Grafos

Búsqueda de caminos

Mejor Camino

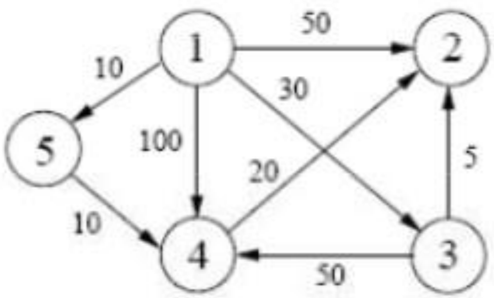
Árbol de expansión mínimo

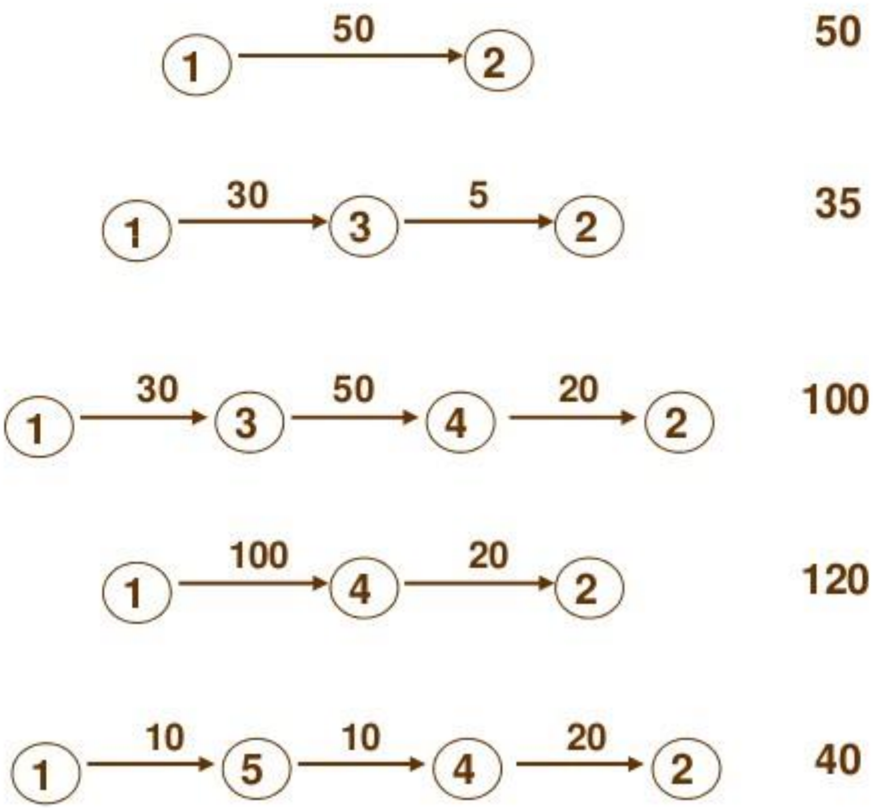
Búsqueda de caminos

- El recorrido DFS, se puede adaptar para:
  - Buscar un camino entre dos vértices
  - Buscar todos los caminos entre un par de vértices
  - Buscar el mejor camino entre dos vértices (camino mas corto o que sume menos costo)

Todos los caminos de 1 a 2 con sus costos

Ejemplo:





Repaso de recorrido DFS

dfs(origen):

1.- Marcar origen como visitado

2.- Para cada “adyacente” a origen:

3.- Si “adyacente” no esta visitado:

4.- dfs(adyacente)

Adaptación del DFS para buscar un camino

```
buscar_1_camino(origen, destino, camino):
    1.- Agregar origen al camino
    2.- Marcar origen como visitado
    3.- Si (origen == destino):
        4  print camino
        5  return True
    6.- Si no:
        7.- Para cada "adyacente" a origen:
            8.- Si "adyacente" no esta visitado:
                9.- ok = buscar_1_camino(adyacente,  destino, camino)
                10.- if(ok):
                    11.- return true
                12.- Quitar ultimo vértice (adyacente) de camino
```

Adaptación del DFS para buscar todos los caminos

buscar_todos(origen, destino, camino, resultado):

1.- Agregar origen al camino

2.- Marcar origen como visitado

2.- Si (origen == destino):

    3- Agregar camino a resultado

4.- Si no:

    5.- Para cada “adyacente” a origen:

        6.- Si “adyacente” no esta visitado:

            7.- buscar_todos(adyacente, destino, camino, resultado)

            8.- Desmarcar adyacente como visitado

            9.- Quitar ultimo vértice (adyacente) de camino

DFS para buscar el mejor camino

buscar_mejor(origen, destino, camino, mejorCamino):

1.- Agregar origen al camino

2.- Marcar origen como visitado

3.- Si (origen == destino):

  4- Si camino es mejor que mejorCamino

    5.- Copiar camino → mejorCamino

6.- Si no:

  7.- Para cada “adyacente” a origen:

    8.- Si “adyacente” no esta visitado:

      9.- buscar_mejor(adyacente, destino, camino, mejorCamino)

      10.-Desmarcar adyacente como visitado

      11.- Quitar ultimo vértice (adyacente) de camino

Caminos de costo mínimo

- El algorítmo de Dijkstra permite calcular los caminos de costo mínimo desde un vértice de origen dado a todos los restantes vértices del grafo.
- Consideraciones: las aristas no pueden ser de costo negativo.

Estructura complementaria usada por el algoritmo de Dijkstra

El algorítmo mantiene una tabla auxiliar con la siguiente información por cada vértice “v” del grafo:

- Dv : distancia mínima desde el origen (inicialmente ∞ para todos lo vértices excepto el origen con valor 0)
- Pv : vértice por donde paso para llegar al vértice “v”
- Conocido : indica si ya fue procesado

Algoritmo de Dijkstra

Dijkstra:

1.- Mientra, haya vertices no conocidos:

2.- Elijo de la tabla auxiliar el vértice “v”, NO PROCESADO con MENOR DISTANCIA

3.- Marcar “v” como procesado

4.- Para todos los adyacentes a “v”, actualizar la distancia de los mismos en la tabla auxiliar

Algoritmo de Dijkstra (cont)

La actualización de la distancia de los adyacentes ***w*** se realiza con el siguiente criterio:

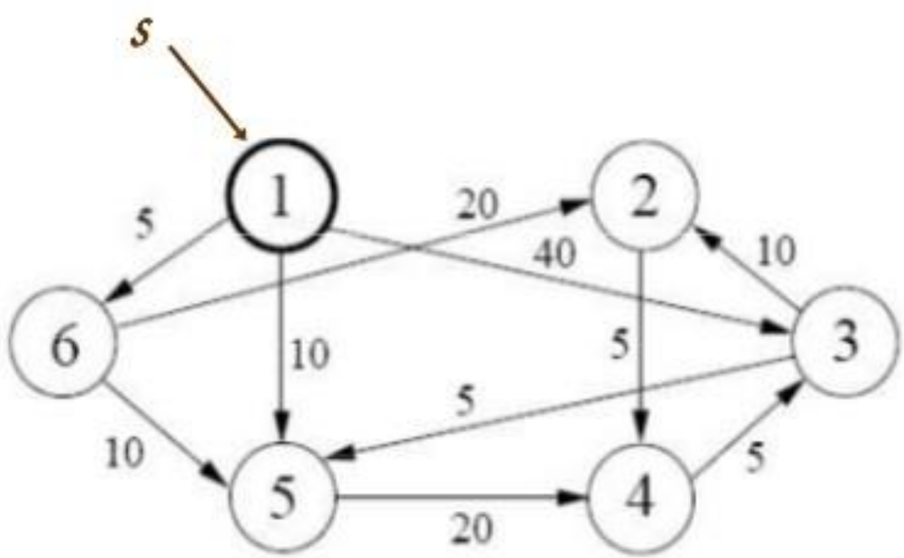
- Se compara D_w con $\underbrace{D_v + c(v,w)}$

Distancia de *s* a *w*
(sin pasar por *v*)

Distancia de *s* a *w*,
pasando por *v*

- Se actualiza si $D_w > D_v + c(v,w)$

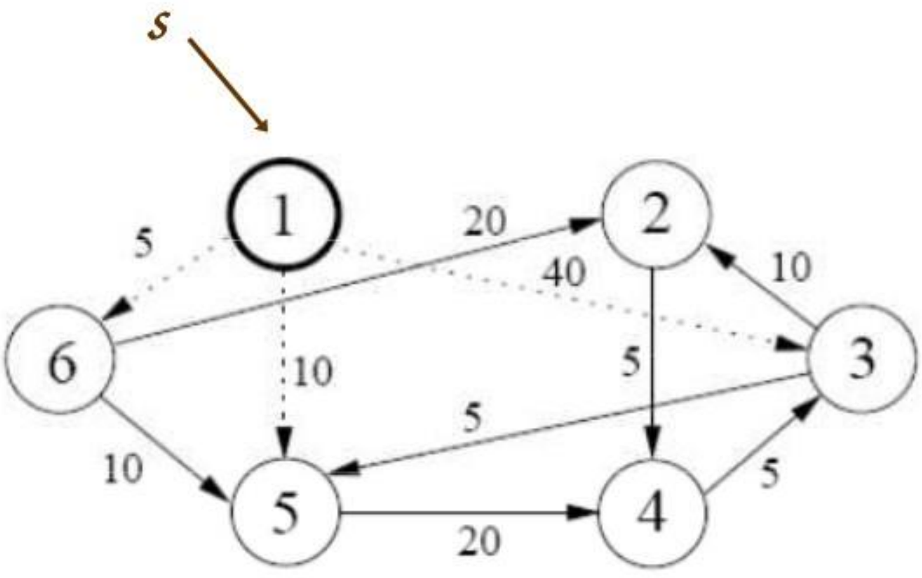
Ejemplo



Valores iniciales de la tabla

V	D_v	P_v	Conoc.
1	0	0	0
2	∞	0	0
3	∞	0	0
4	∞	0	0
5	∞	0	0
6	∞	0	0

Ejemplo (cont)

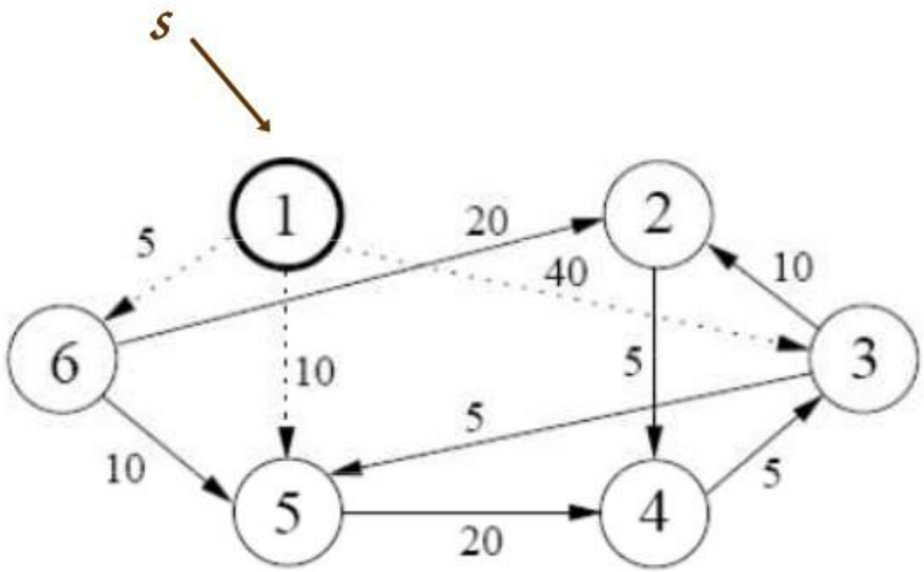


•Valores al seleccionar el vértice 1

•Actualiza la distancia de 3, 5 y 6

V	D_v	P_v	Conoc.
1	0	0	1
2	∞	0	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	0

Ejemplo (cont)

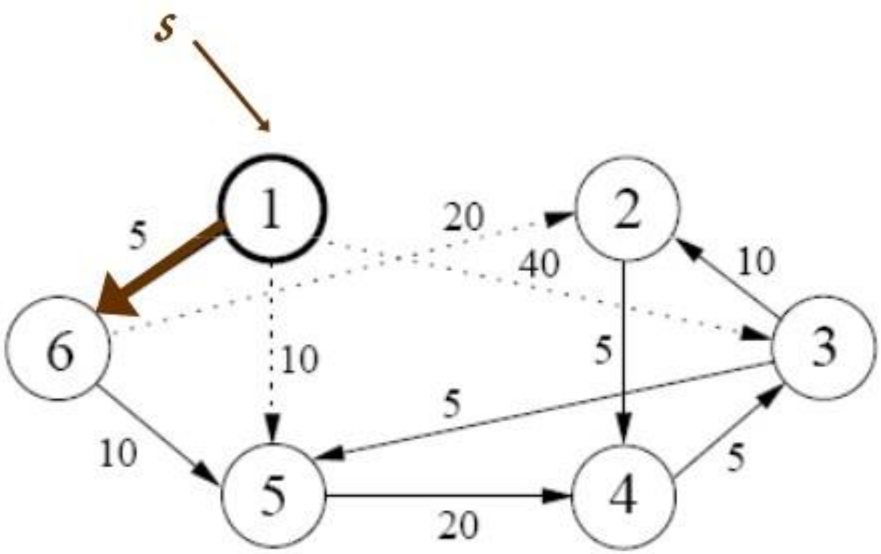


•Valores al seleccionar el vértice 1

•Actualiza la distancia de 3, 5 y 6

V	D_v	P_v	Conoc.
1	0	0	1
2	∞	0	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	0

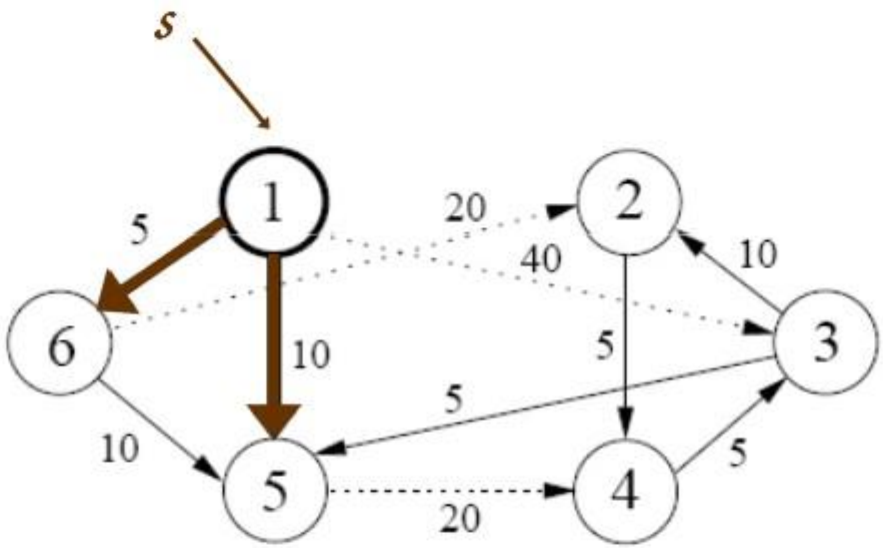
Ejemplo (cont)



- Valores al seleccionar el vértice 6
- Actualiza la distancia de 2

V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	1

Ejemplo (cont)



- Valores al seleccionar el vértice 5
- Actualiza la distancia de 4

V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	30	5	0
5	10	1	1
6	5	1	1

Ejemplo (cont)

- Los próximos vértices a elegir son: 2, 4 y 3 en ese orden.

El resultado final es:

V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	1
3	35	4	1
4	30	5	1
5	10	1	1
6	5	1	1

Recuperación de los mejores caminos

- Dijkstra permite obtener los mejores caminos desde un vértice hacia los demás.
- La información de cuales son estos caminos, está en la tabla auxiliar que se utilizó durante el algoritmo.
- Para obtener alguno de estos caminos hay que procesar el mismo, desde el final hacia el principio.
- … por ejemplo:

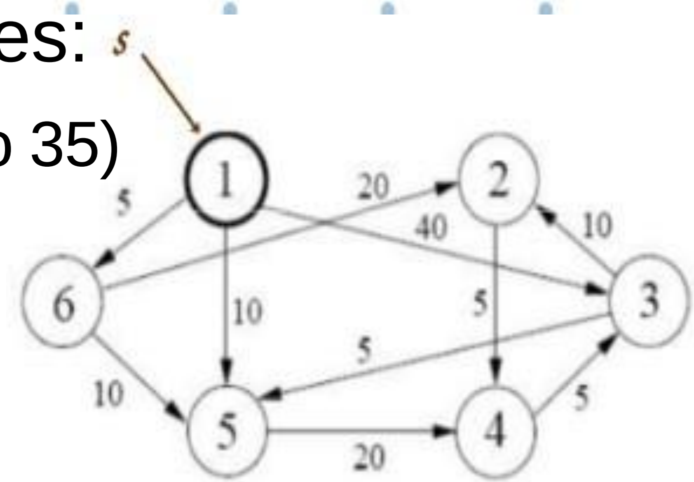
Recuperación de los mejores caminos (cont)

- Por ejemplo, el mejor camino desde el vértice 1 al vértice 3 es:
  - a 3 llego pasando por 4
  - a 4 llego pasando por 5
  - a 5 llego pasando por 1

V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	1
3	35	4	1
4	30	5	1
5	10	1	1
6	5	1	1

Por lo tanto el mejor camino a 3 es:

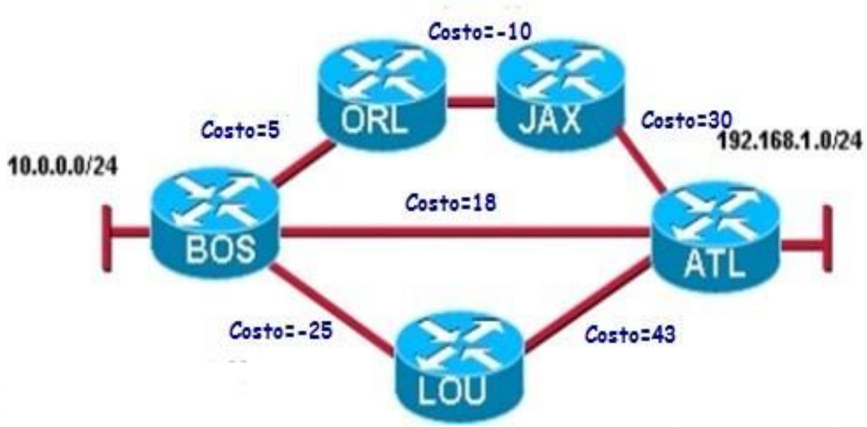
- 1 → 5 → 4 → 3 (Camino con costo 35)

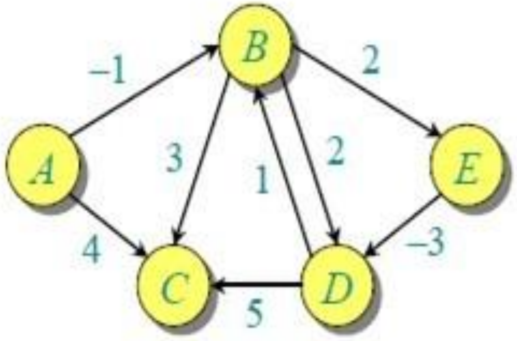


Caminos mínimos
Grafos con pesos positivos y negativos

Ejemplos:

- Simulacionescientíficas
- Redes de flujo
- Protocolosde ruteo basados en vector de distancias

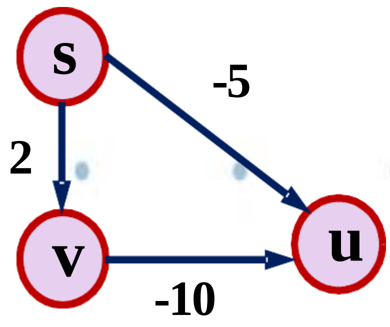




Caminos mínimos
Grafos con pesos positivos y negativos

➢ Estrategia: Encolar los vértices

Si el grafo tiene aristas negativas, el algoritmo de Dijkstra puede dar un resultado erróneo.



V	D_v	P_v	Conoc.
s	0	0	1
u	-5	s	1
v	2	s	1

Error !

La distancia mínima de **s** a **u** es -8

Caminos mínimos
Grafos con pesos positivos y negativos

Pasos:

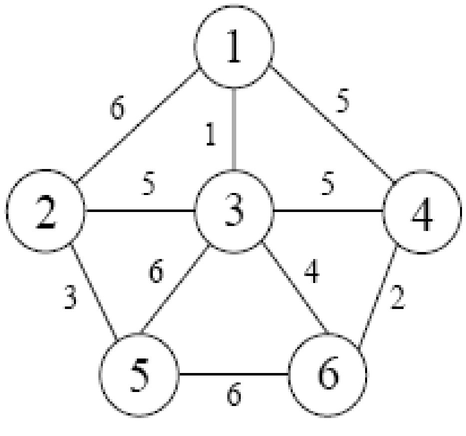
- Encolar el vértice origen ***s***.
- Procesar la cola:
  - Desencolar un vértice.
  - Actualizar la distancia de los adyacentes D_w siguiendo el mismo criterio de Dijkstra.
  - Si ***w*** no está en la cola, encolarlo.

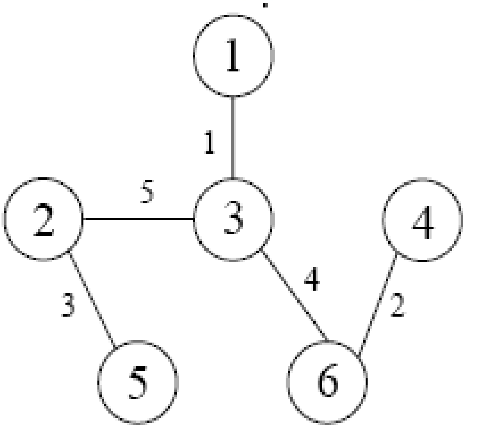
El costo total del algoritmo es O(|V| |E|)

Árbol de expansión mínima
Definición

Dado un grafo G=(V , E) no dirigido y conexo

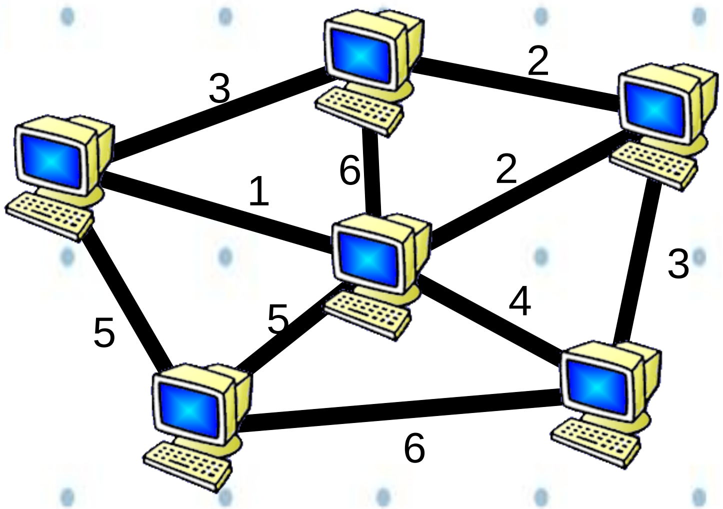
El árbol de expansión mínima es un árbol formado por las aristas de G que conectan todos los vértices con un costo total mínimo.

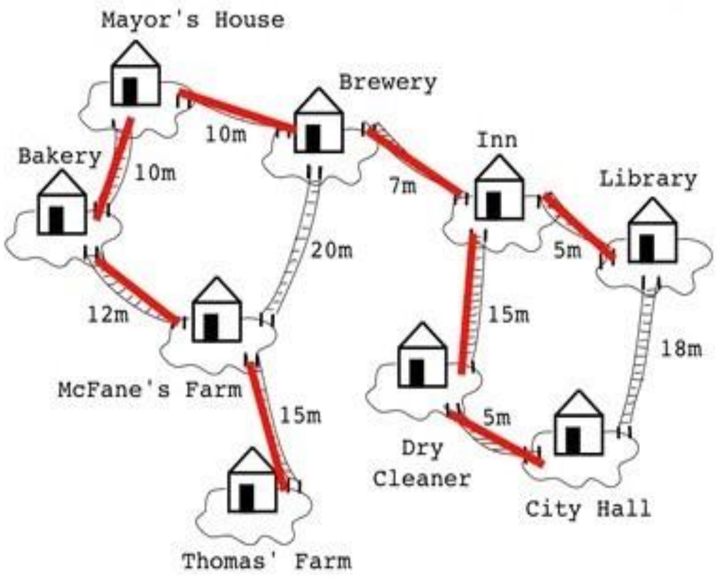




Árbol de expansión mínima

Ejemplo:





Conectar todas las computadoras con el menor costo total

Conectar todas las ciudades con el menor costo total

Árbol de expansión mínima
Algoritmo de Prim

- Construye el árbol haciéndolo crecer por etapas

Se elige un vértice como raíz del árbol.

En las siguientes etapas:

a) se selecciona la arista (u,v) de mínimo costo que cumpla: $u \in$ árbol y $v \notin$ árbol
b) se agrega al árbol la arista seleccionada en a) (es decir, ahora el vértice $v \in$ árbol)
c) se repite a) y b) hasta que se hayan tomado todos los vértices del grafo.

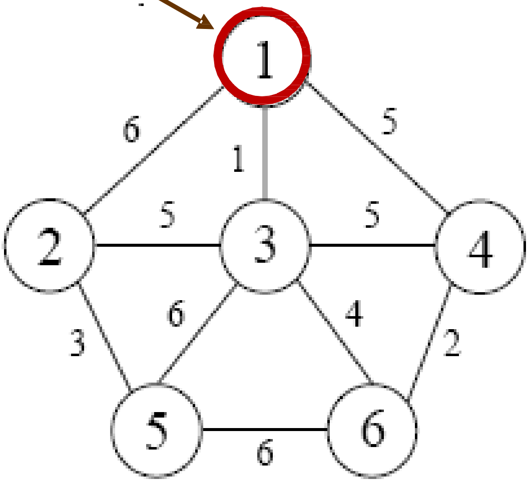
Árbol de expansión mínima
Algoritmo de Prim

- Construye el árbol haciéndolo crecer por etapas

Ejemplo:

1 Paso

Nodo raíz



costo de la arista (v,w)

Vértice inicial

Vértice elegido

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	∞	0	0
3	∞	0	0
4	∞	0	0
5	∞	0	0
6	∞	0	0

Árbol de expansión mínima
Algoritmo de Prim

1 Paso

Vértice elegido

Vértices actualizados

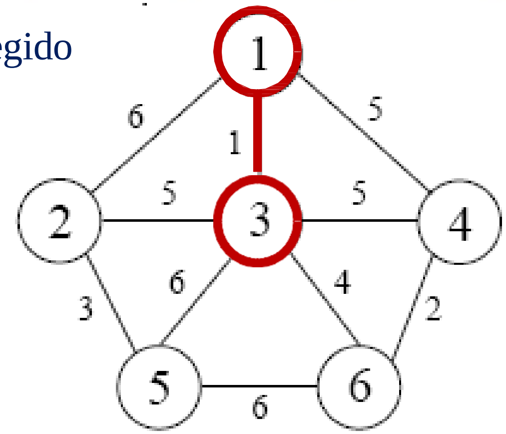
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	6	1	0
3	1	1	0
4	5	1	0
5	∞	0	0
6	∞	0	0

Árbol de expansión mínima
Algoritmo de Prim

V	***Costo***	***W***	***Conoc.***
1	0	0	1
2	6	1	0
3	1	1	1
4	5	1	0
5	∞	0	0
6	∞	0	0

Vértice elegido

2 Paso



Se agrega la arista (1,3) y el vértice 3

Árbol de expansión mínima
Algoritmo de Prim

2 Paso

Vértice elegido

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	6	1	0
3	1	1	1
4	5	1	0
5	∞	0	0
6	∞	0	0

Vértices actualizados

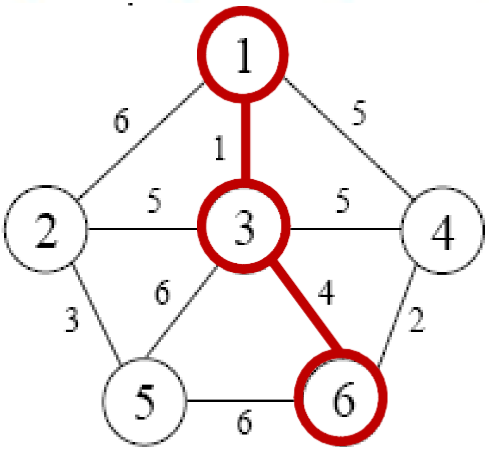
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	0
3	1	1	1
4	5	1	0
5	6	3	0
6	4	3	0

Árbol de expansión mínima
Algoritmo de Prim

V	*Costo*	*W*	*Conoc.*
1	∞	0	1
2	5	3	0
3	1	1	1
4	5	1	0
5	6	3	0
6	4	3	1

Vértice elegido

3 Paso



Se agrega la arista (3,6) y el vértice 6

Árbol de expansión mínima
Algoritmo de Prim

3 Paso

V	*Costo*	*W*	*Conoc.*
1	∞	0	1
2	5	3	0
3	1	1	1
4	5	1	0
5	6	3	0
6	4	3	1

Vértice elegido

Vértices actualizados

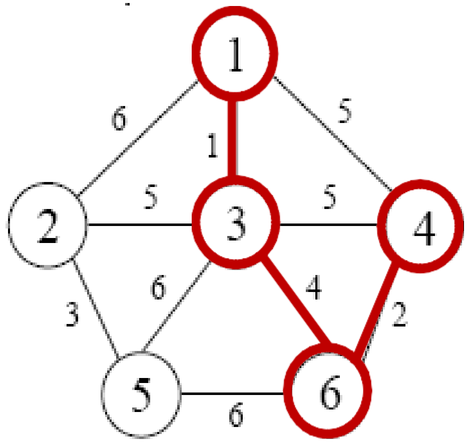
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	0
3	1	1	1
4	2	6	0
5	6	3	0
6	4	3	0

Árbol de expansión mínima
Algoritmo de Prim

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	0
3	1	1	1
4	2	6	1
5	6	3	0
6	4	3	1

Vértice elegido

4 Paso



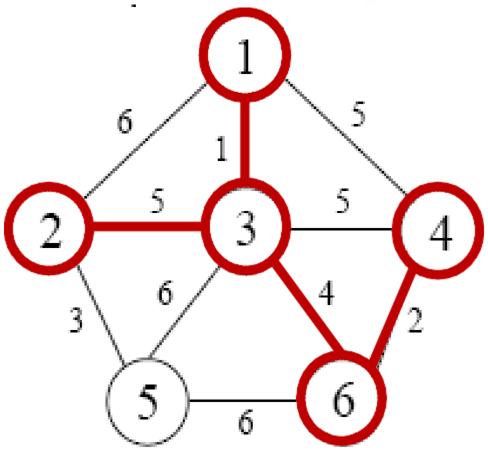
Se agrega la arista (6,4) y el vértice 4

Árbol de expansión mínima
Algoritmo de Prim

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	6	3	0
6	4	3	1

Vértice elegido

5 Paso



Se agrega la arista (3,2) y el vértice 2

Árbol de expansión mínima
Algoritmo de Prim

5 Paso

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	6	3	0
6	4	3	1

Vértice elegido

Vértice actualizado

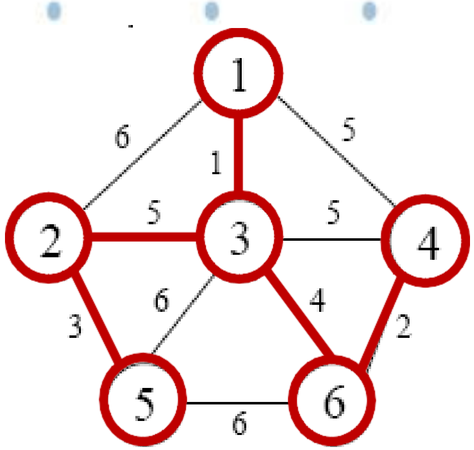
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	3	2	0
6	4	3	1

Árbol de expansión mínima
Algoritmo de Prim

V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	3	2	1
6	4	3	1

Vértice elegido

6° Paso



Se agrega la arista (2,5) y el vértice 5

Árbol de expansión mínima
Algoritmo de Prim

- Para la implementación se usa una tabla (similar a la utilizada en la implementación del algoritmo de Dijkstra).
- La dinámica del algoritmo consiste en, una vez seleccionado una *arista* (u,v) de costo mínimo tq $u \in$ árbol y $v \notin$ árbol:
  - se agrega la arista seleccionada al árbol
  - se actualizan los costos a los adyacentes del vértice v de la sig. manera :
    - ✓ se compara Costo_w con $c(v,w)$

Costo mínimo a w (costo de la arista entre un vértice perteneciente al árbol y vértice w)

Costo de la arista (v,w)

  - se actualiza si $\text{Costo}_w > c(v,w)$

Árbol de expansión mínima
Algoritmo de Prim

- Se hacen las mismas consideraciones que para el algoritmo de Dijkstra
  - Si se implementa con una tabla secuencial:
    - → El costo total del algoritmo es $O(|V|^2)$
  - Si se implementa con heap:
    - → El costo total del algoritmo es $O(|E| \log|V|)$

Árbol de expansión mínima
Algoritmo de Kruskal

- Selecciona las aristas en orden creciente según su peso y las acepta si no originan un ciclo.
- El invariante que usa me indica que en cada punto del proceso, dos vértices pertenecen al mismo conjunto si y sólo sí están conectados.
- Si dos vértices ***u*** y ***v*** están en el mismo conjunto, la arista (***u***,***v***) es rechazada porque al aceptarla forma un ciclo.

Árbol de expansión mínima
Algoritmo de Kruskal

- Inicialmente cada vértice pertenece a su propio conjunto

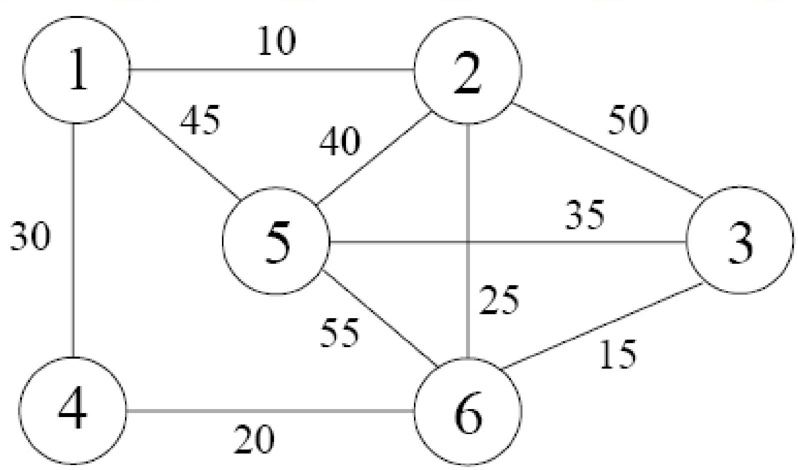
  → |V| conjuntos con un único elemento

- Al aceptar una arista se realiza la Unión de dos conjuntos

- Las aristas se organizan en una heap, para ir recuperando la de mínimo costo en cada paso

Árbol de expansión mínima
Algoritmo de Kruskal

Ejemplo:



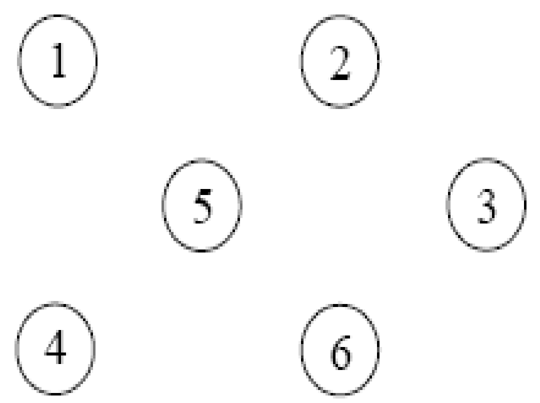
Aristas ordenadas por su costo de menor a mayor:

(1,2) → 10
(3,6) → 15
(4,6) → 20
(2,6) → 25
(1,4) → 30
(5,3) → 35
(5,2) → 40
(1,5) → 45
(2,3) → 50
(5,6) → 55

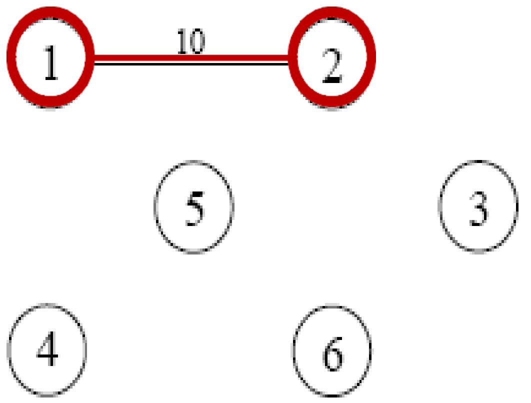
- Ordenar las aristas, usando un algoritmo de ordenación
- Construir una min-heap → **más eficiente**

Árbol de expansión mínima
Algoritmo de Kruskal

Inicialmente cada vértice está en su propio conjunto

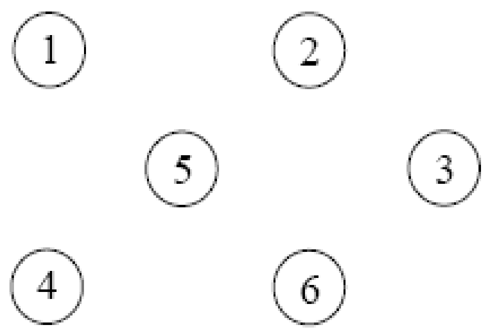


Se agrega la arista (1,2)

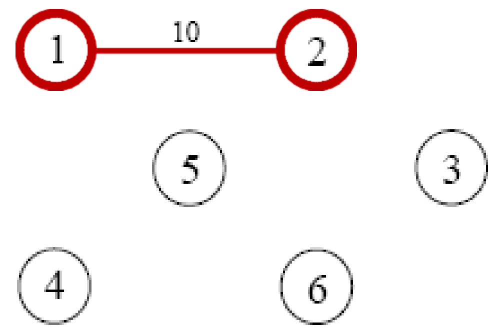


Árbol de expansión mínima
Algoritmo de Kruskal

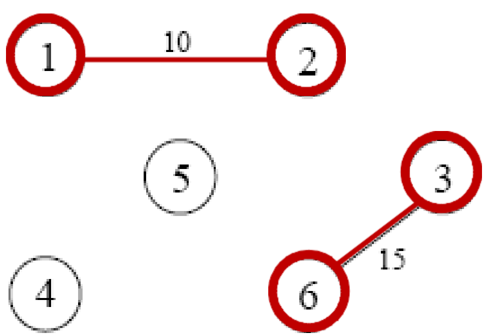
Inicialmente cada vértice está en su propio conjunto

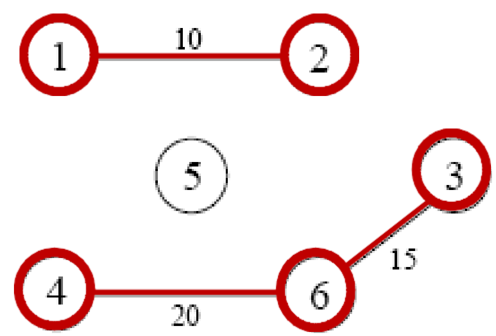


Se agrega la arista (1,2)



Se agrega la arista (3,6)

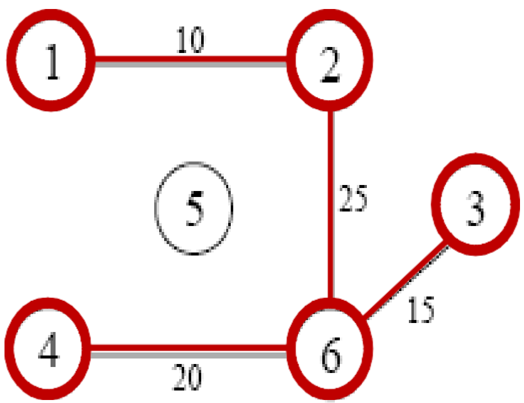




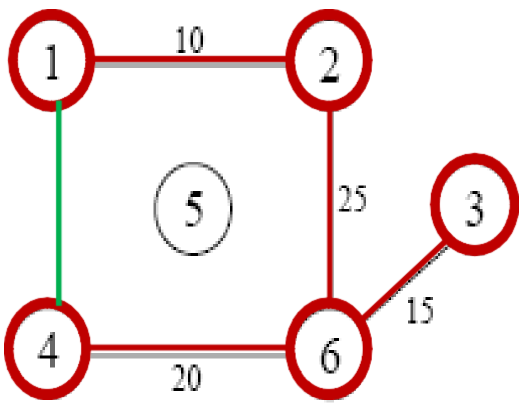
Se agrega la arista (4,6)

Árbol de expansión mínima
Algoritmo de Kruskal

Se agrega la arista (2,6)



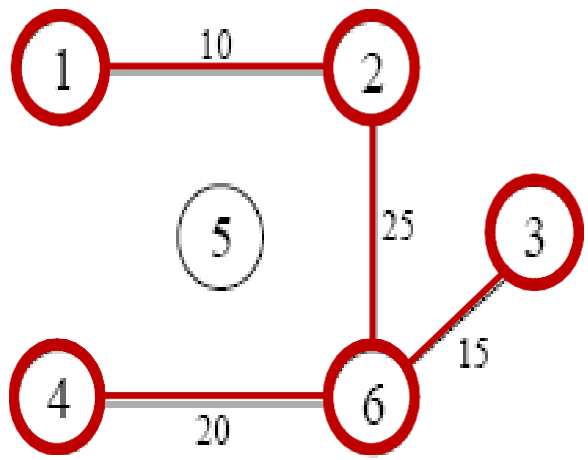
¿Se agrega la arista (1,4) con costo 30?



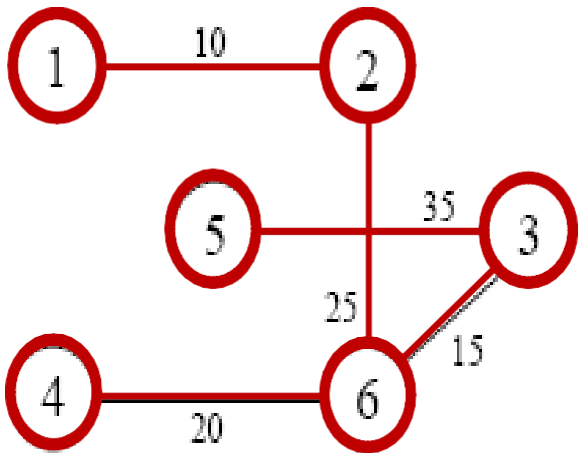
No, porque forma ciclo, ya que pertenece a la misma componente conexa

Árbol de expansión mínima
Algoritmo de Kruskal

Se agrega la arista(2,6)



Se agrega la arista(3,5)



Árbol de expansión mínima
Algoritmo de Kruskal

- Se organizan las aristas en una heap, para optimizar la recuperación de la arista de mínimo costo

- El tamaño de la heap es $|E|$, y extraer cada arista lleva $O(\log |E|)$

- El tiempo de ejecución es $O(|E| \log|E|)$

- Dado que $|E| \leq |V|^2$, $\log |E| \leq 2 \log |V|$,

  → el costo total del algoritmo es $O(|E| \log|V|)$