

# Grafos

## *Camino de costo mínimo*

- *Data Structures and Algorithm Analysis in Java; 2nd Ed. Mark Allen Weiss (Capítulo 9)*
- *Estructuras de datos y algoritmos; Mark Allen Weiss. (Capítulo 9)*

# Agenda

- ❖ 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:  $v_1, v_2, v_3, \dots, v_N$

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.)

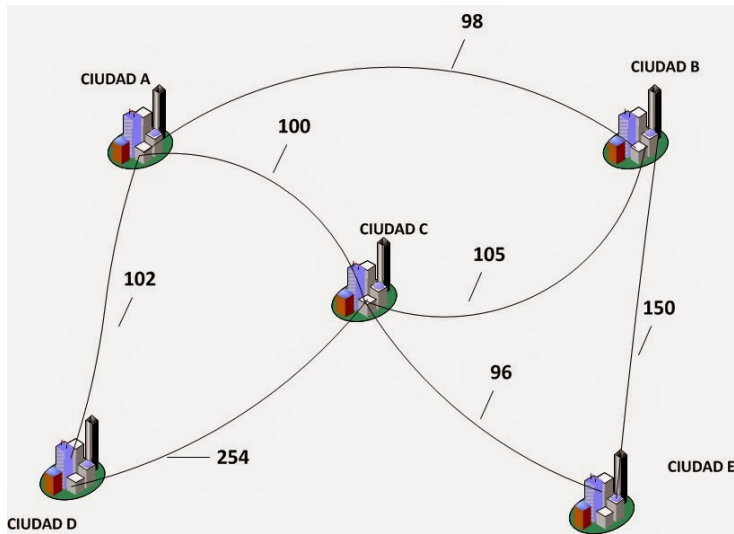
El camino de costo mínimo desde un vértice  $v_i$  a otro vértice  $v_j$  es aquel en que la suma de los costos de las aristas es mínima.

Esto significa que:

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

# Camino de costo mínimo

Ejemplos:



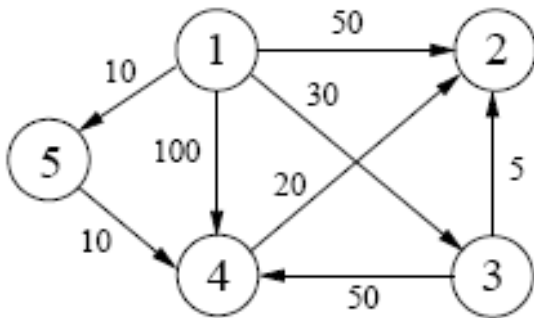
**Ciudades** conectadas por  
**Rutas** con **distancias**



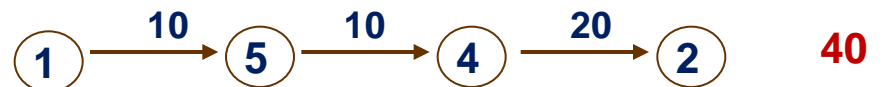
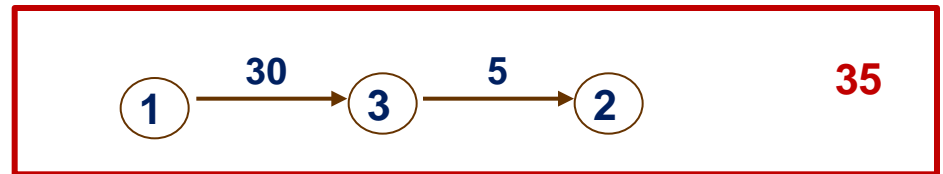
**Personas** conectadas a  
través de las redes sociales

# 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 pesos*

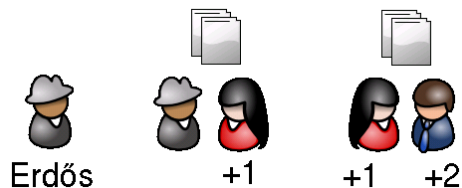
### Ejemplos

#### ➤ Seis grados de separación

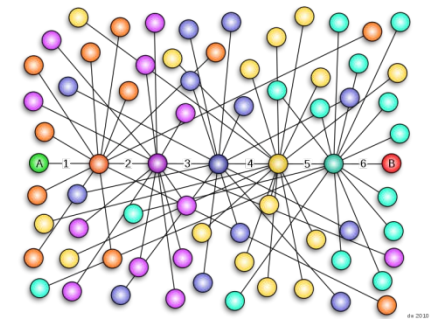
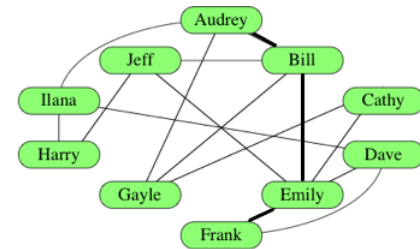
Se le llama *seis grados de separación* a la hipótesis que intenta probar que cualquiera en la Tierra puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no tiene más de cinco intermediarios (conectando a ambas personas con sólo seis enlaces)

#### ➤ 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 pesos

Hilo

Diego Golombek @DiegoGolombek

1. ¿ESTAMOS TAN AISLADOS? El "mito" de los seis grados de separación.

Aislamiento social, cuarentena, #QuedateEnCasa. Pero, ¿estamos tan separados realmente?

9:24 a. m. · 1 abr. 2020 · TweetDeck

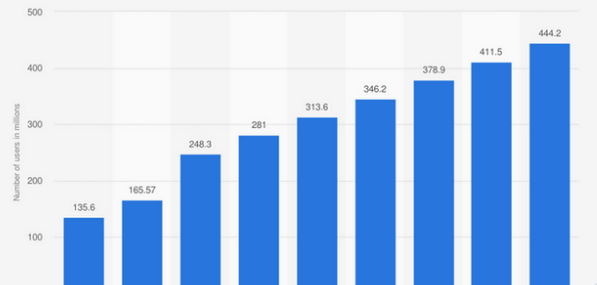
285 Retweets 38 Tweets citados 820 Me gusta



Diego Golombek @DiegoGolombek · 1 abr.

9. Hay pruebas de que seis es multitud. El uso de redes sociales es ideal para encontrar los grados de separación entre dos personas. Recientemente el equipo de Facebook analizó las conexiones de 721 millones de usuarios que combinados alcanzaban a unos 69.000 millones de amigos.

Number of Facebook users in India from 2015 to 2018 with a forecast until 2023 (in millions)



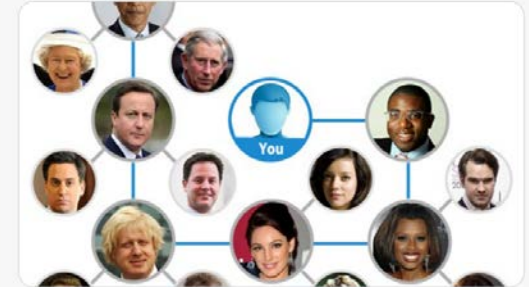
1 6 53



Diego Golombek @DiegoGolombek · 1 abr.

En respuesta a @DiegoGolombek

2. Es parte de una leyenda urbana: entre dos personas cualesquiera en la tierra no hay más de seis grados de separación. En otras palabras, ¿qué nos separa, a ustedes o a mí, de la madre Teresa, de Barack Obama o de Paul McCartney?



2 8 72



Diego Golombek @DiegoGolombek · 1 abr.

3. Es fácil: el socio del cuñado de la prima del intendente que es amigo del empresario que conoce al embajador que alguna vez le dio la mano (o el codo) a cualquiera de estos dos personajes, y paramos de contar.



Diego Golombek @DiegoGolombek · 1 abr.

10. Los primeros cálculos dieron alrededor de 5 pasos entre dos usuarios cualquiera en Facebook.

### Grados de Separación en Facebook

2008

2011



1 6 68

# Algoritmos de Caminos mínimos

## Grafos sin pesos

### Personas relevantes

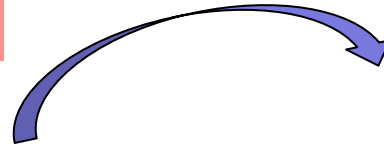


**Diego Golombek** ✓

@DiegoGolombek

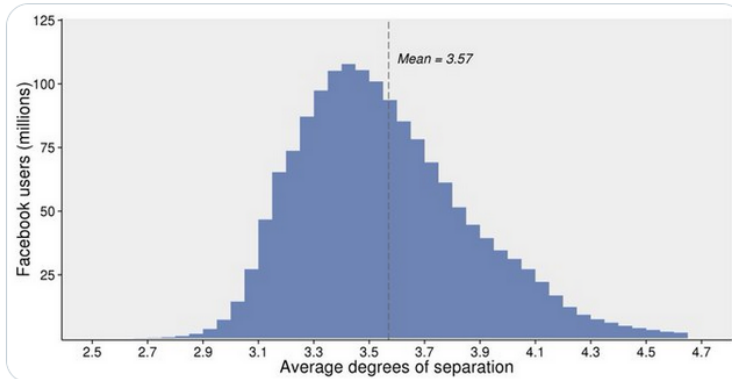
Seguir

Biologo. Investigador / Researcher at CONICET. Profesor Universidad Nacional de Quilmes.



**Diego Golombek** ✓ @DiegoGolombek · 1 abr.

11. Pero un análisis más reciente dio aun menos: entre dos personas cualquiera en la red hay un promedio de... ¡3,5 pasos! Con las redes sociales el mundo es aun más pañuelo.



3

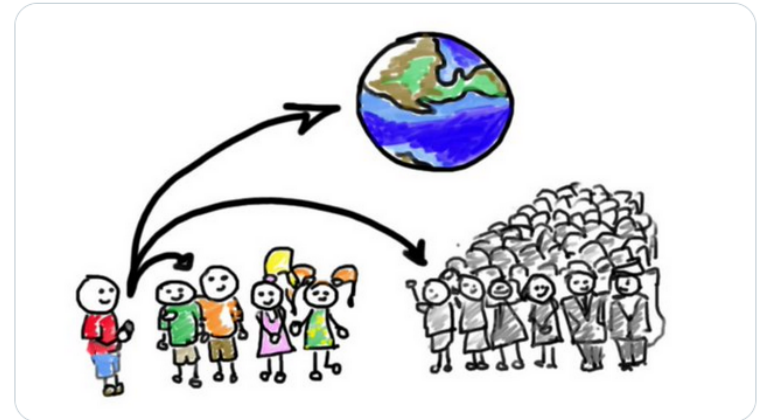
10

80



**Diego Golombek** ✓ @DiegoGolombek · 1 abr.

13. Pero seamos más conservadores: si tenemos 50 amigos que tienen otros 50 amigos que no son amigos nuestros y así sucesivamente... En 5 pasos estaríamos conectados con 3125 millones de personas: algo así como la mitad del mundo.



**Diego Golombek** ✓ @DiegoGolombek · 1 abr.

14. Lo interesante es que a través de estos grados de conexión (o de separación) se transmite de todo: desde potenciales enfermedades hasta la felicidad.

1

9

81



<https://twitter.com/DiegoGolombek/status/1245326104090509312>

# Algoritmos de Caminos mínimos

## *Grafos sin pesos*

- Para cada vértice  $v$  se mantiene la siguiente información:
    - $D_v$  : distancia mínima desde el origen  $s$  (inicialmente  $\infty$  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)
- (este último campo no va a ser necesario para esta clase de grafos)

# Algoritmos de Caminos mínimos

## *Grafos sin pesos*

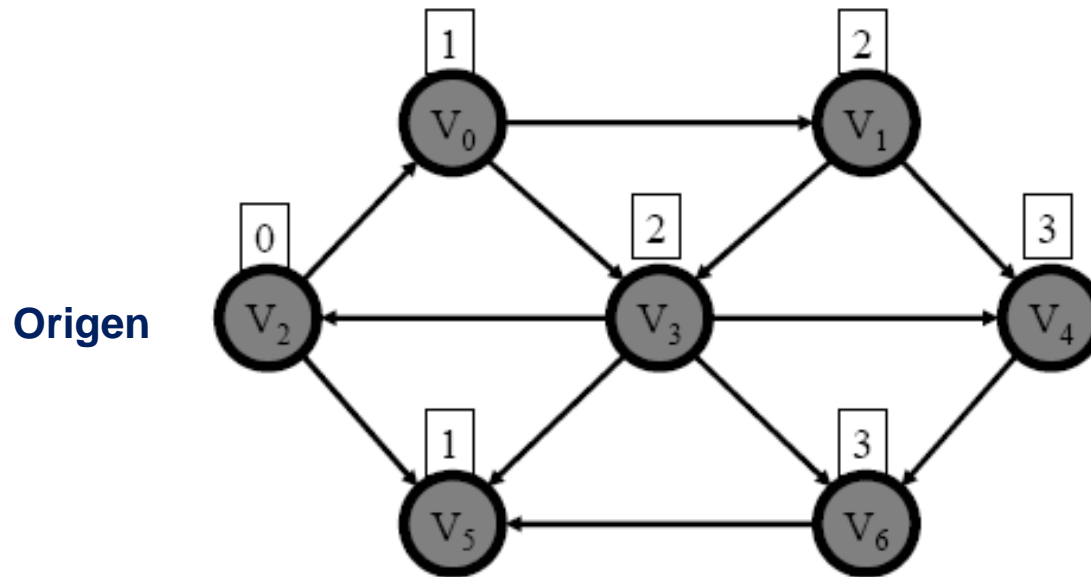
### ➤ Estrategia: Recorrido en amplitud (BFS)

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  $v$

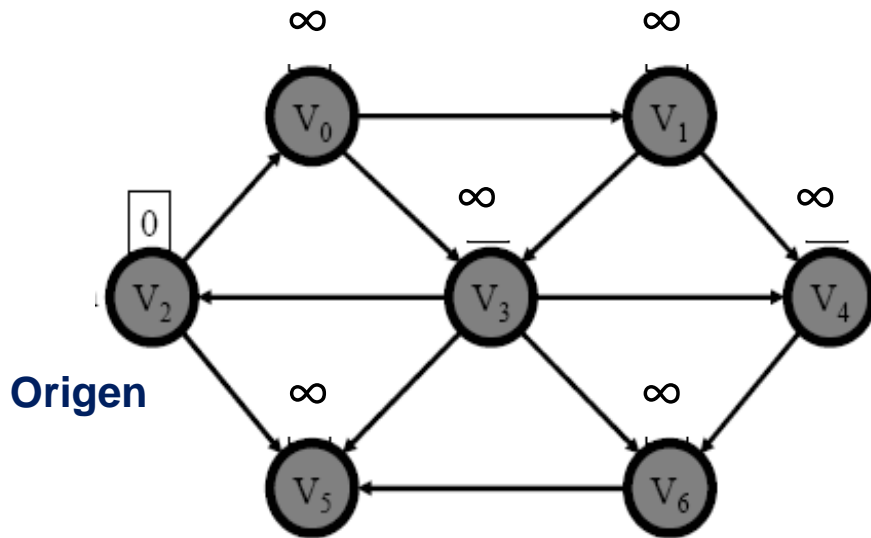
# Algoritmos de Caminos mínimos

## *Grafos sin pesos (cont.)*



# Algoritmos de Caminos mínimos

## *Grafos sin pesos (cont.)*



Valores iniciales de la tabla

$V_i$	$D_v$	$P_v$	Conoc
$V_0$	$\infty$	0	0
$V_1$	$\infty$	0	0
$V_2$	0	0	0
$V_3$	$\infty$	0	0
$V_4$	$\infty$	0	0
$V_5$	$\infty$	0	0
$V_6$	$\infty$	0	0

# Algoritmos de 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 ( $Q,s$ );  $Conoc_s = 1$ ;  
(4)   Mientras (not esVacio( $Q$ ))) {  
(5)       Desencolar ( $Q,u$ );  
(6)       para c/vértice  $w \in V$  adyacente a  $u$  {  
(7)           si ( $w$  no es conocido) {  
(8)                $D_w = D_u + 1$ ;  
(9)                $P_w = u$ ;  
(10)            Encolar( $Q,w$ );  $Conoc_w = 1$ ;  
(11)            }  
(12)        }  
(13)    }  
}
```



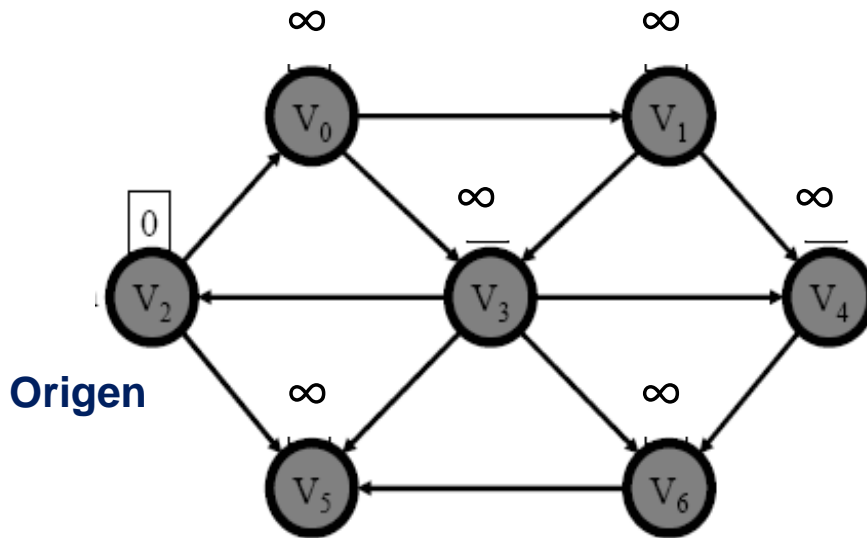
# Algoritmos de Caminos mínimos

## *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); Conocv = 1;  
(4)       Mientras (not esVacio(Q)) {  
(5)           Desencolar (Q,u);  
(6)           para c/vértice  $w \in V$  adyacente a u {  
(7)               si (w no es conocido) {  
(8)                    $D_w = D_u + 1$ ;  
(9)                    $P_w = u$ ;  
(10)                  Encolar(Q,w); Conocw = 1;  
(11)              }  
(12)          }  
(13)      }  
}
```

# Algoritmos de Caminos mínimos

## *Grafos sin pesos (cont.)*

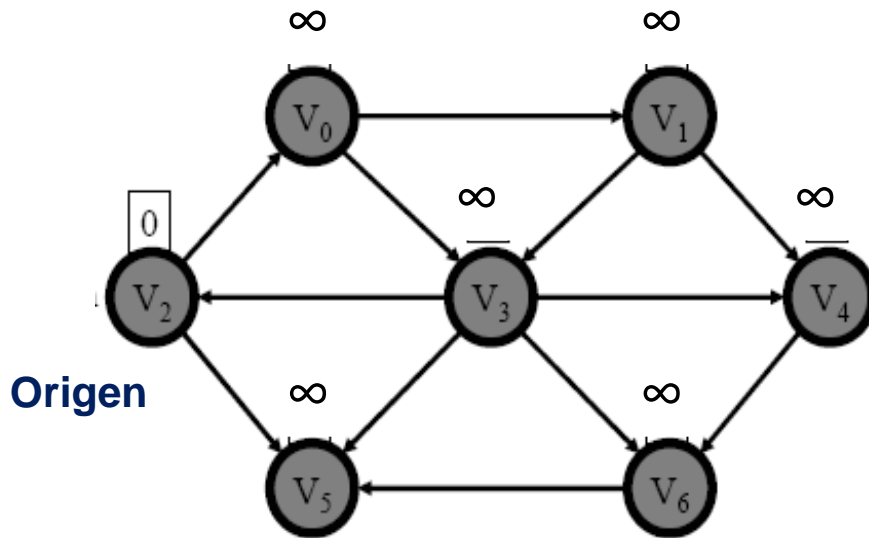


Valores iniciales de la tabla

$V_i$	$D_v$	$P_v$	Conoc
$V_0$	$\infty$	0	0
$V_1$	$\infty$	0	0
$V_2$	0	0	0
$V_3$	$\infty$	0	0
$V_4$	$\infty$	0	0
$V_5$	$\infty$	0	0
$V_6$	$\infty$	0	0

# Algoritmos de Caminos mínimos

## *Grafos sin pesos (cont.)*



Valores iniciales de la tabla

$V_i$	$D_v$	$P_v$
$V_0$	$\infty$	0
$V_1$	$\infty$	0
$V_2$	0	0
$V_3$	$\infty$	0
$V_4$	$\infty$	0
$V_5$	$\infty$	0
$V_6$	$\infty$	0

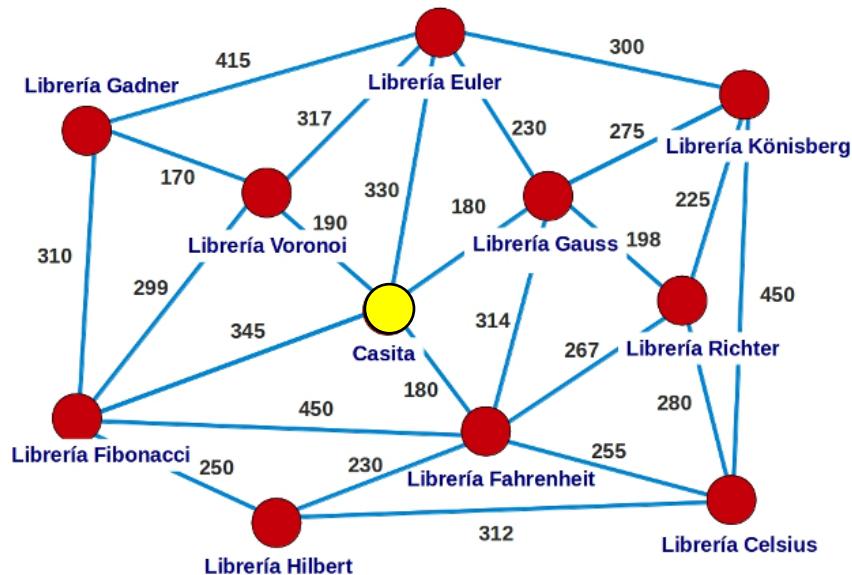
# Algoritmos de Caminos mínimos

## *basado en BFS*

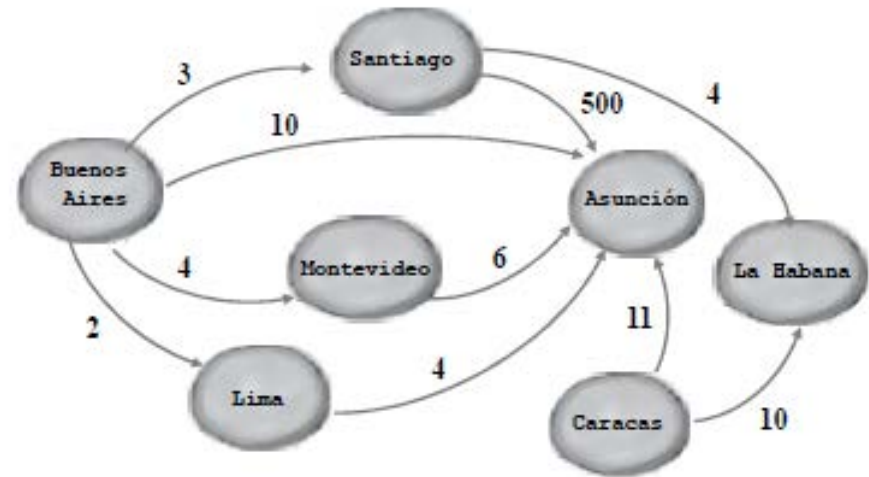
```
Camino_min_GrafoNoPesadoG,s) {  
(1)   para cada vértice  $v \in V$   
(2)        $D_v = \infty$ ;  $P_v = 0$ ;  
(3)    $D_s = 0$ ; Encolar (Q,s);  
(4)   Mientras (not esVacio(Q)) {  
(5)       Desencolar(Q,u);  
(6)       para c/vértice  $w \in V$  adyacente a  $u$  {  
(7)           si ( $D_w = \infty$ ) {  
(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*



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



Encontrar la ruta aérea más corta desde Buenos Aires a Asunción

# *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  $\infty$  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  $v$ )

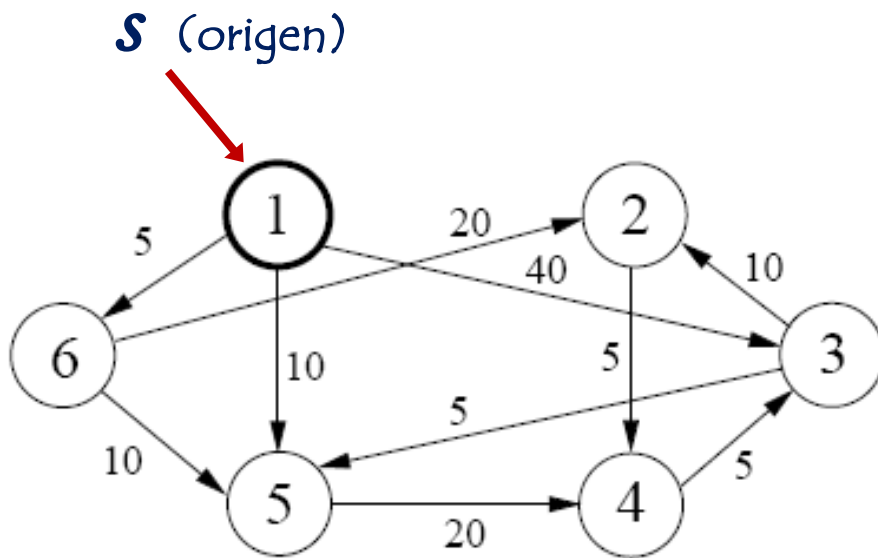
Distancia de  $s$  a  $w$ ,  
pasando por  $v$

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



# Algoritmo de Dijkstra

## Ejemplo

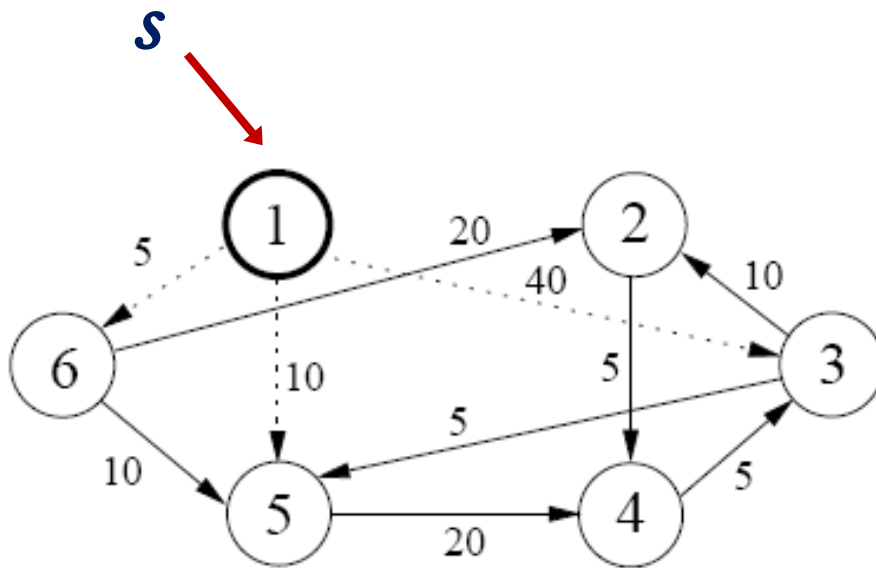


Valores iniciales de la tabla

V	$D_v$	$P_v$	Conoc.
1	0	0	0
2	$\infty$	0	0
3	$\infty$	0	0
4	$\infty$	0	0
5	$\infty$	0	0
6	$\infty$	0	0

# Algoritmo de Dijkstra

## Ejemplo (cont.)

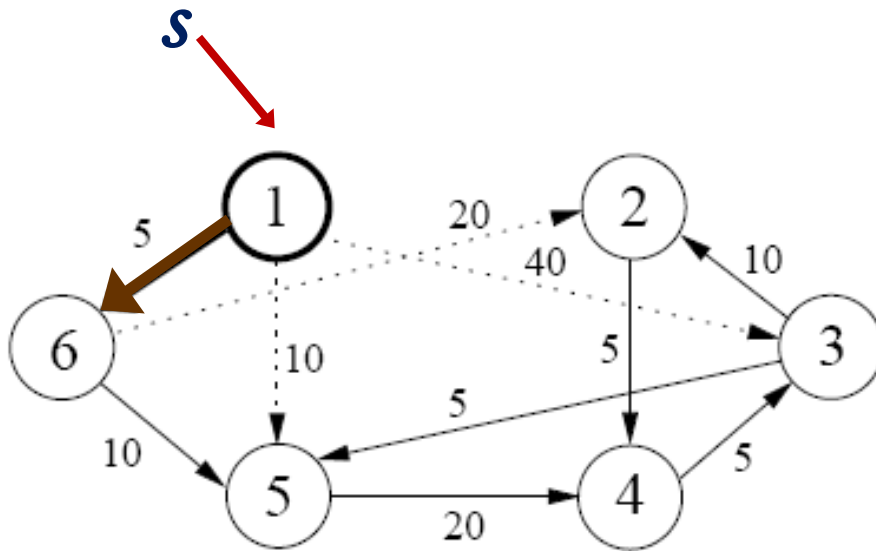


- Valores al seleccionar el vértice 1
- Actualiza la distancia de 3, 5 y 6

V	$D_v$	$P_v$	Conoc.
1	0	0	<b>1</b>
2	$\infty$	0	0
3	<b>40</b>	<b>1</b>	0
4	$\infty$	0	0
5	<b>10</b>	<b>1</b>	0
6	<b>5</b>	<b>1</b>	0

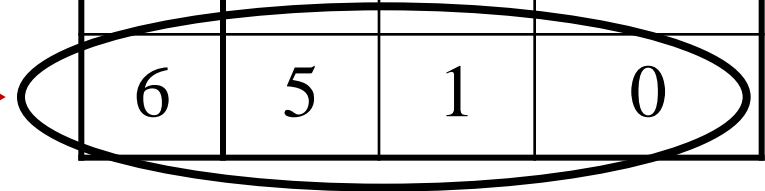
# Algoritmo de Dijkstra

## Ejemplo (cont.)



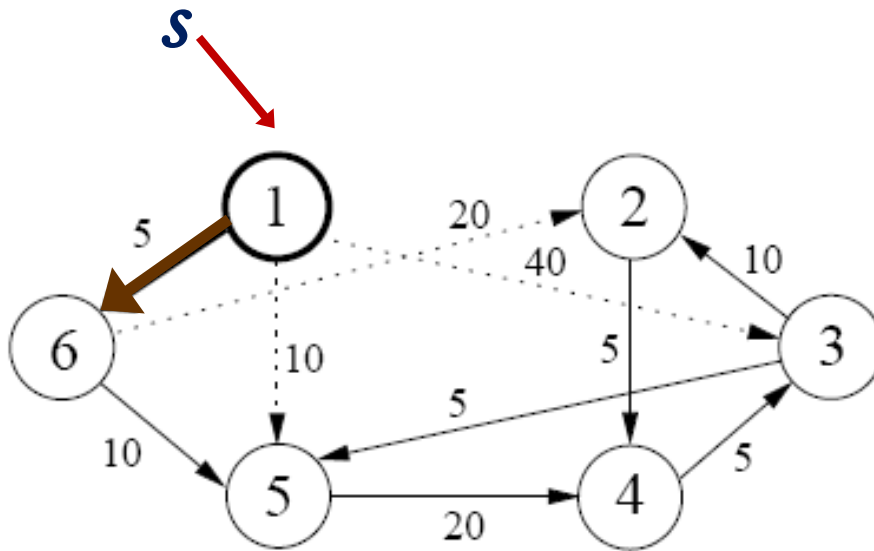
V	$D_v$	$P_v$	Conoc.
1	0	0	1
2	$\infty$	0	0
3	40	1	0
4	$\infty$	0	0
5	10	1	0
6	5	1	0

Próximo vértice a elegir →



# Algoritmo de Dijkstra

## Ejemplo (cont.)

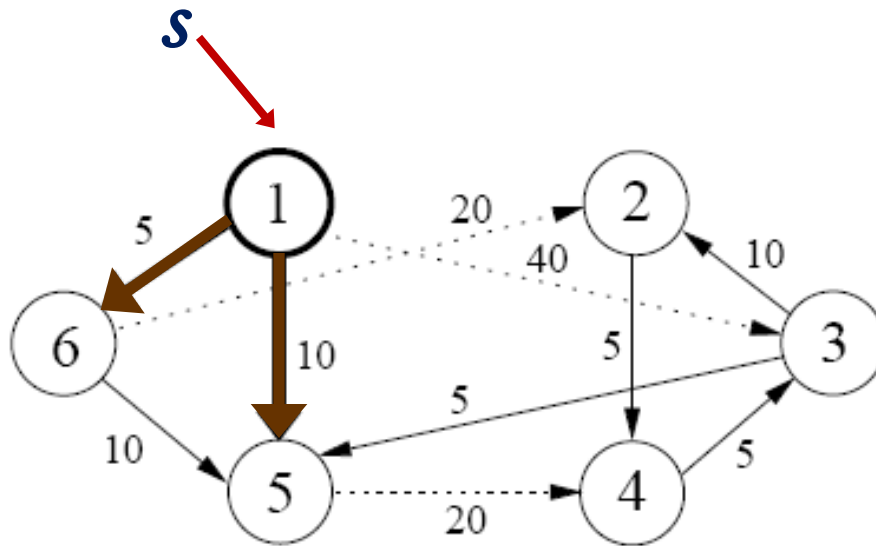


- Valores al seleccionar el vértice 6
- Actualiza la distancia de 2
- La distancia de 5 es mayor que la de la tabla (no se actualiza)

V	$D_v$	$P_v$	Conoc.
1	0	0	1
2	<b>25</b>	<b>6</b>	0
3	40	1	0
4	$\infty$	0	0
5	10	1	0
6	5	1	<b>1</b>

# Algoritmo de Dijkstra

## Ejemplo (cont.)

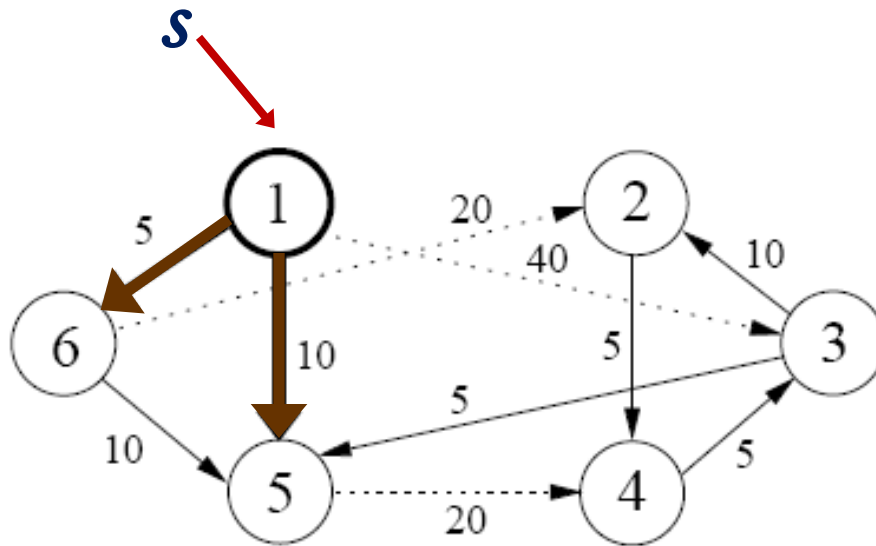


Próximo vértice a elegir

V	$D_v$	$P_v$	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	$\infty$	0	0
5	10	1	0
6	5	1	1

# Algoritmo de Dijkstra

## Ejemplo (cont.)

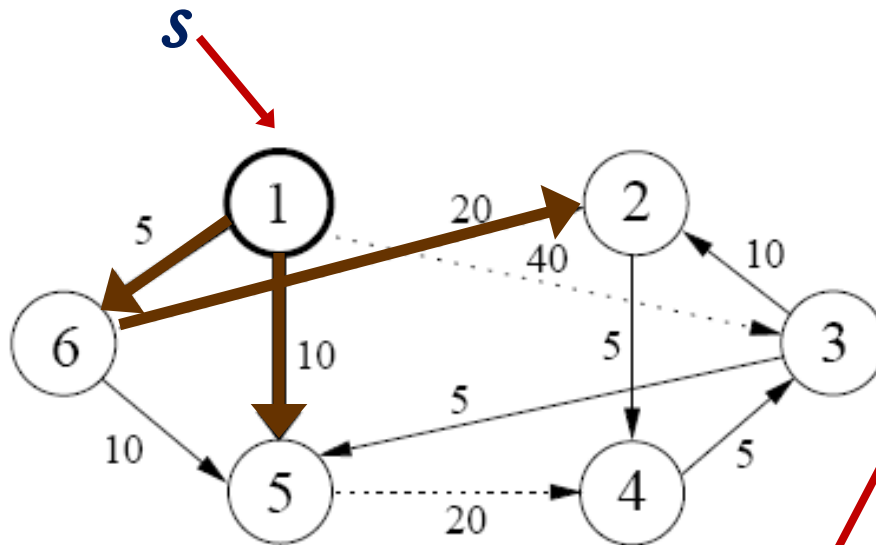


- Valores al seleccionar el vértice 5
- Actualiza la distancia de 4

V	$D_v$	$P_v$	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	<b>30</b>	<b>5</b>	0
5	10	1	<b>1</b>
6	5	1	1

# Algoritmo de Dijkstra

## Ejemplo (cont.)

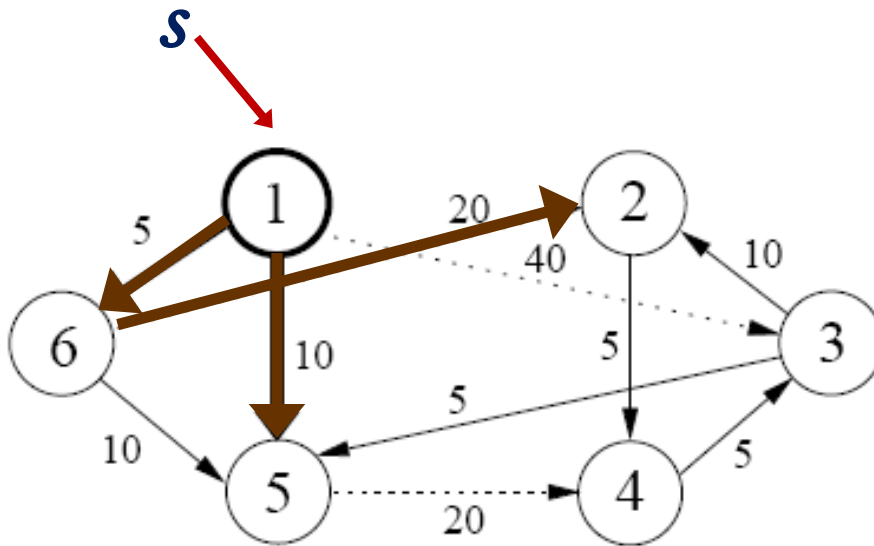


Próximo vértice a elegir

V	$D_v$	$P_v$	Conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	<b>30</b>	<b>5</b>	0
5	10	1	1
6	5	1	1

# Algoritmo de Dijkstra

## Ejemplo (cont.)



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

V	D <sub>v</sub>	P <sub>v</sub>	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

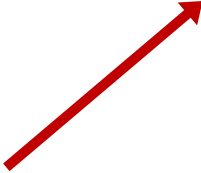


# *Algoritmo de Dijkstra*

## *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)*

Si almacenamos las distancias en un **vector**, tendremos que :

- El bucle *para* de la línea (4) se ejecuta para todos los vértices  
→  $|V|$  iteraciones
- La operación *vérticeDesconocidoMenorDist* -línea (5)- 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* -línea (5)- 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) que se ejecuta para los vértices adyacentes de cada vértice, también supone *modificar* la prioridad (distancia) y *reorganizar* la heap luego de la línea (10). 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* para evitar *modificar* y *reorganizar* la heap:

la actualización de la heap luego de la línea (10) se puede resolver insertando el vértice  $w$  y su nuevo valor  $D_w$  cada vez que éste se modifica.

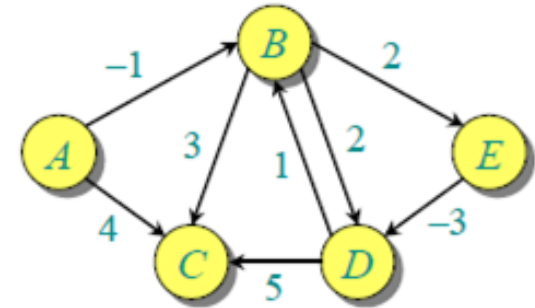
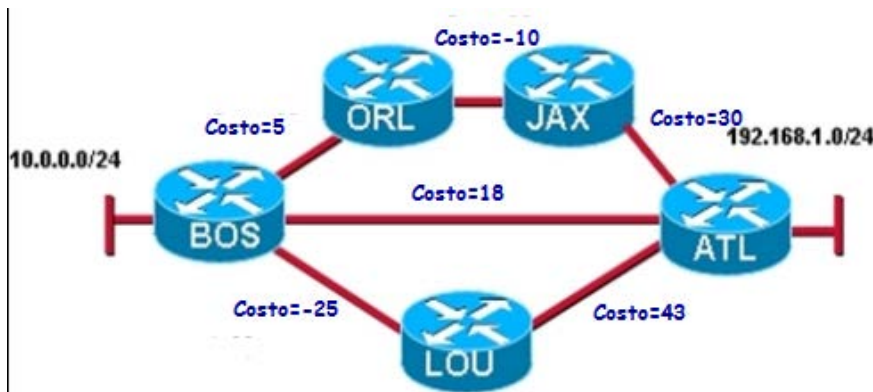
- El tamaño de la heap puede crecer hasta  $|E|$ .  
Dado que  $|E| \leq |V|^2$ ,  $\log |E| \leq 2 \log |V|$ , el costo total del algoritmo no varía
- El costo total del algoritmo es  $O(|E| \log |V|)$

# Algoritmos de Caminos mínimos

## *Grafos con pesos positivos y negativos*

### Ejemplos:

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

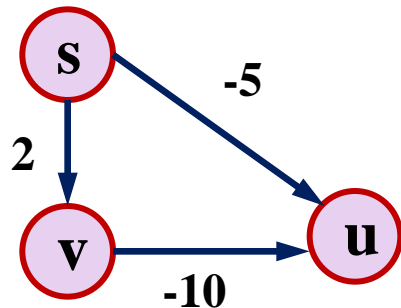


# Algoritmos de Caminos mínimos

## *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

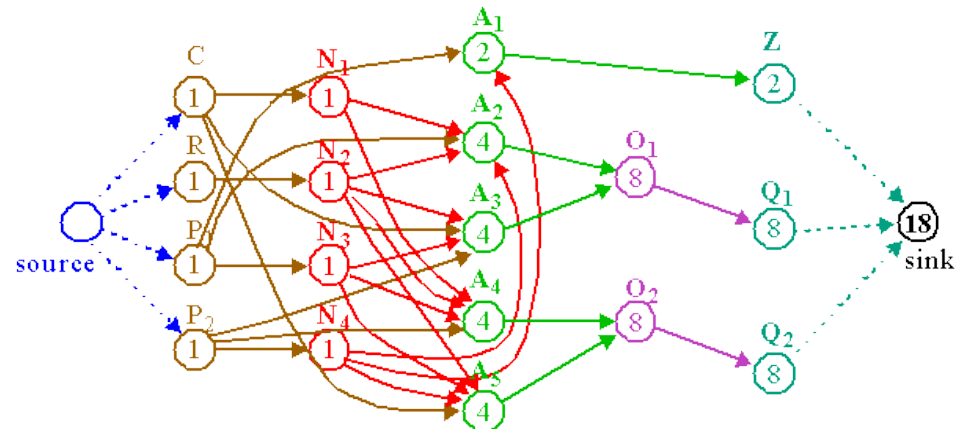
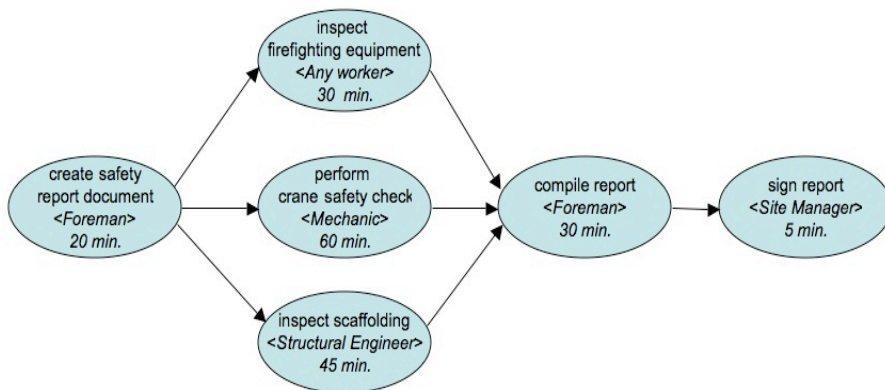
## *Grafos con pesos positivos y negativos (cont.)*

```
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  $D_w > D_u + c(u, w)$  {
                     $D_w = D_u + c(u, w)$ ;
                     $P_w = u$ ;
                }
        }
    }
}

```

# Camino mínimos entre todos los pares de vértices

## ➤ Estrategia: Algoritmo de Floyd

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




Matriz de costos  
mínimos

Matriz de vértices  
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 ( $D[i,j] > D[i,k] + D[k,j]$ ) {
         $D[i,j] = D[i,k] + D[k,j];$ 
         $P[i,j] = k;$ 
      }
```



Distancia entre los vértices  $i$  y  $j$ , pasando por  $k$

<b>Grafos</b>	<b>BFS <math>O(V+E)</math></b>	<b>Dijkstra <math>O(E \log V)</math></b>	<b>Algoritmo modificado (encola vértices) <math>O(V \cdot E)</math></b>	<b>Optimización de Dijkstra (sort top) <math>O(V+E)</math></b>
<b>No pesados</b>	Óptimo	Correcto	Malo	Incorrecto si tiene ciclos
<b>Pesados</b>	Incorrecto	Óptimo	Malo	Incorrecto si tiene ciclos
<b>Pesos negativos</b>	Incorrecto	Incorrecto	Óptimo	Incorrecto si tiene ciclos
<b>Grafos pesados acíclicos</b>	Incorrecto	Correcto	Malo	Óptimo

Correcto → adecuado pero no es el mejor

Malo → una solución muy lenta