Agenda - Grafos

- Árbol de expansión mínimo

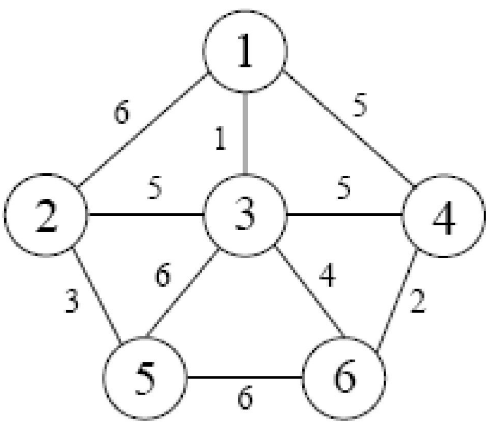
Agenda – Grafos

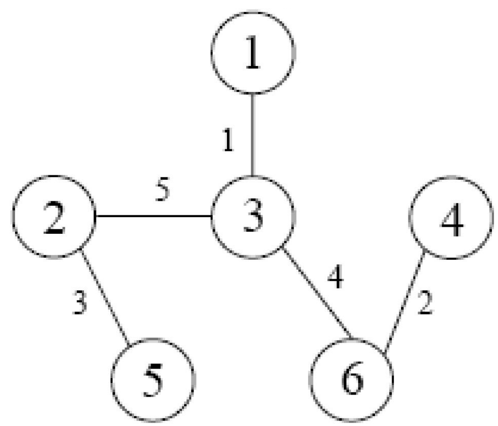
- Árbol de expansión mínimo
  - Definición
  - Aplicaciones
  - Algoritmo de Prim
  - Algoritmo de Kruskal

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.



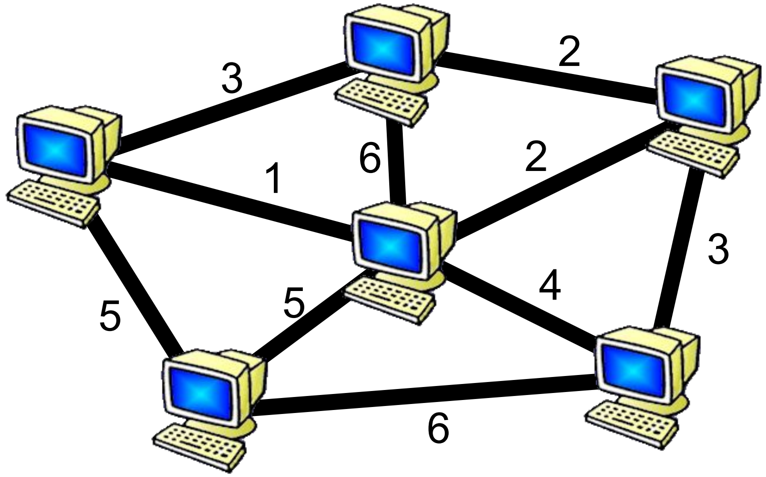


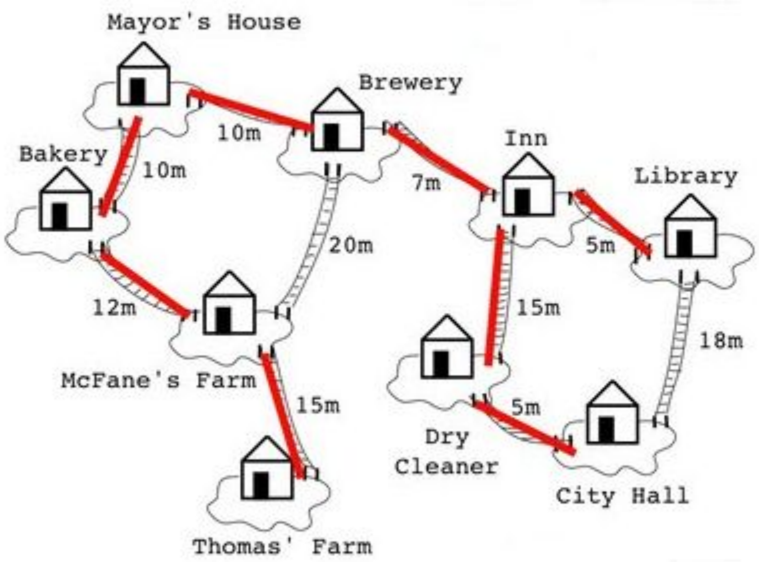
Árbol de expansión mínima
Aplicaciones

- Construcción de tendidos eléctricos
- Diseño de redes de tuberías
- Cableado de redes de comunicaciones
- Diseño de redes de logística y transporte
- Taxonomías
- …….

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.

Algoritmo de Prim
Implementación

- 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*)

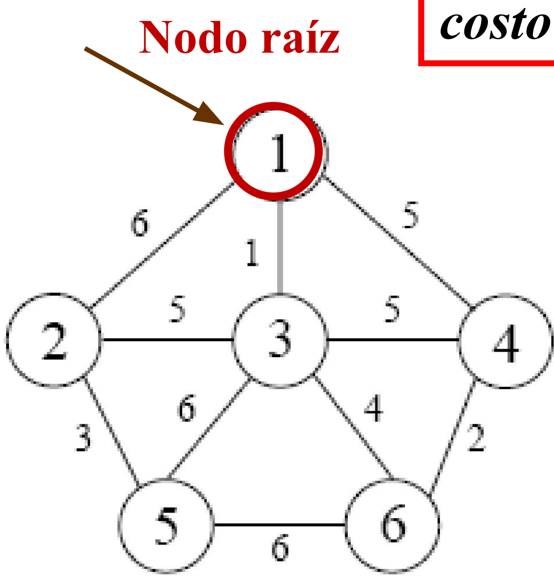
Costo de la arista (*v* ,*w*)

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

Algoritmo de Prim
Implementación

- Construye el árbol haciéndolo crecer por etapas

Ejemplo: **1° Paso**



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

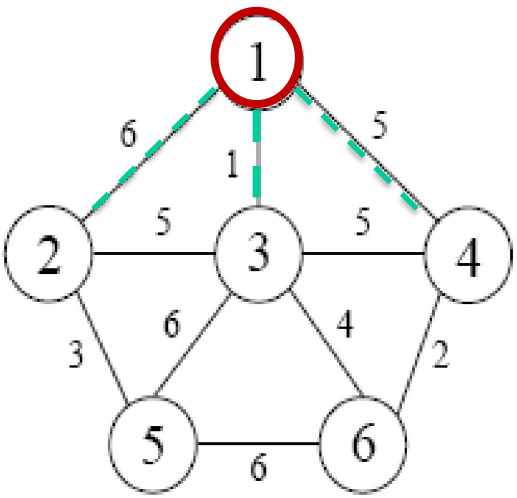
Algoritmo de Prim
Implementación

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

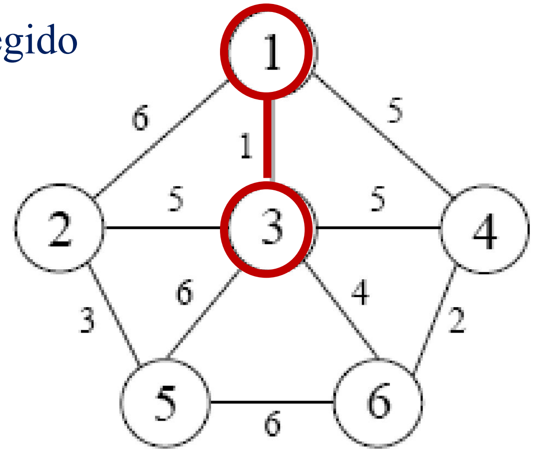


Algoritmo de Prim
Implementación

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

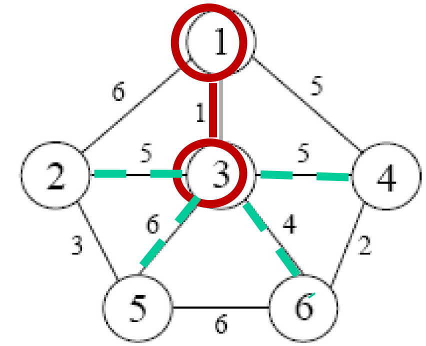
Vértice elegido

2° Paso



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

Algoritmo de Prim
Implementación



2° Paso

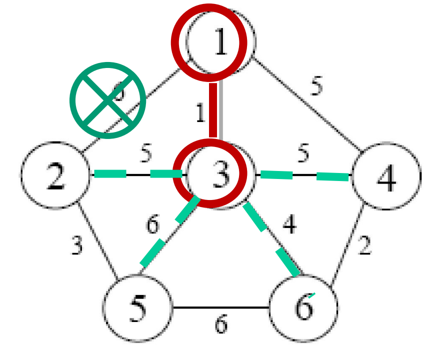
Vértice elegido

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

Algoritmo de Prim
Implementación



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

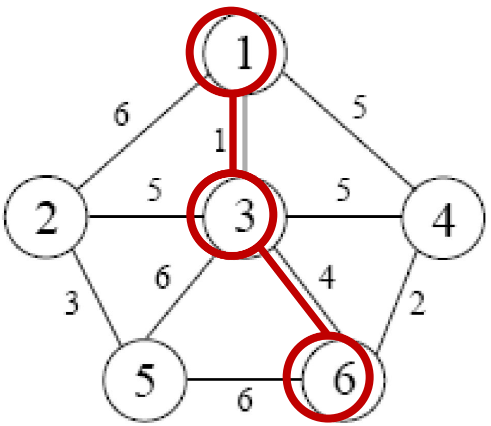
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	0
3	1	1	1
4	5	1	0
5	6	3	0
6	4	3	0

Algoritmo de Prim
Implementación

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

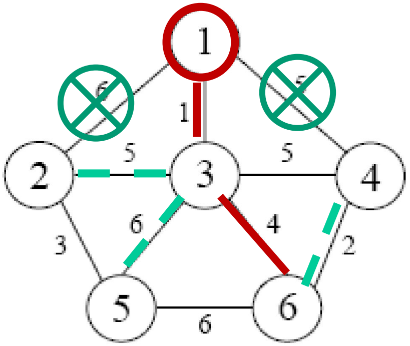
3° Paso



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

Algoritmo de Prim

Implementación



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

Vértices actualizados

Vértice elegido

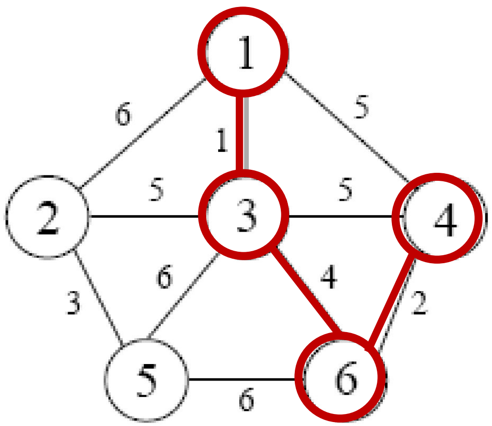
V	Costo	W	Conoc.
1	0	0	1
2	5	3	0
3	1	1	1
4	2	6	0
5	6	3	0
6	4	3	0

Algoritmo de Prim
Implementación

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



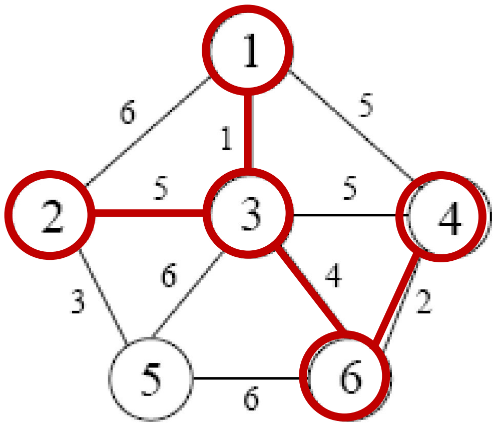
Se agrega la arista (6,4) y el vértice 4

Algoritmo de Prim
Implementación

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

Algoritmo de Prim
Implementación

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

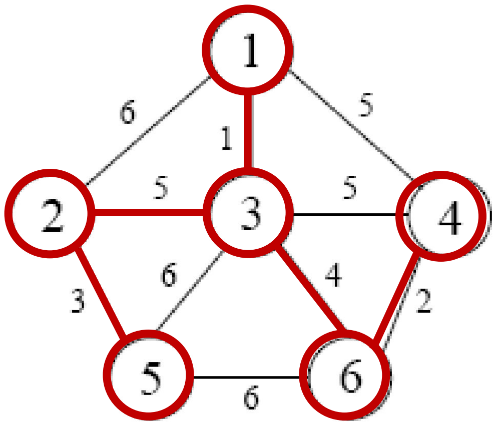
V	*Costo*	*W*	*Conoc.*
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	3	2	0
6	4	3	1

Algoritmo de Prim
Implementación

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

Algoritmo de Prim
Tiempo de Ejecución

- 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

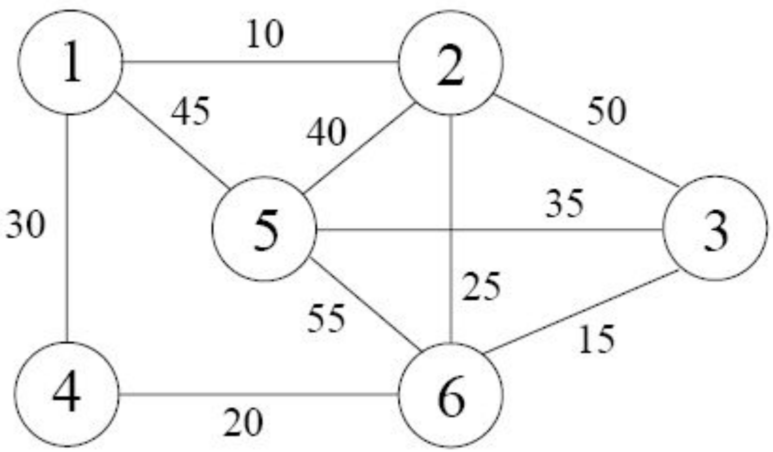
- Selecciona las aristas en orden creciente según su peso y las acepta si no originan un ciclo.
- El invariante que usa me indica que en cada punto del proceso, dos vértices pertenecen al mismo conjunto si y sólo sí están conectados.
- Si dos vértices ***u*** y ***v*** están en el mismo conjunto, la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

Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

- 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.

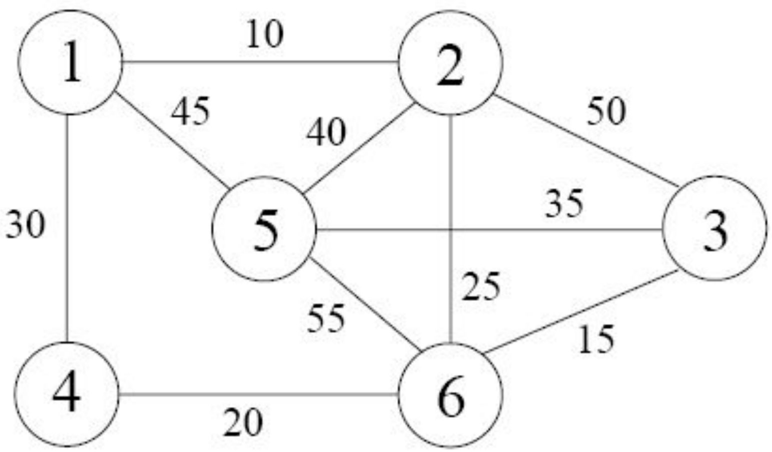
Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

Ejemplo:



Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

Ejemplo:



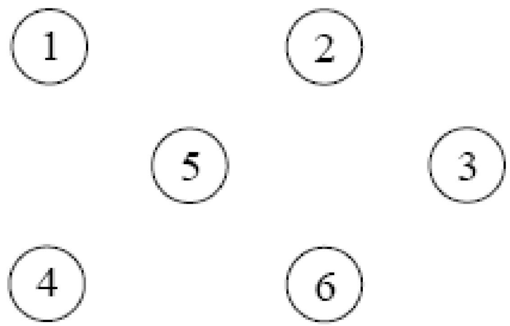
Aristas ordenadas por su costo de menor a mayor:

(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

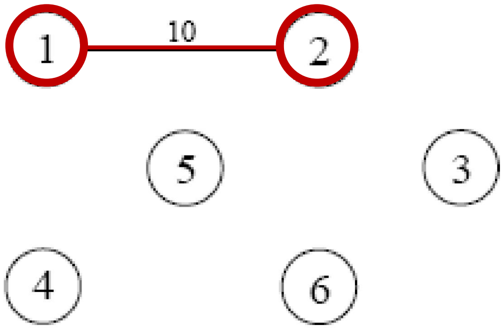
- Ordenar las aristas, usando un algoritmo de ordenación
- Construir una min-heap □ **más eficiente**

Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

Inicialmente cada vértice está en su propio conjunto

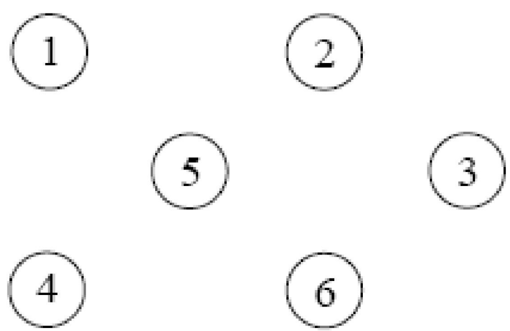


Se agrega la arista (1,2)

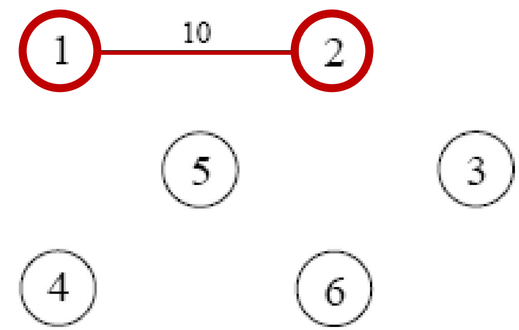


Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

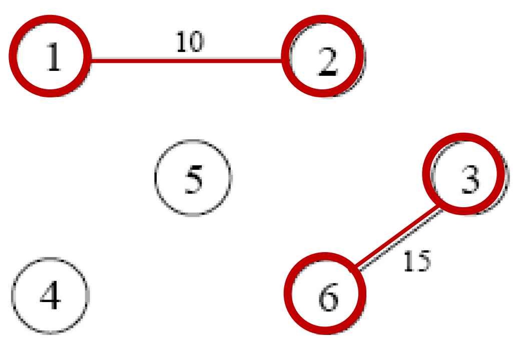
Inicialmente cada vértice está en su propio conjunto

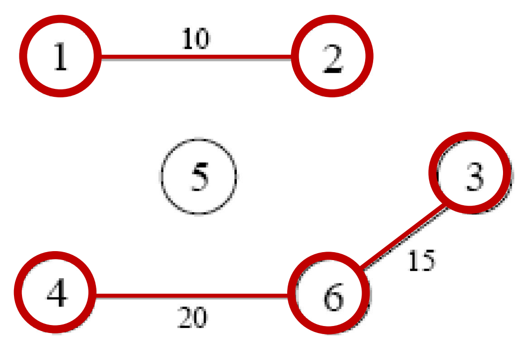


Se agrega la arista (1,2)



Se agrega la arista (3,6)

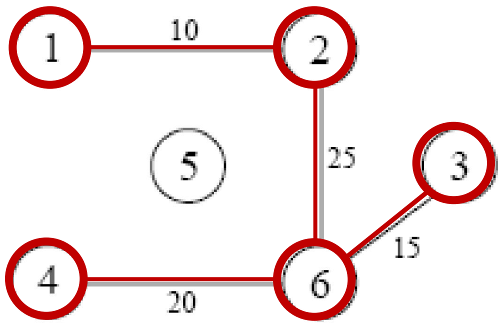




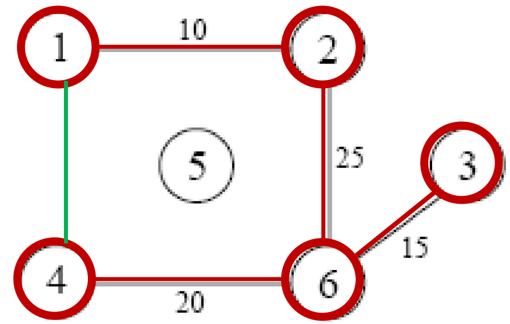
Se agrega la arista (4,6)

Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

Se agrega la arista (2,6)



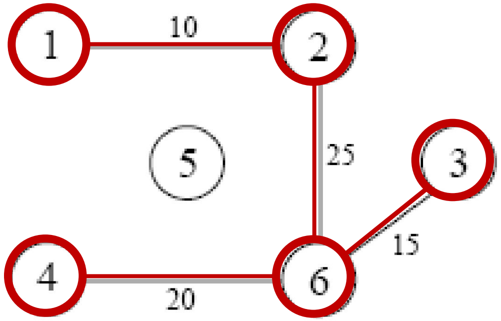
¿Se agrega la arista (1,4) con costo 30?



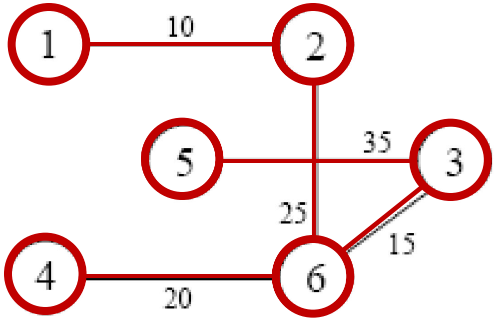
No, porque forma ciclo, ya que pertenece a la misma componente conexa

Árbol de expansión mínima
Algoritmo de Kruskal (cont.)

Se agrega la arista (2,6)



Se agrega la arista (3,5)



Algoritmo de Kruskal
Tiempo de Ejecución

- 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 $O(|E| \log|V|)$

Grafos
Conclusiones

- Podemos utilizar grafos para modelar problemas de la "vida real".
- Los grafos son una herramienta fundamental en resolución de problemas.

- Representación:
    - Tamaño reducido: matrices de adyacencia.
    - Tamaño grande y grafo "disperso": listas de adyacencia.

Grafos
Conclusiones

- Existen muchos algoritmos “clásicos” para resolver diferentes problemas sobre grafos.
- **Nuestro trabajo:** saber modelar los problemas de interés usando grafos y encontrar el algoritmo adecuado para la aplicación que se requiera.
- Es importante el estudio de problemas genéricos sobre grafos.
- La búsqueda primero en profundidad (DFS) y búsqueda en amplitud (BFS) son herramientas básicas, subyacentes en muchos de los algoritmos estudiados

Problema de interés ➡ Problema con grafos ➡ Algoritmo genérico con grafos ➡ Algoritmo para el problema de interés