

Programación 3

Cursada 2016

Prof. Alejandra Schiavoni

Ingeniería en Computación - UNLP



GRAFOS



Agenda - Grafos

- Caminos de costo mínimo
- Árbol de expansión mínimo



Agenda - Grafos

- Caminos de costo mínimo
- Árbol de expansión mínimo



Agenda – Grafos

- ❖ Caminos de costo mínimo
 - ❖ Definición
 - ❖ Algoritmos para el cálculo del camino mínimo desde un origen en:
 - Grafos sin peso
 - Grafos con pesos positivos
 - Algoritmo de Dijkstra: dos implementaciones
 - Grafos con pesos positivos y negativos
 - Grafos acíclicos
 - ❖ Algoritmo para el cálculo de los caminos mínimos entre todos los pares de vértices



Camino de costo mínimo

Definición

Sea $G=(V,A)$ un grafo dirigido y pesado, el costo $c(i,j)$ está asociado a la arista $v(i,j)$.

Dado un camino: $v_1, v_2, v_3, \dots \dots v_N$

El costo del camino es:

$$C = \sum_{i=1}^{N-1} c(i, i+1)$$

Este valor también se llama longitud del camino pesado.

La longitud del camino no pesado es la cantidad de aristas



Camino de costo mínimo

Definición (cont.)

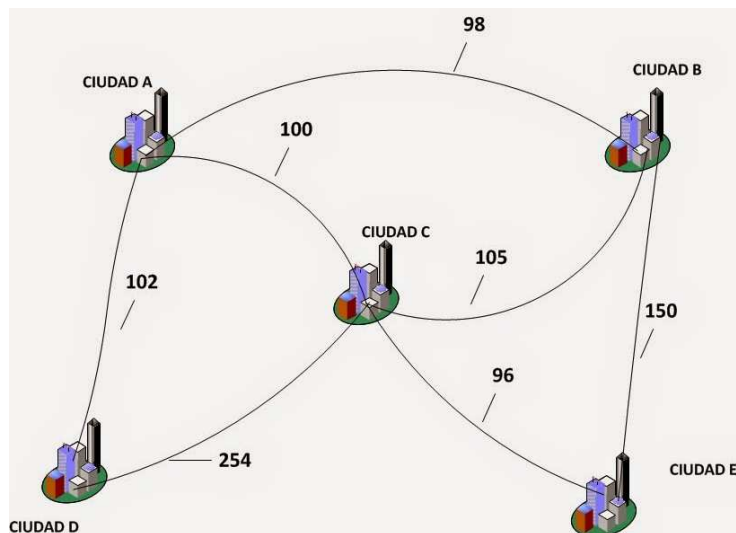
El camino de costo mínimo desde un vértice v_i a otro vértice v_j es aquel en que la suma de los costos de las aristas es mínima.

Esto significa que:

$$C = \sum_{i=1}^{N-1} c(i, i+1) \quad \text{es mínima}$$

Camino de costo mínimo

Ejemplos:



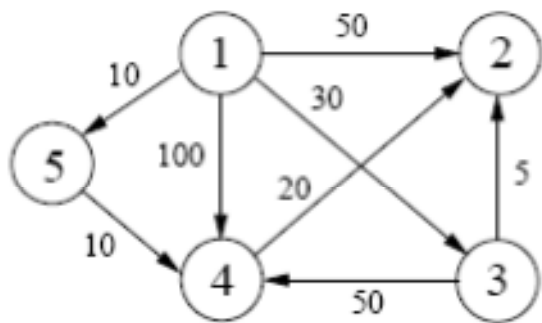
Ciudades conectadas por
Rutas con **distancias**



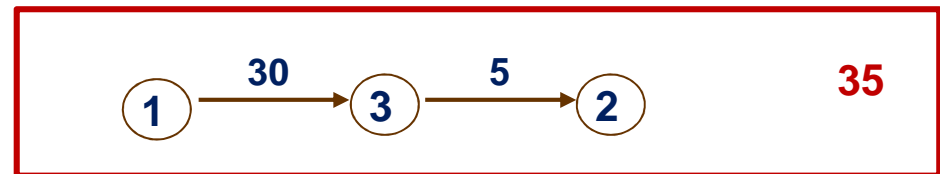
Personas conectadas a
través de las redes sociales

Camino de costo mínimo

Ejemplo:



Camino posible desde el
vértice 1 al vértice 2





Algoritmos de Caminos mínimos

- Grafos sin peso
- Grafos con pesos positivos
- Grafos con pesos positivos y negativos
- Grafos dirigidos acíclicos



Algoritmos de Caminos mínimos

Los algoritmos calculan los caminos mínimos desde un vértice origen s a **todos** los restantes vértices del grafo

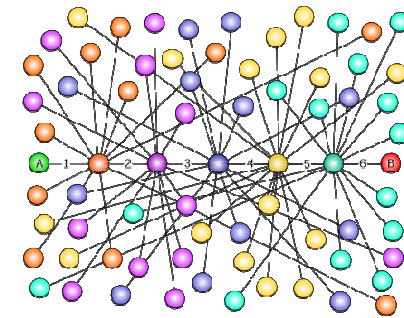
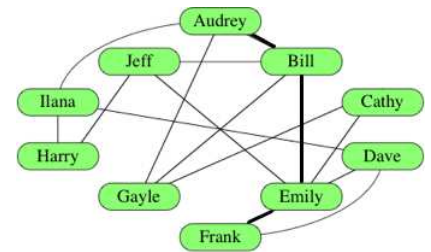
Algoritmos de Caminos mínimos

Grafos sin peso

Ejemplos

➤ Seis grados de separación

Se le llama *seis grados de separación* a la hipótesis que intenta probar que cualquiera en la Tierra puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios (conectando a ambas personas con sólo seis enlaces)



➤ Número de Erdős

Es un modo de describir la distancia colaborativa, en lo relativo a trabajos matemáticos entre un autor y Paul Erdős (matemático húngaro considerado uno de los escritores más prolíficos de trabajos matemáticos)



Si la **mujer de rojo** colabora directamente con Erdős en un trabajo, y luego el **hombre de azul** colabora con ella; entonces el hombre de azul tiene un número de Erdős con valor 2, y está "a dos pasos" de Paul Erdős (asumiendo que nunca ha colaborado directamente con éste).

➤ El número de Bacon es una aplicación de la misma idea en la industria fílmica- un cálculo que conecta actores que han aparecido junto al actor *Kevin Bacon* en alguna película.



Algoritmos de Caminos mínimos

Grafos sin peso

➤ Estrategia: Recorrido en amplitud (BFS)

Para cada vértice v mantiene la siguiente información:

- D_v : distancia mínima desde el origen (inicialmente ∞ para todos los vértices excepto el origen con valor 0)
- P_v : vértice por donde paso para llegar
- Conocido: dato booleano que me indica si está procesado (inicialmente todos en 0)



Algoritmos de Caminos mínimos

Grafos sin peso

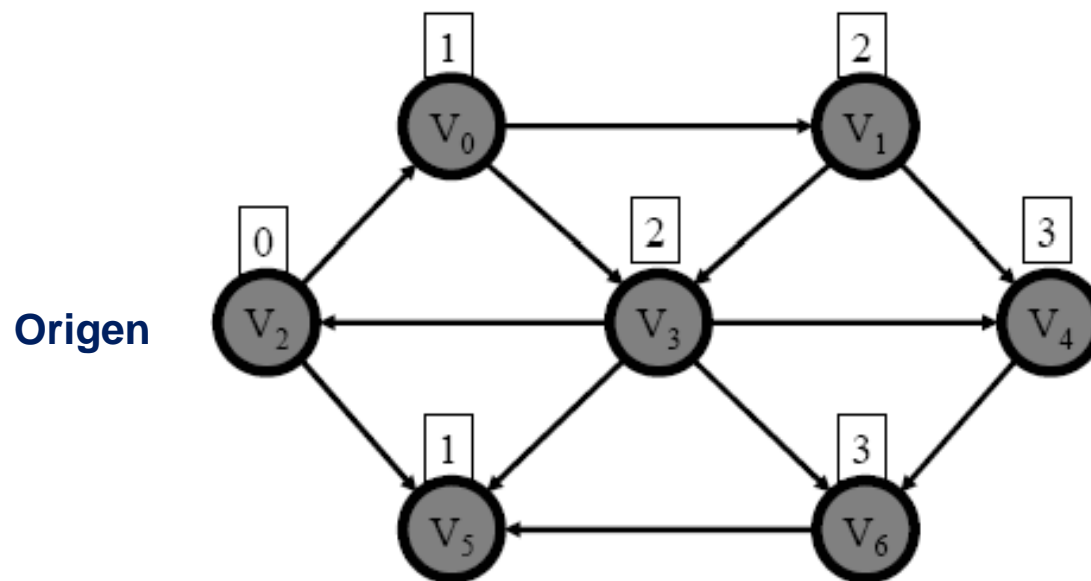
➤ Estrategia: Recorrido en amplitud (BFS)

Pasos:

- Avanzar por niveles a partir del origen, asignando distancias según se avanza (se utiliza una cola)
- Inicialmente, es $D_w = \infty$. Al inspeccionar w se reduce al valor correcto $D_w = D_v + 1$
- Desde cada v , visitamos a todos los nodos adyacentes a v

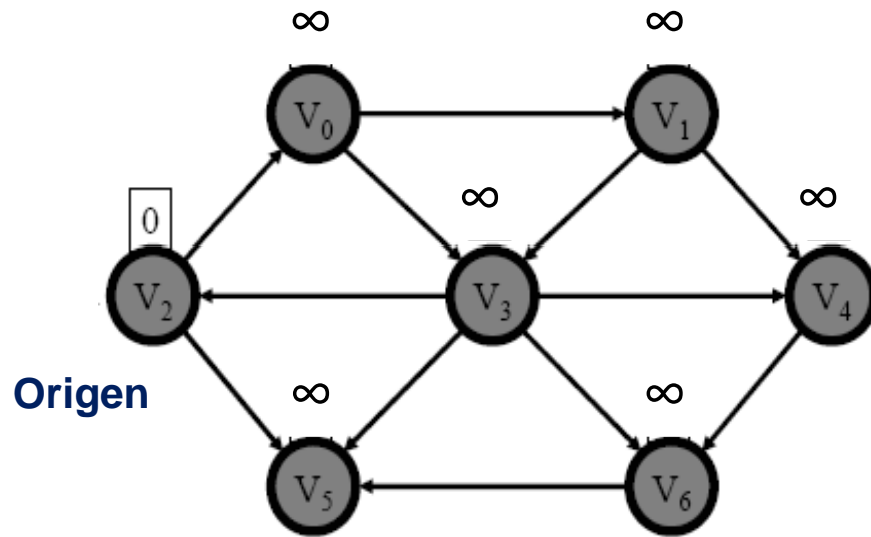
Algoritmos de Caminos mínimos

Grafos sin peso (cont.)



Algoritmos de Caminos mínimos

Grafos sin peso (cont.)



Valores iniciales de la tabla

V_i	D_v	P_v	Conoc
V_0	∞	0	0
V_1	∞	0	0
V_2	0	0	0
V_3	∞	0	0
V_4	∞	0	0
V_5	∞	0	0
V_6	∞	0	0

Algoritmo Caminos mínimos basado en *BFS*

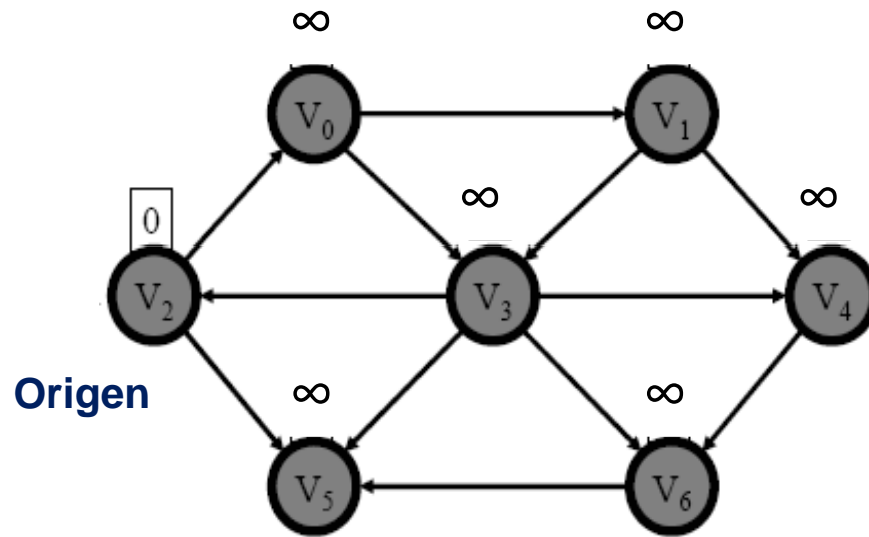
```
Camino_min_GrafoNoPesadoG,s) {  
(1)   para cada vértice  $v \in V$   
(2)        $D_v = \infty$ ;  $P_v = 0$ ;  $Conoc_v = 0$ ;  
(3)    $D_s = 0$ ; Encolar (Q,s);  $Conoc_v = 1$ ;  
(4)   Mientras (not esVacio(Q)) {  
(5)       Desencolar (Q,u);  
(6)       Marcar u como conocido;  
(7)       para c/vértice  $w \in V$  adyacente a u {  
(8)           si ( $w$  no es conocido) {  
(9)                $D_w = D_u + 1$ ;  
(10)               $P_w = u$ ;  
(11)              Encolar(Q,w);  $Conoc_w = 1$ ;  
(12)          }  
(13)      }  
(14)  }
```

Algoritmo Caminos mínimos basado en *BFS*

```
Camino_min_GrafoNoPesadoG,s) {  
(1)   para cada vértice  $v \in V$   
(2)        $D_v = \infty$ ;  $P_v = 0$ ; Conocv = 0;  
(3)    $D_s = 0$ ; Encolar (Q,s); Conocs = 1;  
(4)   Mientras (not esVacio(Q)) {  
(5)       Desencolar (Q,u);  
(6)       Marcar u como conocido;  
(7)       para c/vértice  $w \in V$  adyacente a u {  
(8)           si (w no es conocido) {  
(9)                $D_w = D_u + 1$ ;  
(10)               $P_w = u$ ;  
(11)              Encolar(Q,w); Conocw = 1;  
(12)          }  
(13)      }  
(14)  }
```

Algoritmos de Caminos mínimos

Grafos sin peso (cont.)

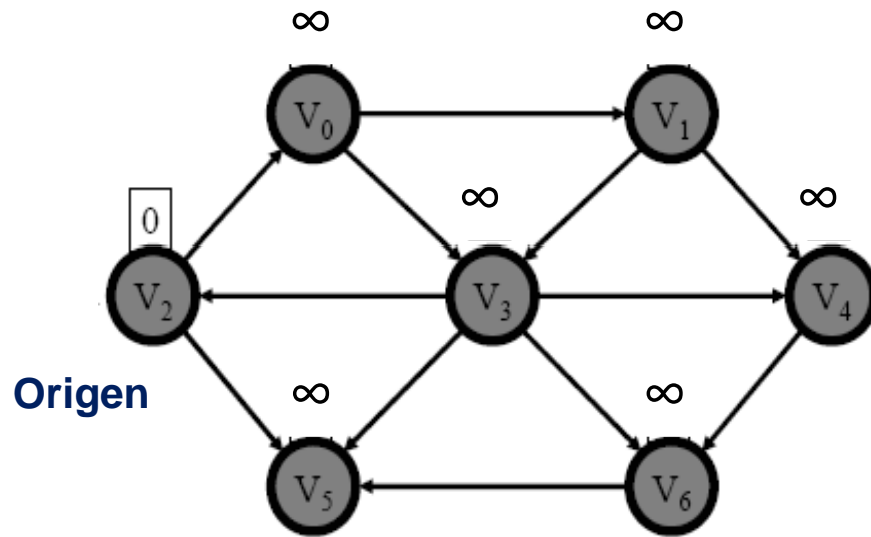


Valores iniciales de la tabla

V_i	D_v	P_v	Conoc
V_0	∞	0	0
V_1	∞	0	0
V_2	0	0	0
V_3	∞	0	0
V_4	∞	0	0
V_5	∞	0	0
V_6	∞	0	0

Algoritmos de Caminos mínimos

Grafos sin peso (cont.)



Valores iniciales de la tabla

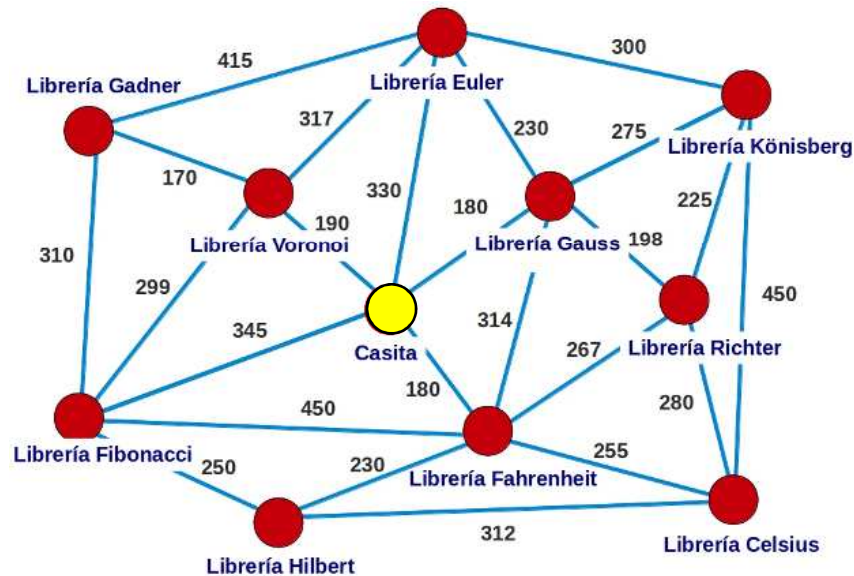
V_i	D_v	P_v
V_0	∞	0
V_1	∞	0
V_2	0	0
V_3	∞	0
V_4	∞	0
V_5	∞	0
V_6	∞	0

Algoritmo Caminos mínimos basado en *BFS*

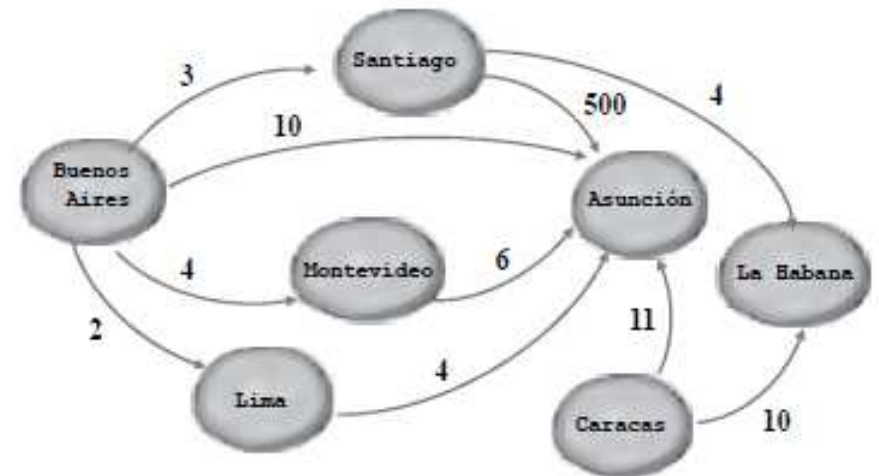
```
Camino_min_GrafoNoPesadoG,s) {  
(1)   para cada vértice  $v \in V$   
(2)        $D_v = \infty$ ;  $P_v = 0$ ;  
(3)    $D_s = 0$ ; Encolar ( $Q,s$ );  
(4)   Mientras (not esVacio( $Q$ )) {  
(5)       Desencolar( $Q,u$ );  
(6)       para c/vértice  $w \in V$  adyacente a  $u$  {  
(7)           si ( $D_w = \infty$ ) {  
(8)                $D_w = D_u + 1$ ;  
(9)                $P_w = u$ ;  
(10)            Encolar( $Q,w$ );  
(11)        }  
(12)    }  
(13) }  
}
```

Algoritmos de Caminos mínimos

Grafos con pesos positivos – Algoritmo de Dijkstra



Encontrar los caminos más cortos desde Casita a cada una de las librerías



Encontrar la ruta aérea más corta desde Buenos Aires a Asunción



Algoritmos de Caminos mínimos

Grafos con pesos positivos – Algoritmo de Dijkstra

➤ Estrategia: Algoritmo de Dijkstra

Pasos:

- Dado un vértice origen s , elegir el vértice v que esté a la menor distancia de s , dentro de los vértices no procesados
- Marcar v como procesado
- Actualizar la distancia de w adyacente a v



Algoritmo de Dijkstra (cont.)

- Para cada vértice v mantiene la siguiente información:
 - D_v : distancia mínima desde el origen (inicialmente ∞ para todos los vértices excepto el origen con valor 0)
 - P_v : vértice por donde paso para llegar
 - Conocido : dato booleano que me indica si está procesado (inicialmente todos en 0)

Algoritmo de Dijkstra (cont.)

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

- Se compara D_w con $D_v + c(v,w)$

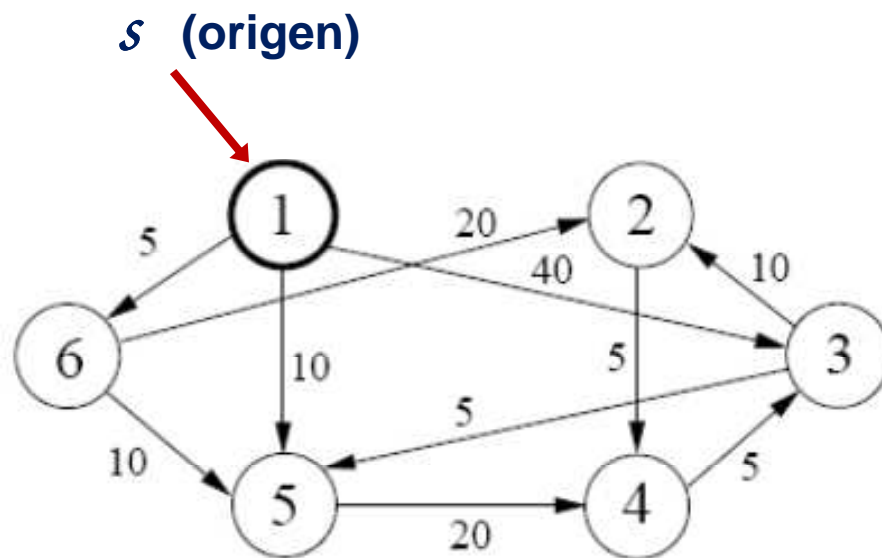
Distancia de s a w
(sin pasar por v)

Distancia de s a w ,
pasando por v

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

Algoritmo de Dijkstra

Ejemplo

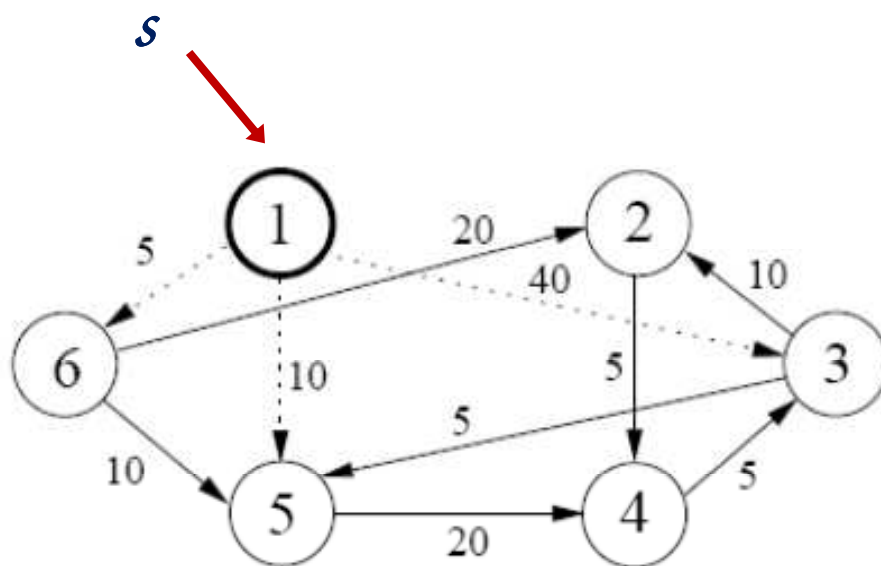


Valores iniciales de la tabla

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

Algoritmo de Dijkstra

Ejemplo (cont.)

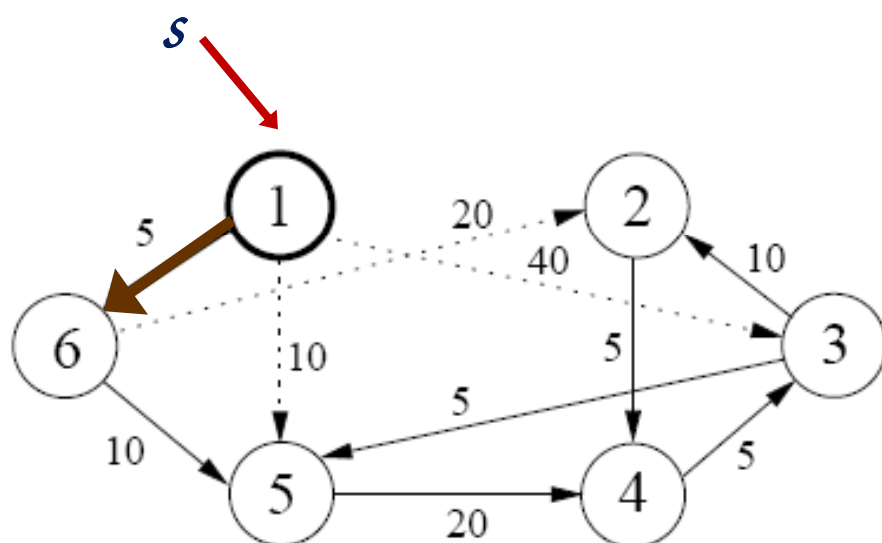


- Valores al seleccionar el vértice 1
- Actualiza la distancia de 3, 5 y 6

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

Algoritmo de Dijkstra

Ejemplo (cont.)



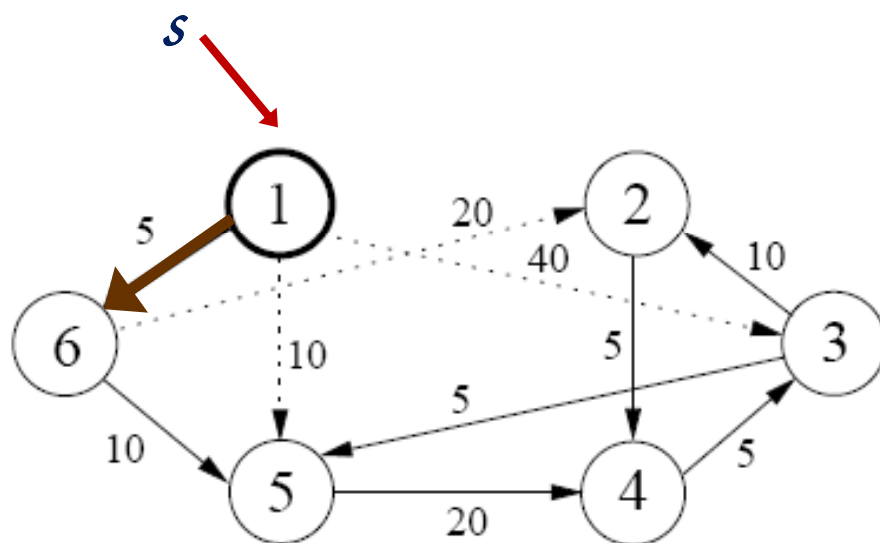
Próximo vértice a elegir



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

Algoritmo de Dijkstra

Ejemplo (cont.)

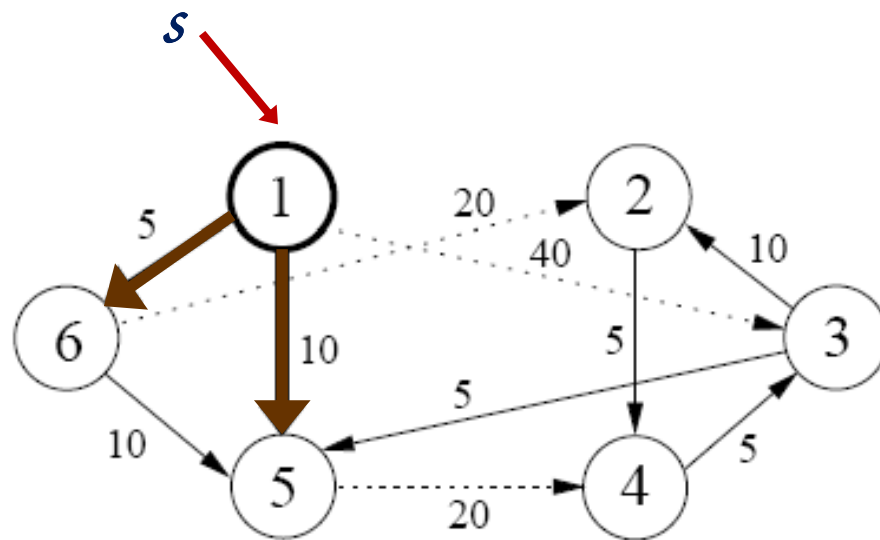


V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	1

- Valores al seleccionar el vértice 6
- Actualiza la distancia de 2
- La distancia de 5 es mayor que la de la tabla (no se actualiza)

Algoritmo de Dijkstra

Ejemplo (cont.)

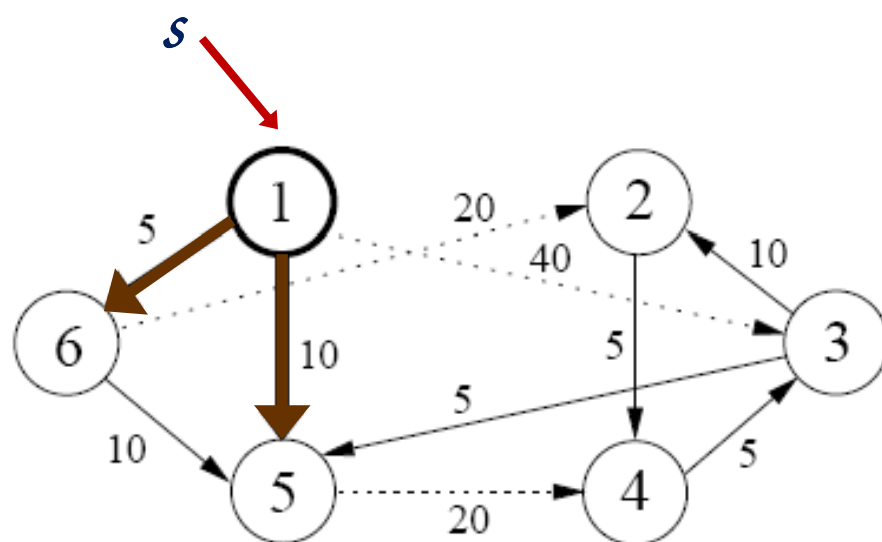


Próximo vértice a elegir

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

Algoritmo de Dijkstra

Ejemplo (cont.)

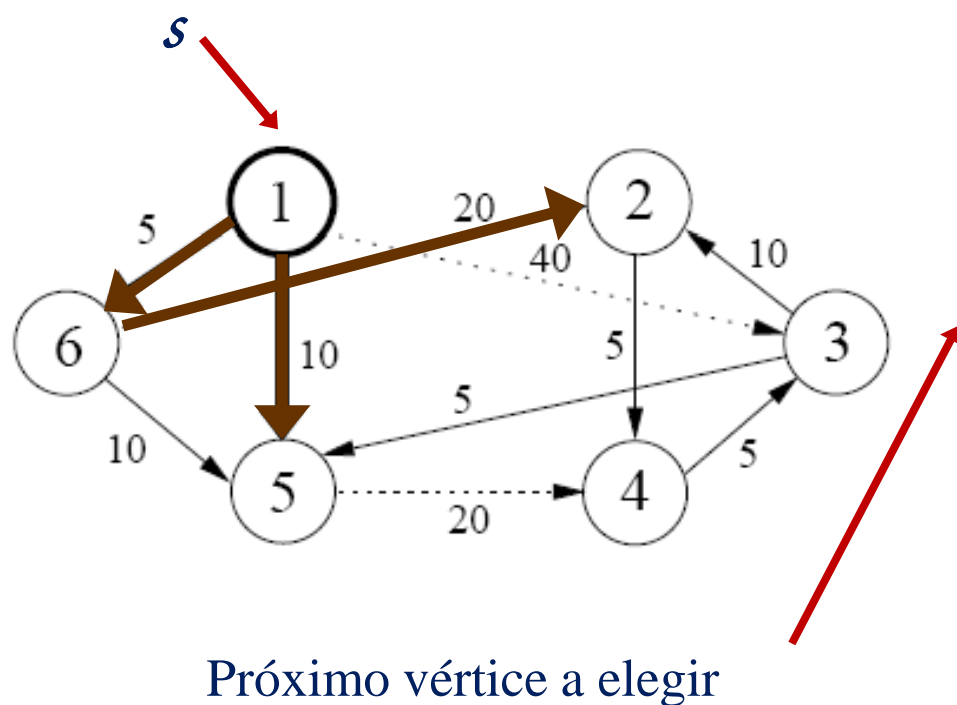


- Valores al seleccionar el vértice 5
- Actualiza la distancia de 4

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

Algoritmo de Dijkstra

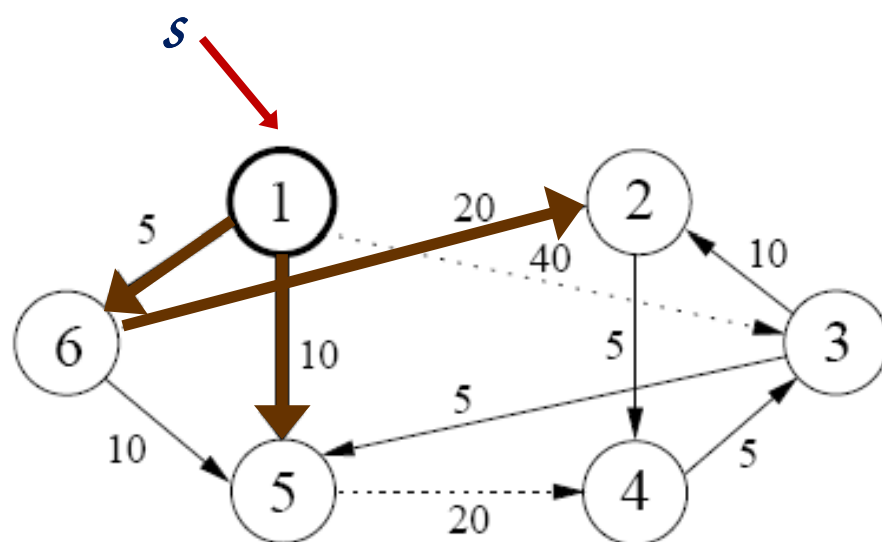
Ejemplo (cont.)



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

Algoritmo de Dijkstra

Ejemplo (cont.)



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

- Valores al seleccionar el vértice 2
- La distancia de 4 es igual que la de la tabla (no se actualiza)

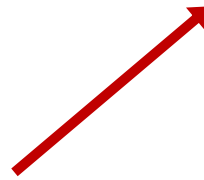


Algoritmo de Dijkstra

Ejemplo (cont.)

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

El resultado final es:

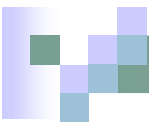


V	D_v	P_v	Conoc.
1	0	0	1
2	25	6	1
3	35	4	1
4	30	5	1
5	10	1	1
6	5	1	1



Algoritmo de Dijkstra

```
Dijkstra( $G, w, s$ ) {  
(1)   para cada vértice  $v \in V$   
(2)        $D_v = \infty$ ;    $P_v = 0$ ;  
(3)    $D_s = 0$ ;  
(4)   para cada vértice  $v \in V$  {  
(5)        $u = \text{vérticeDesconocidoMenorDist}$ ;  
(6)       Marcar  $u$  como conocido;  
(7)       para cada vértice  $w \in V$  adyacente a  $u$   
(8)           si ( $w$  no está conocido)  
(9)               si ( $D_w > D_u + c(u, w)$ ) {  
(10)                    $D_w = D_u + c(u, w)$ ;  
(11)                    $P_w = u$ ;  
(12)               }  
(13)       }  
(14) }  
}
```



Algoritmo de Dijkstra

Tiempo de ejecución (I)

- El bucle *para* de la línea (4) se ejecuta para todos los vértices
→ $|V|$ iteraciones
- La operación *vérticeDesconocidoMenorDist* es $O(|V|)$ y dado que se realiza $|V|$ veces
→ el costo total de *vérticeDesconocidoMenorDist* es $O(|V|^2)$
- El bucle *para* de la línea (7) se ejecuta para los vértices adyacentes de cada vértice. El número total de iteraciones será la cantidad de aristas del grafo.
→ $|E|$ iteraciones
- El costo total del algoritmo es $(|V|^2 + |E|)$ es $O(|V|^2)$




Algoritmo de Dijkstra

Tiempo de ejecución (II)

Optimización: la operación *vérticeDesconocidoMenorDist* es más eficiente si almacenamos las distancias en una heap.

- La operación *vérticeDesconocidoMenorDist* es $O(\log|V|)$ y dado que se realiza $|V|$ veces
 - ➔ el costo total de *vérticeDesconocidoMenorDist* es $O(|V| \log |V|)$
- El bucle *para* de la línea (7) supone modificar la prioridad (distancia) y reorganizar la heap. Cada iteración es $O(\log|V|)$
 - ➔ realiza $|E|$ iteraciones, $O(|E| \log|V|)$
- El costo total del algoritmo es $(|V| \log|V| + |E| \log|V|)$ es $O(|E| \log|V|)$



Algoritmo de Dijkstra

Tiempo de ejecución (III)

Variante: usando heap la actualización de la línea (10) se puede resolver insertando w y su nuevo valor D_w cada vez que éste se modifica.

- El tamaño de la heap puede crecer hasta $|E|$.

Dado que $|E| \leq |V|^2$, $\log |E| \leq 2 \log |V|$, el costo total del algoritmo no varía

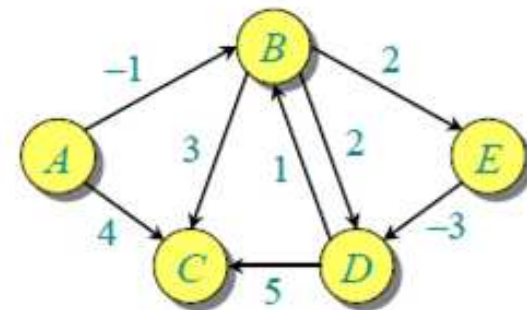
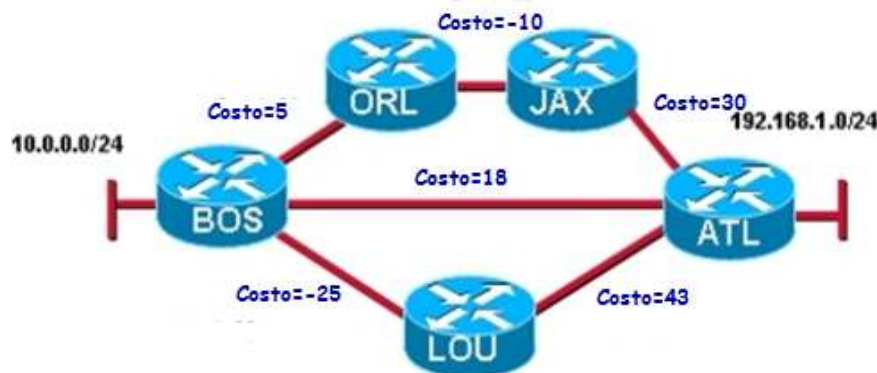
- El costo total del algoritmo es $O(|E| \log |V|)$

Algoritmos de Caminos mínimos

Grafos con pesos positivos y negativos

Ejemplos:

- Simulaciones científicas
- Redes de flujo
- Protocolos de ruteo basados en vector de distancias

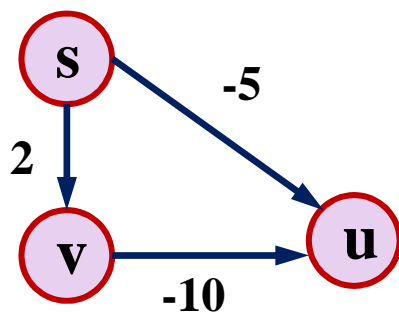


Algoritmos de Caminos mínimos

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



Algoritmos de Caminos mínimos

Grafos con pesos positivos y negativos (cont.)

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 $O(|V| |E|)$



Algoritmos de Caminos mínimos

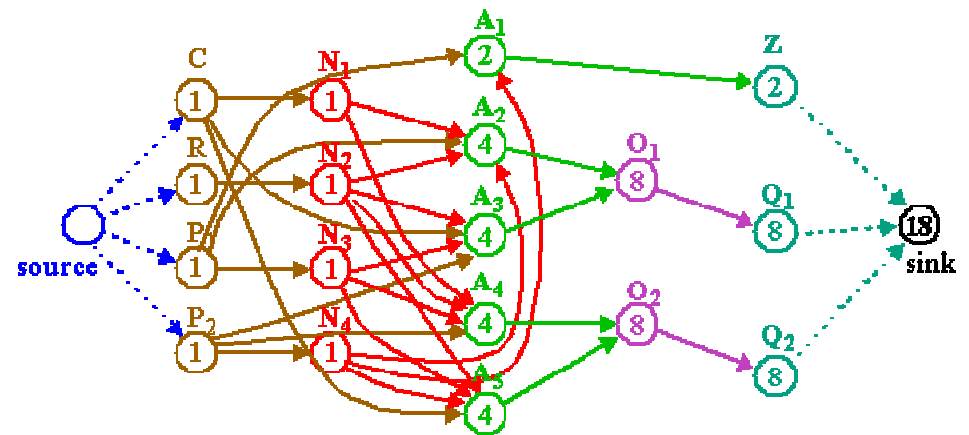
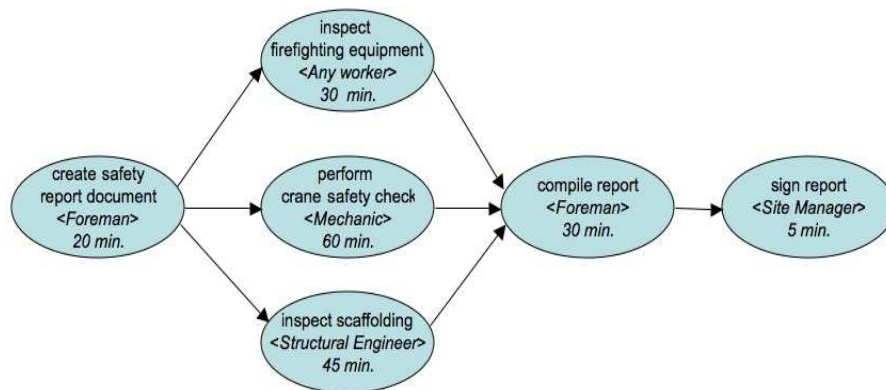
Grafos con pesos positivos y negativos (cont.)

```
Camino_min_GrafoPesosPositivosyNegativosG,s) {  
(1)    $D_s = 0$ ; Encolar (Q,s);  
(2)   Mientras (not esVacio(Q)) {  
(3)       Desencolar(Q,u);  
(4)       para c/vértice  $w \in V$  adyacente a  $u$  {  
(5)           si ( $D_w > D_u + c(u,w)$ ) {  
(6)                $D_w = D_u + c(u,w)$ ;  
(7)                $P_w = u$ ;  
(8)               si (w no está en Q)  
(9)                   Encolar(Q,w);  
(10)          }  
(11)      }  
(12)  }
```

Algoritmos de Caminos mínimos

Grafos acíclicos

- Encontrar la ganancia máxima en un período de tiempo
- Determinar el tiempo requerido para completar una tarea





Algoritmos de Caminos mínimos

Grafos acíclicos

- Estrategia: Orden Topológico
 - Optimización del algoritmo de Dijkstra
 - La selección de cada vértice se realiza siguiendo el orden topológico
 - Esta estrategia funciona correctamente, dado que al seleccionar un vértice v , no se va a encontrar una distancia d_v menor, porque ya se procesaron todos los caminos que llegan a él

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



Algoritmos de Caminos mínimos

Grafos acíclicos

```
Camino_min_GrafoDirigidoAcíclico(G,s){
```

```
    Ordenar topológicamente los vértices de G;
```

```
    Inicializar Tabla de Distancias(G, s);
```

```
    para c/vértice  $u$  del orden topológico
```

```
        para c/vértice  $w \in V$  adyacente a  $u$ 
```

```
            si  $(D_w > D_u + c(u,w))$  {
```

```
                 $D_w = D_u + c(u,w);$ 
```

```
                 $P_w = u;$ 
```

```
            }
```

```
}
```



Algoritmos de Caminos mínimos

Grafos acíclicos

```
Camino_min_GrafoDirigidoAcíclico( $G, s$ ) {  
    Calcular el grado_in de todos los vértices;  
    Encolar en  $Q$  los vértices con grado_in = 0;  
    para cada vértice  $v \in V$   
         $D_v = \infty$ ;  $P_v = 0$ ;  
     $D_s = 0$ ;  
    Mientras (!esVacio( $Q$ )) {  
        Desencolar( $Q, u$ );  
        para  $c$ /vértice  $w \in V$  adyacente a  $u$  {  
            Decrementar grado de entrada de  $w$   
            si (grado_in[ $w$ ] = 0)  
                Encolar( $Q, w$ );  
            si ( $D_u \neq \infty$ )  
                si  $D_w > D_u + c(u, w)$  {  
                     $D_w = D_u + c(u, w)$ ;  
                     $P_w = u$ ;  
                }  
        }  
    }  
}
```



Camínos m nimos entre todos los pares de v rtices

➤ Estrategia: Algoritmo de Floyd

- Lleva dos matrices D y P, ambas de $|V| \times |V|$



Matriz de costos
m nimos



Matriz de v rtices
intermedios

El costo total del algoritmo es $O(|V|^3)$

Algoritmo de Floyd

Toma cada vértice como intermedio, para
calcular los caminos

```
para k=1 hasta cant_Vértices(G)
  para i=1 hasta cant_Vértices(G)
    para j=1 hasta cant_Vértices(G)
      si ( $D[i,j] > D[i,k] + D[k,j]$ ) {
         $D[i,j] = D[i,k] + D[k,j]$ ;
         $P[i,j] = P[k,j]$ ;
      }
```

Distancia entre los
vértices i y j , pasando
por k



Agenda - Grafos

- Caminos de costo mínimo
- Árbol de expansión mínimo



Agenda – Grafos

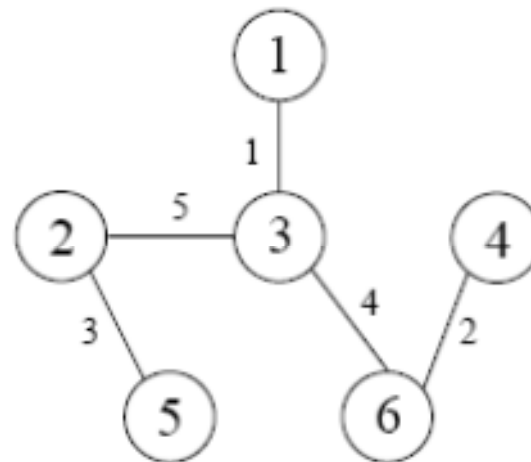
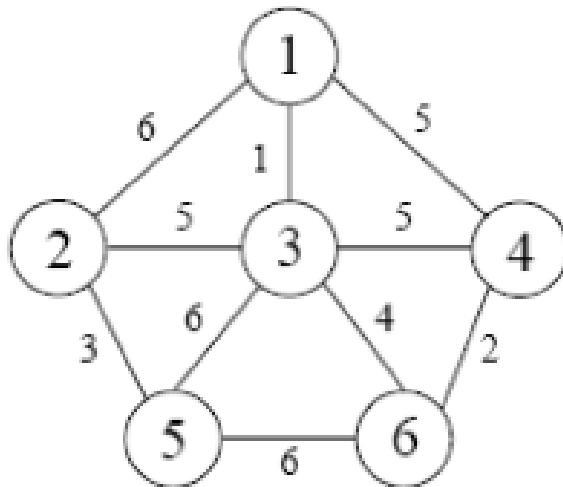
- Árbol de expansión mínimo
 - Definición
 - Aplicaciones
 - Algoritmo de Prim
 - Algoritmo de Kruskal

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.





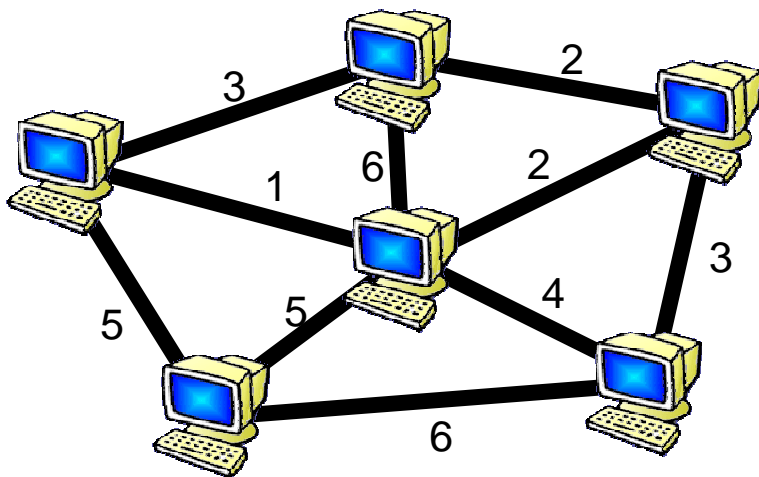
Árbol de expansión mínima

Aplicaciones

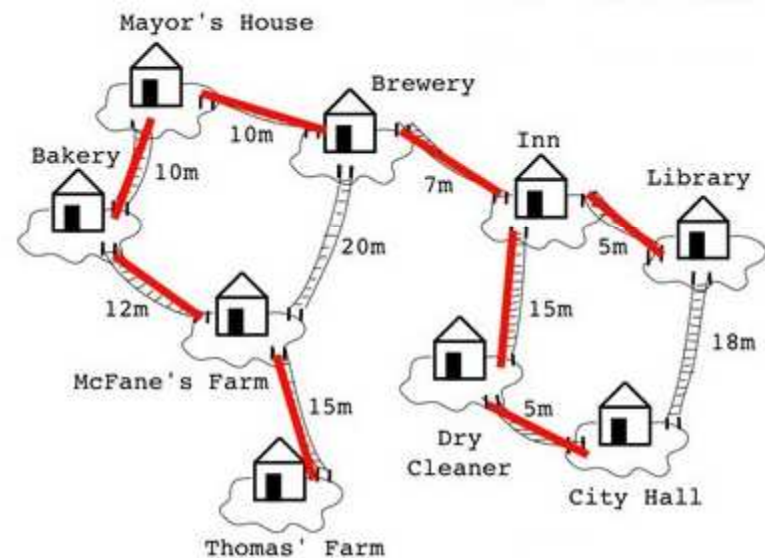
- Construcción de tendidos eléctricos
- Diseño de redes de tuberías
- Cableado de redes de comunicaciones
- Diseño de redes de logística y transporte
- Taxonomías
-

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

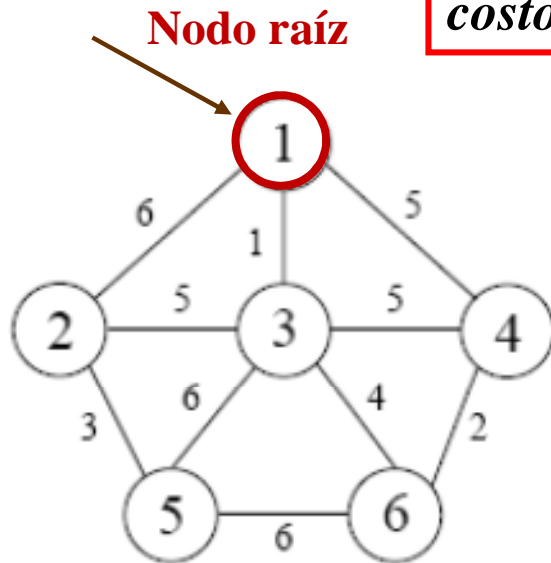
Algoritmo de Prim

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

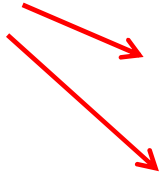
Implementación

1º Paso

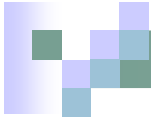
Vértice elegido



Vértices actualizados



<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	6	1	0
3	1	1	0
4	5	1	0
5	∞	0	0
6	∞	0	0



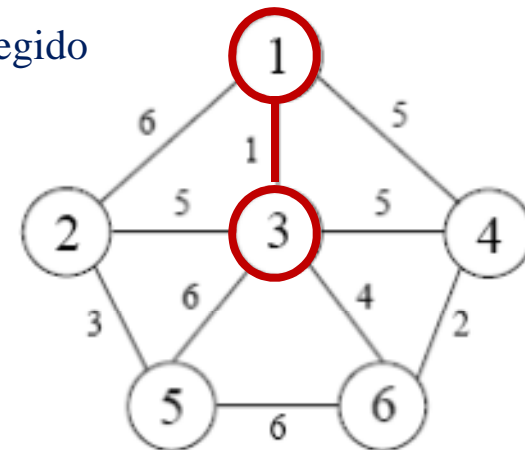
Algoritmo de Prim

Implementación

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
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

2º Paso

Vértice elegido

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	6	1	0
3	1	1	1
4	5	1	0
5	∞	0	0
6	∞	0	0

Vértices actualizados

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	5	3	0
3	1	1	1
4	5	1	0
5	6	3	0
6	4	3	0

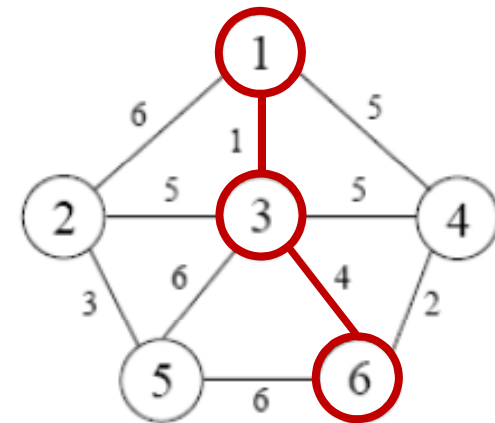
Algoritmo de Prim

Implementación

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
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

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
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

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	5	3	0
3	1	1	1
4	2	6	0
5	6	3	0
6	4	3	0

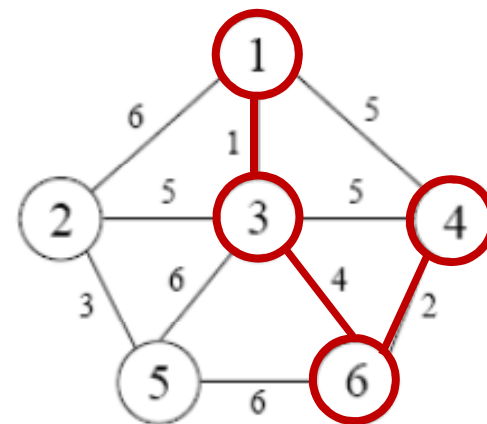
Algoritmo de Prim

Implementación

Vértice elegido →

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	5	3	0
3	1	1	1
4	2	6	1
5	6	3	0
6	4	3	1

4° Paso



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



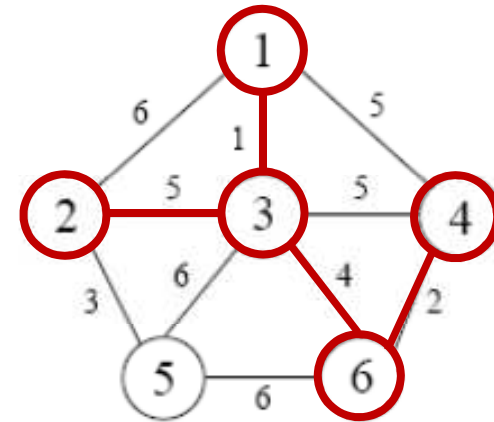
Algoritmo de Prim

Implementación

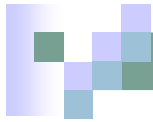
<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	6	3	0
6	4	3	1

Vértice elegido

5° Paso



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



Algoritmo de Prim

Implementación

5° Paso

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
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

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
1	0	0	1
2	5	3	1
3	1	1	1
4	2	6	1
5	3	2	0
6	4	3	1

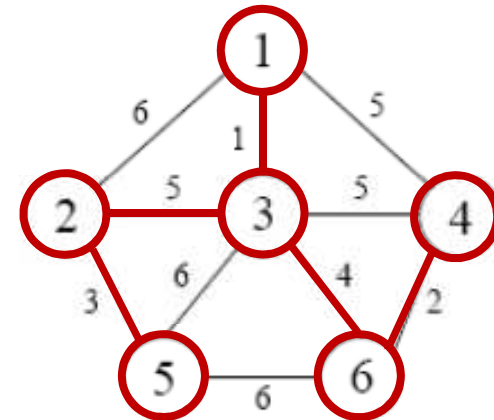
Algoritmo de Prim

Implementación

<i>V</i>	<i>Costo</i>	<i>W</i>	<i>Conoc.</i>
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

Resumiendo la 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 $u \in \text{árbol}$ y $v \notin \text{árbol}$:
 - 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

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 si 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

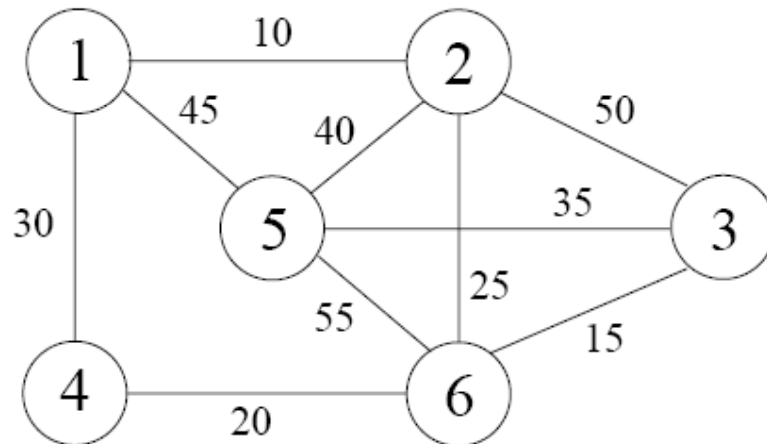
Algoritmo de Kruskal (cont.)

- 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:

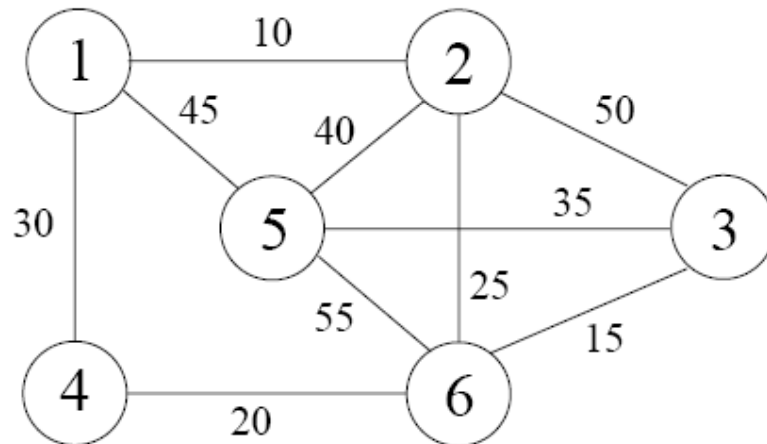


Árbol de expansión mínima

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**

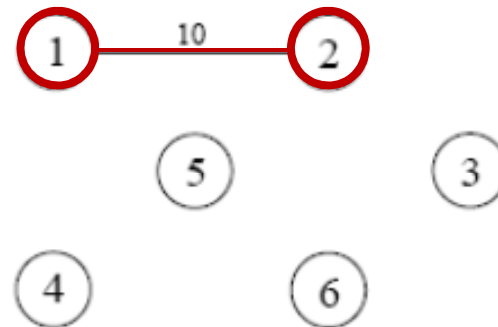
Árbol de expansión mínima

Algoritmo de Kruskal (cont.)

Inicialmente cada vértice está en su propio conjunto



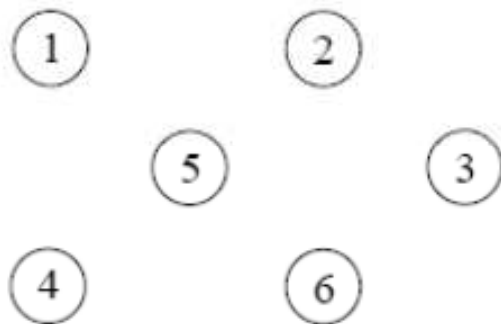
Se agrega la arista (1,2)



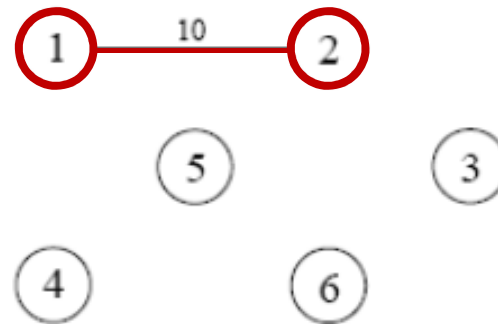
Árbol de expansión mínima

Algoritmo de Kruskal (cont.)

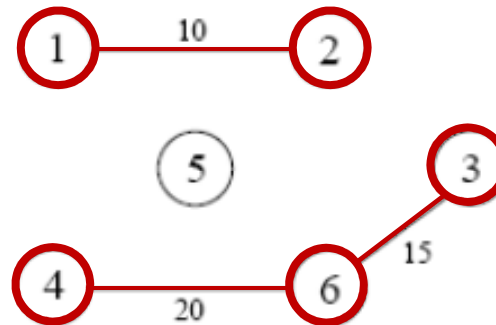
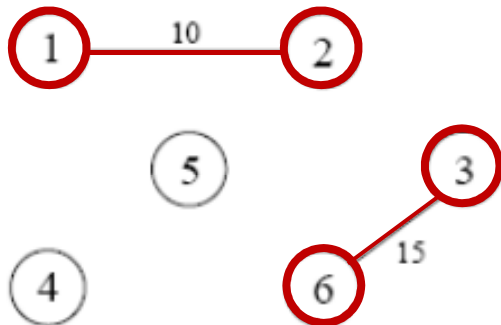
Inicialmente cada vértice está en su propio conjunto



Se agrega la arista (1,2)



Se
agrega
la arista
(3,6)

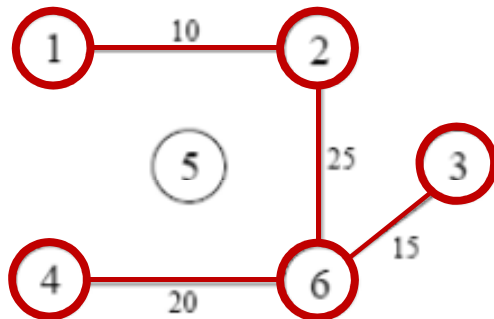


Se
agrega
la arista
(4,6)

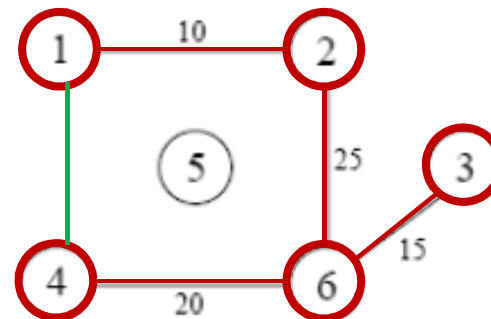
Árbol de expansión mínima

Algoritmo de Kruskal (cont.)

Se agrega la arista (2,6)



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

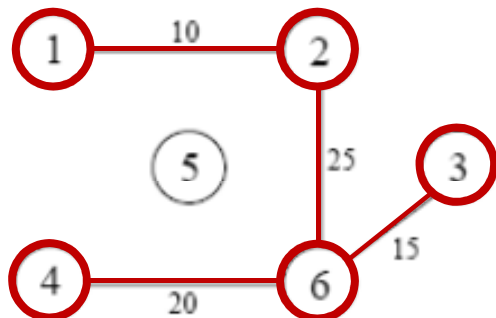


No, porque forma ciclo, ya que pertenece a la misma componente conexa

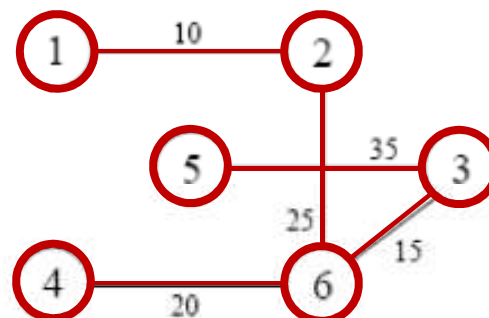
Árbol de expansión mínima

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

