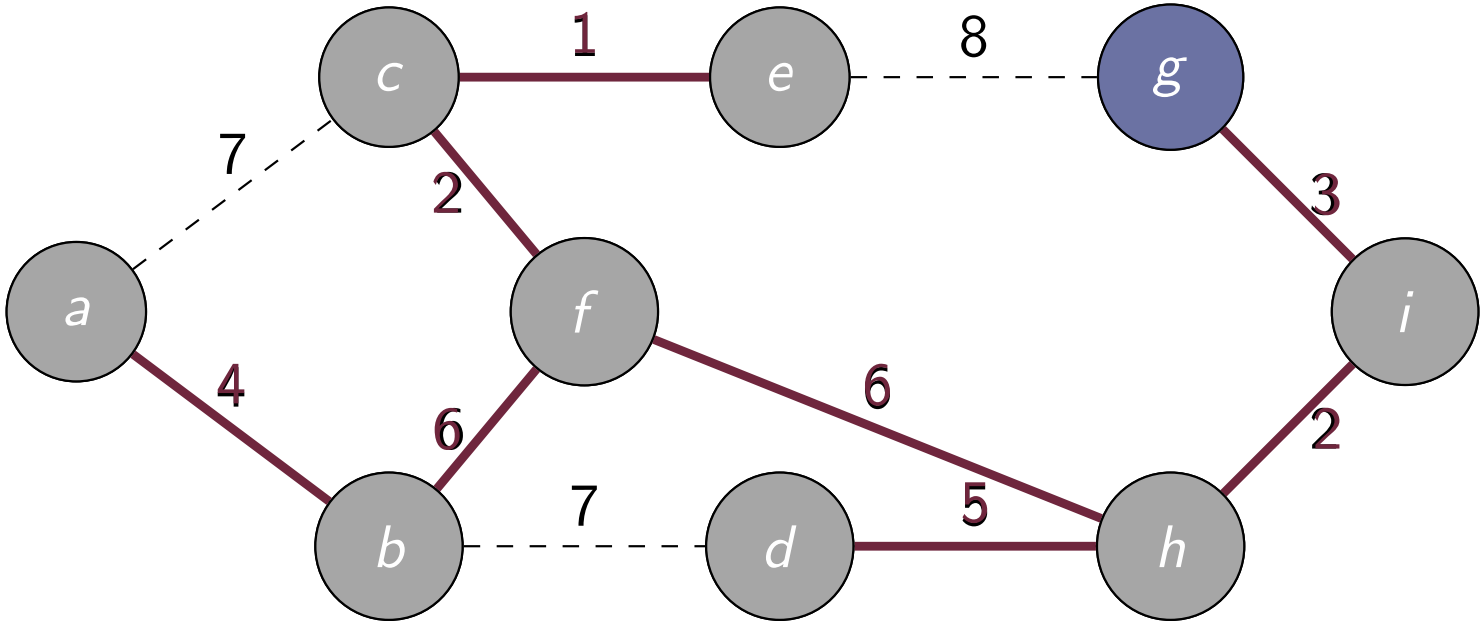
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

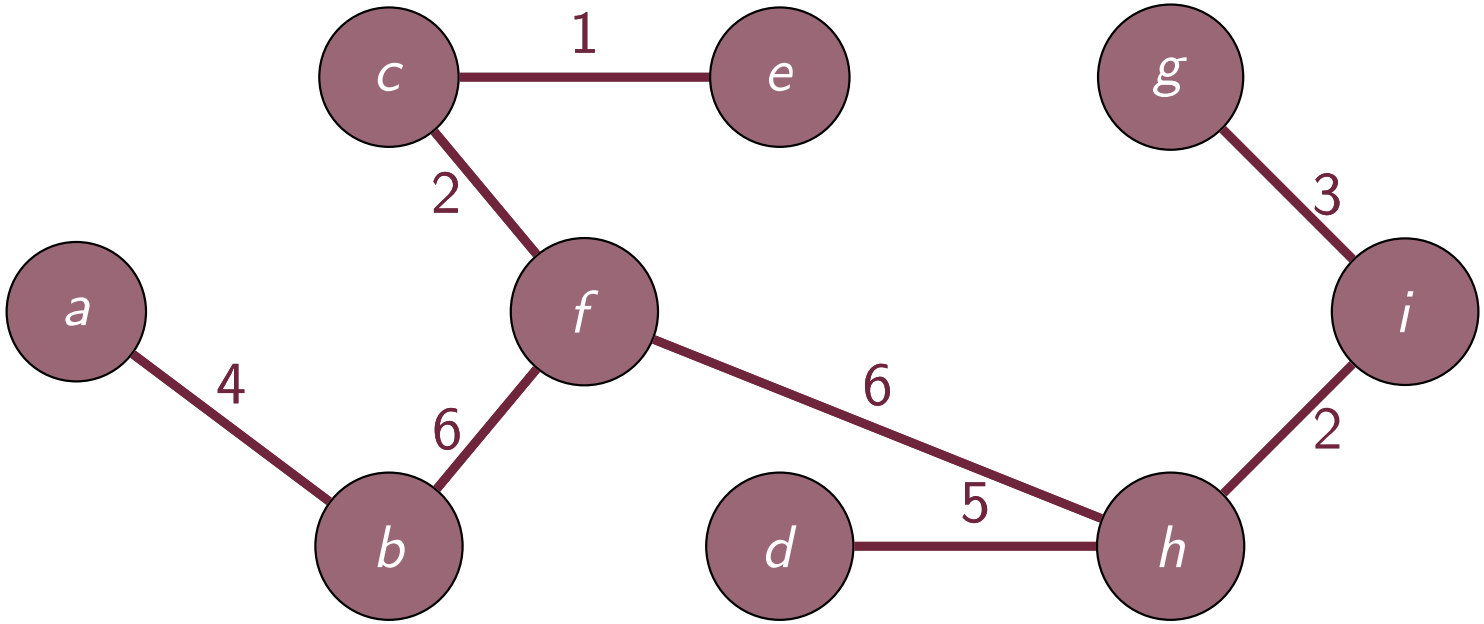
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Prim: Ejemplo II

Obtener el árbol de mínima expransión del siguiente grafo.

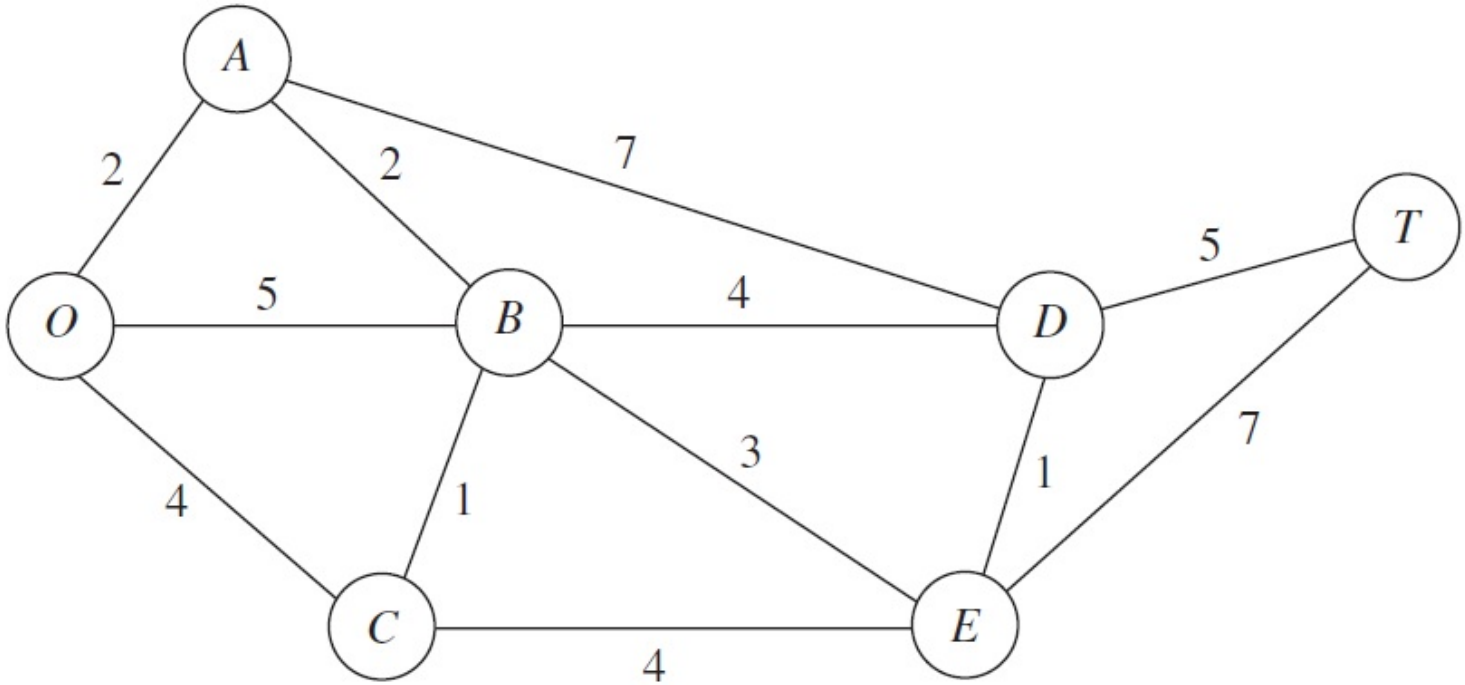


CIMAT

Ejercicio Seervada Park

Considere el problema anterior de Seervada Park. La administración de Seervada Park debe determinar los caminos bajo los cuales se deben tender las líneas telefónicas para conectar todas las estaciones con una longitud total mínima de cable.

Ejercicio Seervada Park



CIMAT

Algoritmo Eliminación-Inversa I

- Este método puede considerarse como la versión inversa del algoritmo Kruskal.
- A diferencia del algoritmo Kruskal, el algoritmo de Eliminación-Inversa comienza con el conjunto de vértices y aristas y, en cada iteración, se elimina una arista.

Algoritmo Eliminación-Inversa II

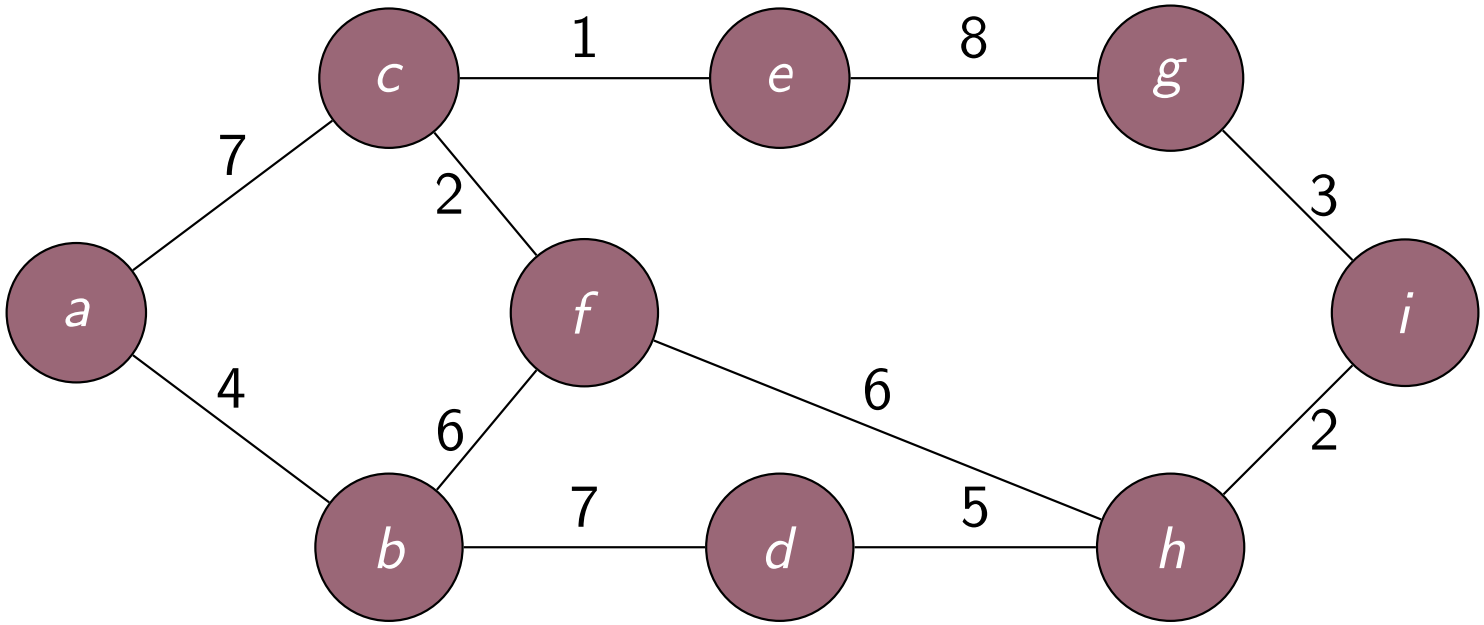
Algorithm Reverse-delete algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Sort the edges $e \in E$ in decreasing order
3: Set $T \leftarrow (V(G), E(G))$
4: **while** $|E(T)| > |V| - 1$ **do**
5: **if** $T + e_i$ does not produce disconnected graph **then**
6: $T \leftarrow T \setminus e_i$
7: **end if**
8: **end while**
9: **return** The MST

CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

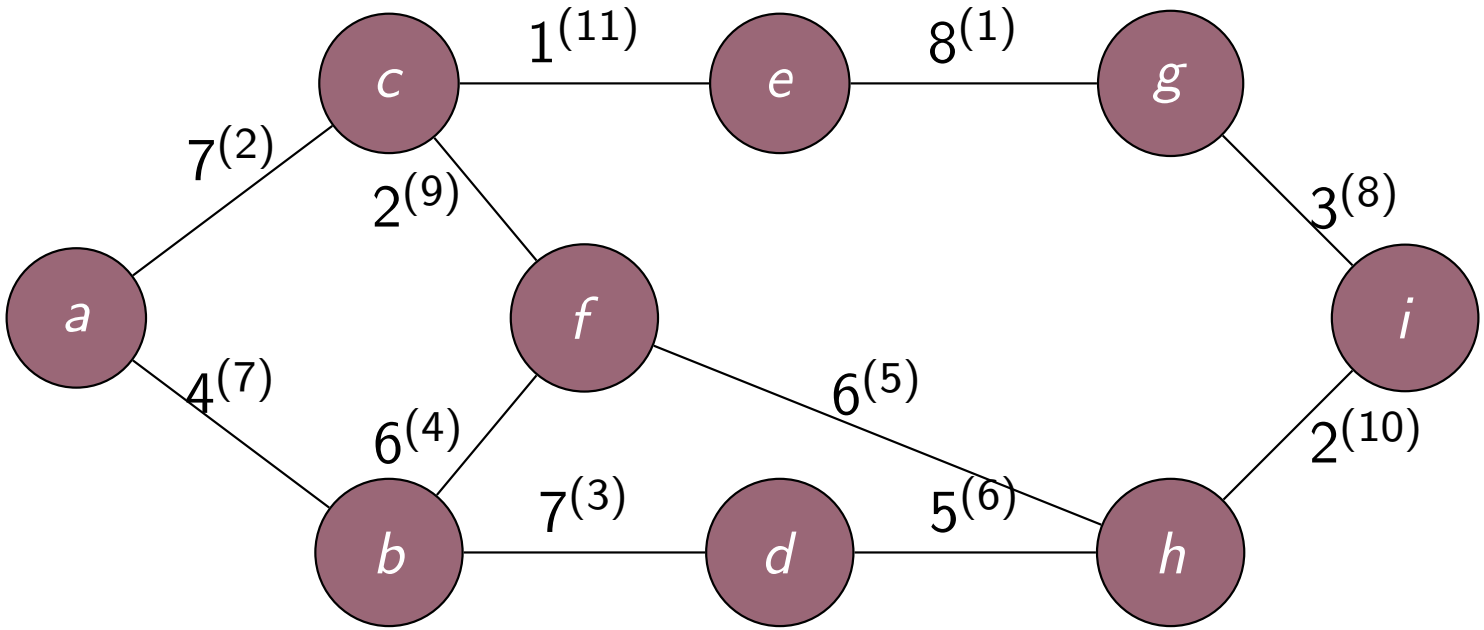
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

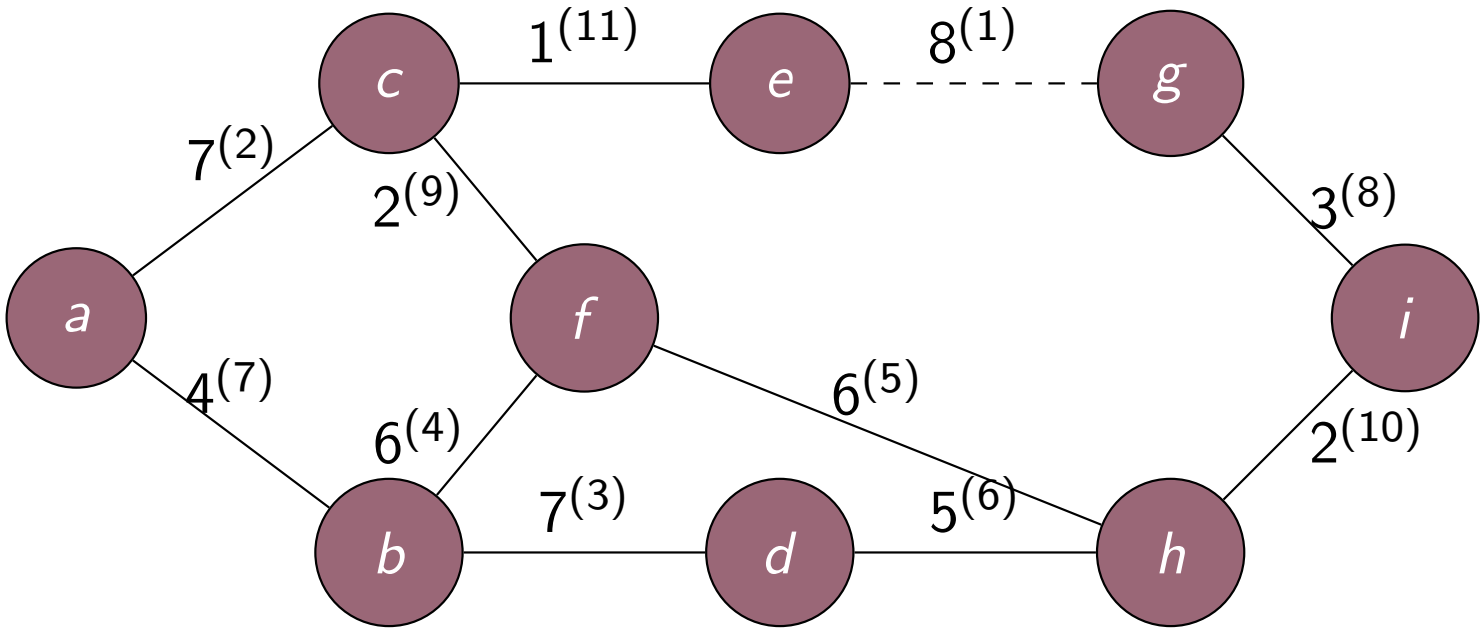
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

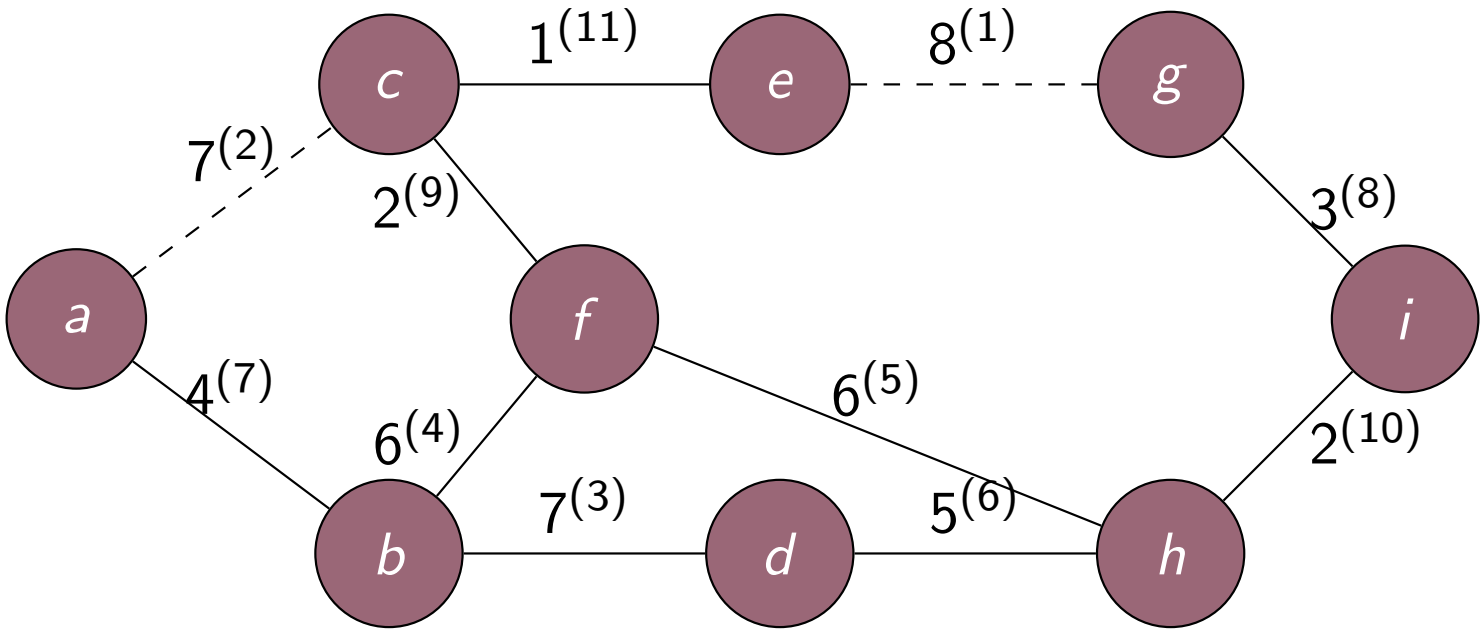
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

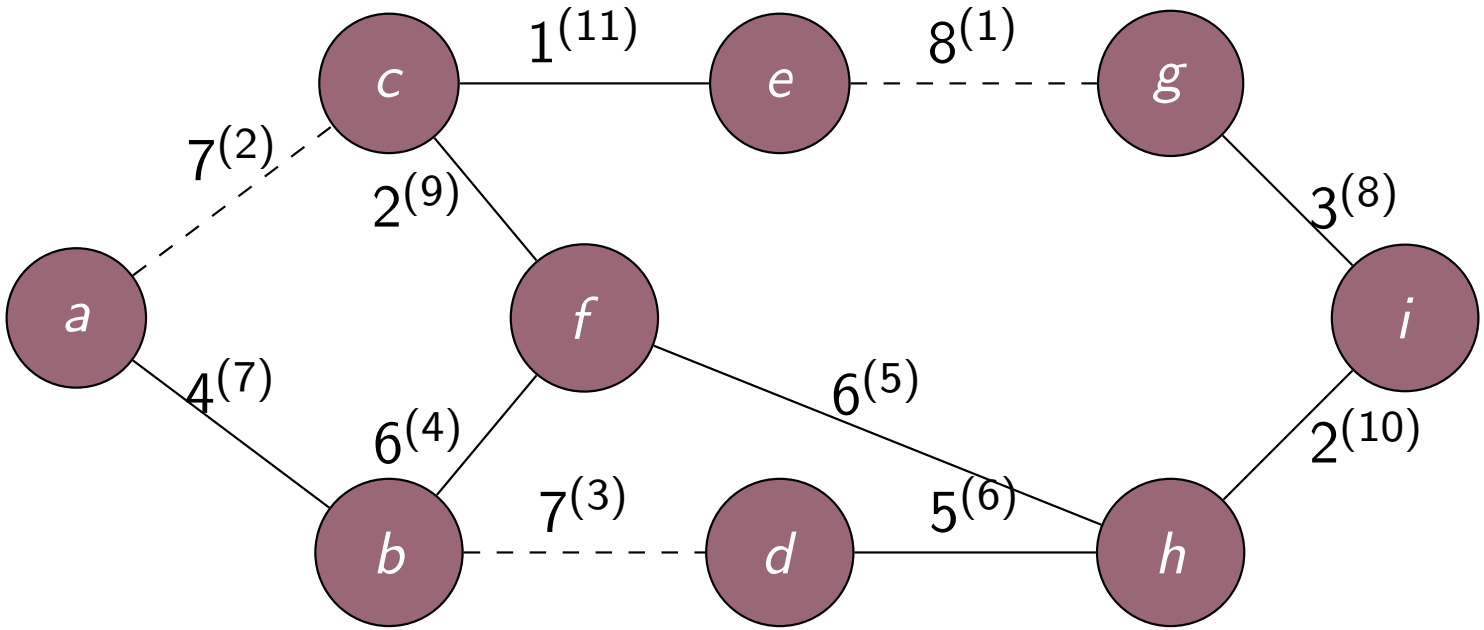
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

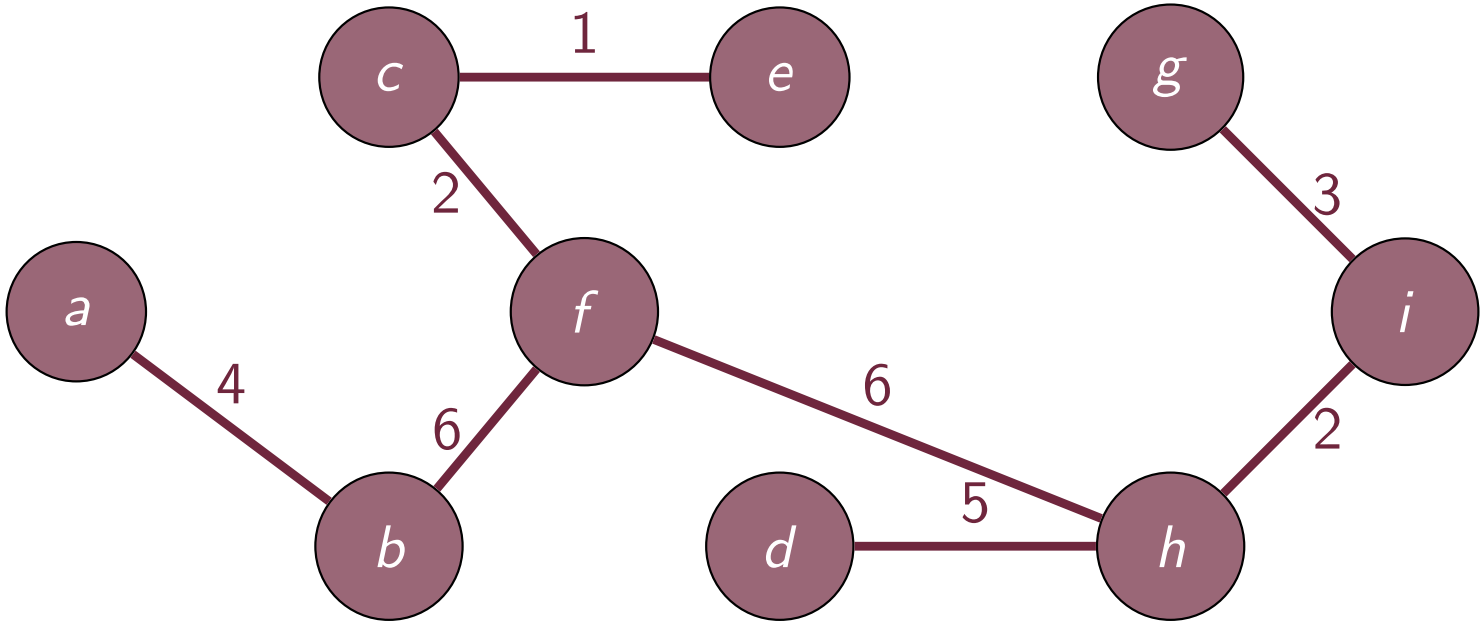
Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Algoritmo Eliminación-Inversa : Ejemplo

Obtener el árbol de mínima expransión del siguiente grafo.



CIMAT

Flujo en redes I

- La principal motivación de este tipo de problema es transportar de forma simultánea tantas unidades como sea posible desde s a t.
- Dado un grafo dirigido $G = (V, E)$ cada arista está asociada con su capacidad $c(e) > 0$. y existe un nodo origen (fuente) s y un nodo destino t, tal que $s \neq t$.
- Una solución a este problema será llamado un **flujo máximo**.

Flujo en redes II

Descripción del problema

El problema de flujo máximo es aquel en el que tenemos una red de comunicaciones y deseamos enviar la mayor cantidad de producto desde un punto llamado origen a otro punto llamado destino, utilizando las conexiones de dicha red pero sin exceder ciertos límites de capacidad que tienen las conexiones.

CIMAT

Flujo en redes III

Variables de decisión:

- x_a es la variable continua asociada a cada arco $a \in A$ que representa el valor que el flujo x asocia al arco a.
- z la variable continua que representa la capacidad de dicho flujo.

Flujo en redes IV

Función objetivo: se requiere maximizar la cantidad de flujo de la red, es decir:

$$\text{maximizar } z$$

Flujo en redes V

Restricciones:

- Conservación de flujo:

$$\sum_{a\in\delta^+(i)} x_a - \sum_{a\in\delta^-(i)} x_a = \begin{cases} z, & \text{si } i = s; \\ -z, & \text{si } i = t; \\ 0, & \text{si } i \in N - \{s, t\}. \end{cases}$$

- Dominio de las variables y restricción de capacidad:

$$0 \leq x_a \leq b_a \quad \forall a \in A$$

Flujo en redes VI

- Este problema tiene numerosas aplicaciones, por ejemplo:
  - Problema de asignación de trabajo.
  - Problema de ruteo de vehículos.
- Algunos algoritmos para resolver estos problemas son:
  - Ford-Fulkerson.
  - Edmonds-Karp.
  - Dinic.

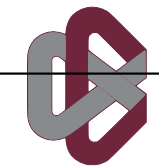
Algoritmo Ford-Fulkerson I

- Se basa en la idea de encontrar rutas de aumento.
- Una ruta de aumento es un ruta con capacidad disponible.
- Por lo tanto, la idea principal es que si hay una ruta de aumento de s a t, entonces se envía un flujo a lo largo de una de las rutas.

Algoritmo Ford-Fulkerson II

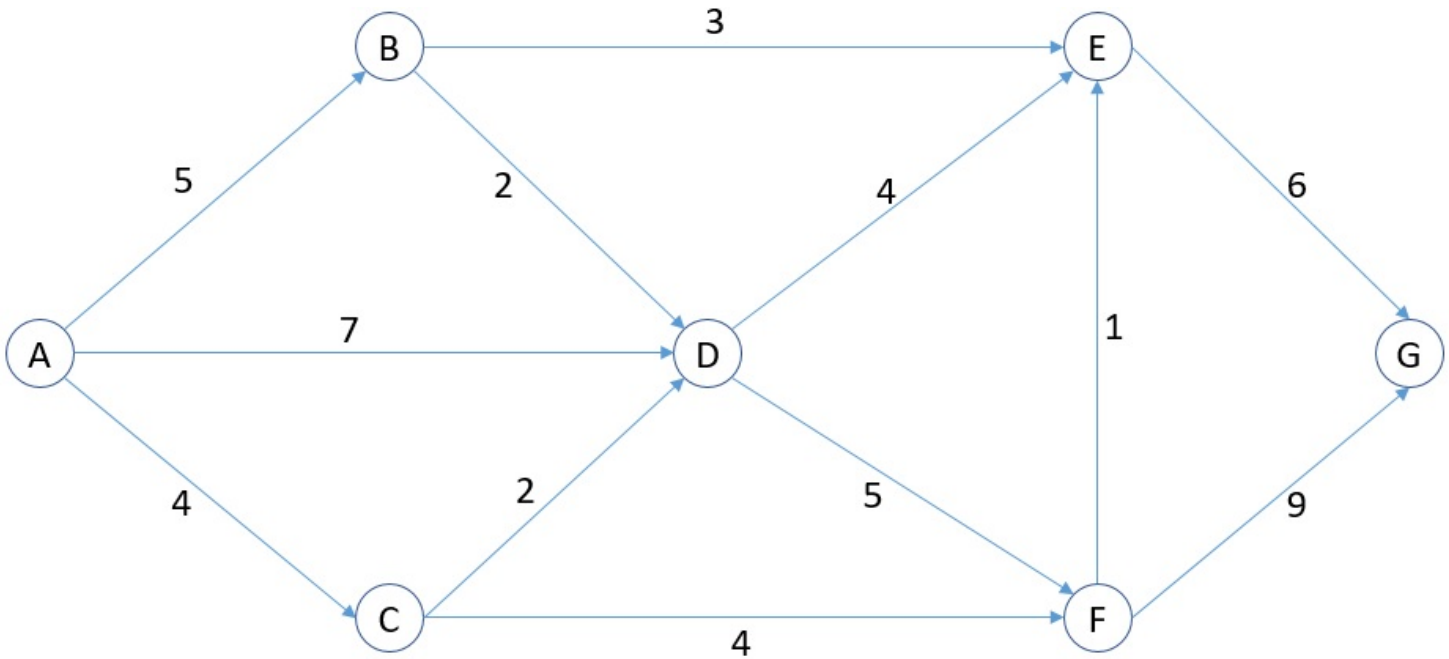
Algorithm Fold-Fulkerson algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Set $f(i,j) \leftarrow 0 : (i,j) \in E(G)$
3: **while** there is an augmentation path from s to t **do**
4: Find the augmentation path (u,v) with the minimum capacity, c_f
5: **for** each edge (u,v) in the augmentation path **do**
6: Increase $f(u,v)$ by c_f
7: Decrease $f(v,u)$ by c_f
8: **end for**
9: **end while**
10: **return** The maximum flow



Algoritmo Ford-Fulkerson: Ejemplo

Determine el flujo máximo del nodo A al nodo G de la sigiente red.



CIMAT

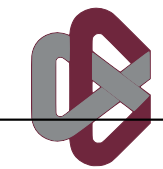
Algoritmo Edmonds-Karp I

- Está basado en el algoritmo Fold-Fulkerson.
- La principal diferencia es que la ruta de aumento está determinada por la amplitud-primer-búsqueda (breadth-first-search).

Algoritmo Edmonds-Karp II

Algorithm Edmonds-Karp algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Set $f(i,j) \leftarrow 0 : (i,j) \in E(G)$
3: **while** there is an augmentation path from s to t **do**
4: Determine the augmentation path using BFS
5: Find the augmentation path (u,v) with the minimum capacity, c_f
6: **for** each edge (u,v) in the augmentation path **do**
7: Increase $f(u,v)$ by c_f
8: Decrease $f(v,u)$ by c_f
9: **end for**
10: **end while**
11: **return** The maximum flow



Algoritmo Dinic I

- Se basa en la idea de encontrar una ruta de aumento.
- Define un grafo que es encontrado usando BFS.
- A diferencia del algoritmo Edmonds-Karp, el algoritmo de Dinic procesa un bloque de flujo.
- Generalmente tiene un costo computacional más bajo que el algoritmo Edmonds-Karp.

Algoritmo Dinic II

Algorithm Dinic's algorithm

1: Let G be the graph, $V(G)$ the set of vertices in graph G, $E(G)$ the set of edges
2: Set $f(i,j) \leftarrow 0 : (i,j) \in E(G)$
3: **while** there is an augmentation path from s to t **do**
4: Construct a graph level, G_L, using BFS
5: Find a blocking flow in G_L
6: Add the augmented flow and updates available capacity
7: **end while**
8: **return** The maximum flow

Aplicaciones I

- Una aplicación del MST es en agrupamientos.
- Dado un conjunto de datos con n muestras y una matriz de similitud $n \times n$, la idea en agrupamiento es dividir los n elementos en grupos de k de tal manera que se maximice la disimilitud entre los elementos de diferentes grupos.

Aplicaciones II

Idea:

- Representar el problema como una grafo, donde cada muestra es un vértice y dos vértices están conectados por una arista ponderada por su similitud (inversa).
- Mantener los agrupamientos como un conjunto de componentes conectados del grafo.
- Iterativamente combinar los agrupamientos que contienen los dos elementos más cercanos agregando un arista entre ellos.
- Detenerse cuando existan k agrupamientos.

Preguntas?