

Tema 2 – Algoritmos en redes

Ramón Agüero Calvo
ramon.agueroc@unican.es

Contenidos

- Introducción
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

Contenidos

- **Introducción**
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

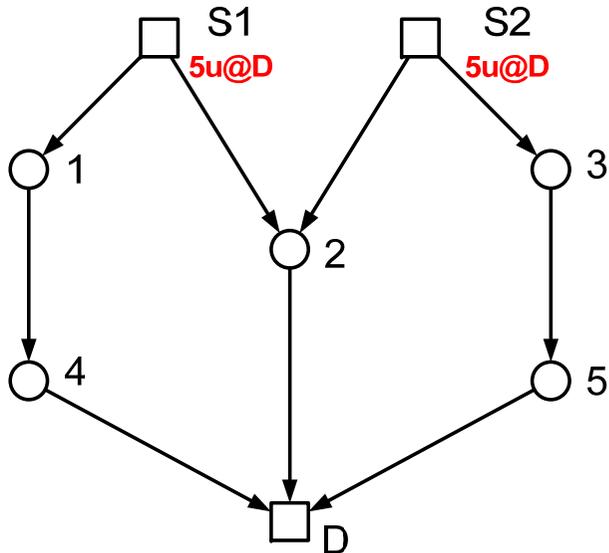
Introducción al encaminamiento en redes

- Objetivo del encaminamiento: Guiar la información entre los nodos origen y destino
- Funciones básicas
 - **Forwarding** o reenvío – un nodo (o *router*) determina la interfaz por la que reenviar un paquete
 - **Routing** o encaminamiento – establecimiento de la ruta (camino) más apropiada entre origen y destino, actualizando las tablas de reenvío en los nodos
- Retos y características
 - Necesidad de mensajes de señalización
 - Coordinación entre los nodos que forman la red
 - Reacción ante fallos en la red → robustez
 - Adaptación a posibles cambios en las condiciones de los enlaces

Elementos y clasificación

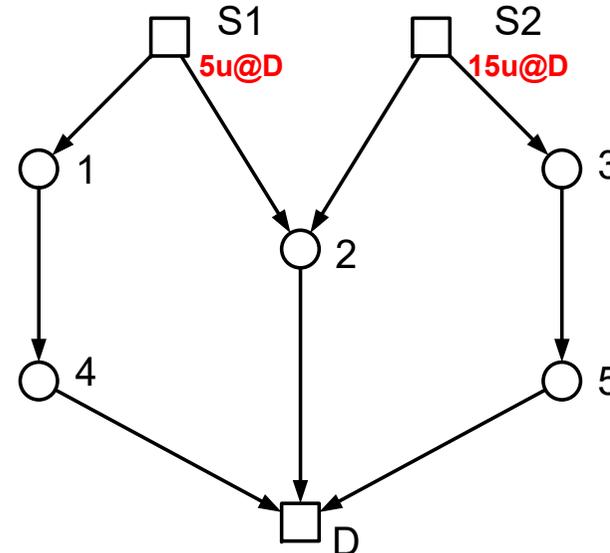
- Métricas de rendimiento
 - # de saltos
 - Coste
 - Retardo
 - Rendimiento (throughput)
- Fuentes de información
 - Local
 - Nodo contiguo (vecino)
 - Nodos de la ruta
 - Todos los nodos
- Dinamismo
 - Estáticos
 - Dinámicos o adaptativos
- Momento de la decisión
 - Paquete (modo datagrama)
 - Establecimiento sesión (modo circuito virtual)
- Lugar de la decisión
 - Cada nodo
 - Nodo central
 - Nodo origen/fuente
- Actualización de la información
 - Continuo
 - Periódico
 - Cambio en la carga
 - Cambio topológico

Problemática del encaminamiento



*S1 y S2 quieren enviar 5 unidades a D
Capacidad de los enlaces: 10 unidades*

- Rutas de menor #saltos
 - S1 → 2 → D
 - S2 → 2 → D
- Rutas con menor retardo
 - S1 → 1 → 4 → D
 - S2 → 3 → 5 → D



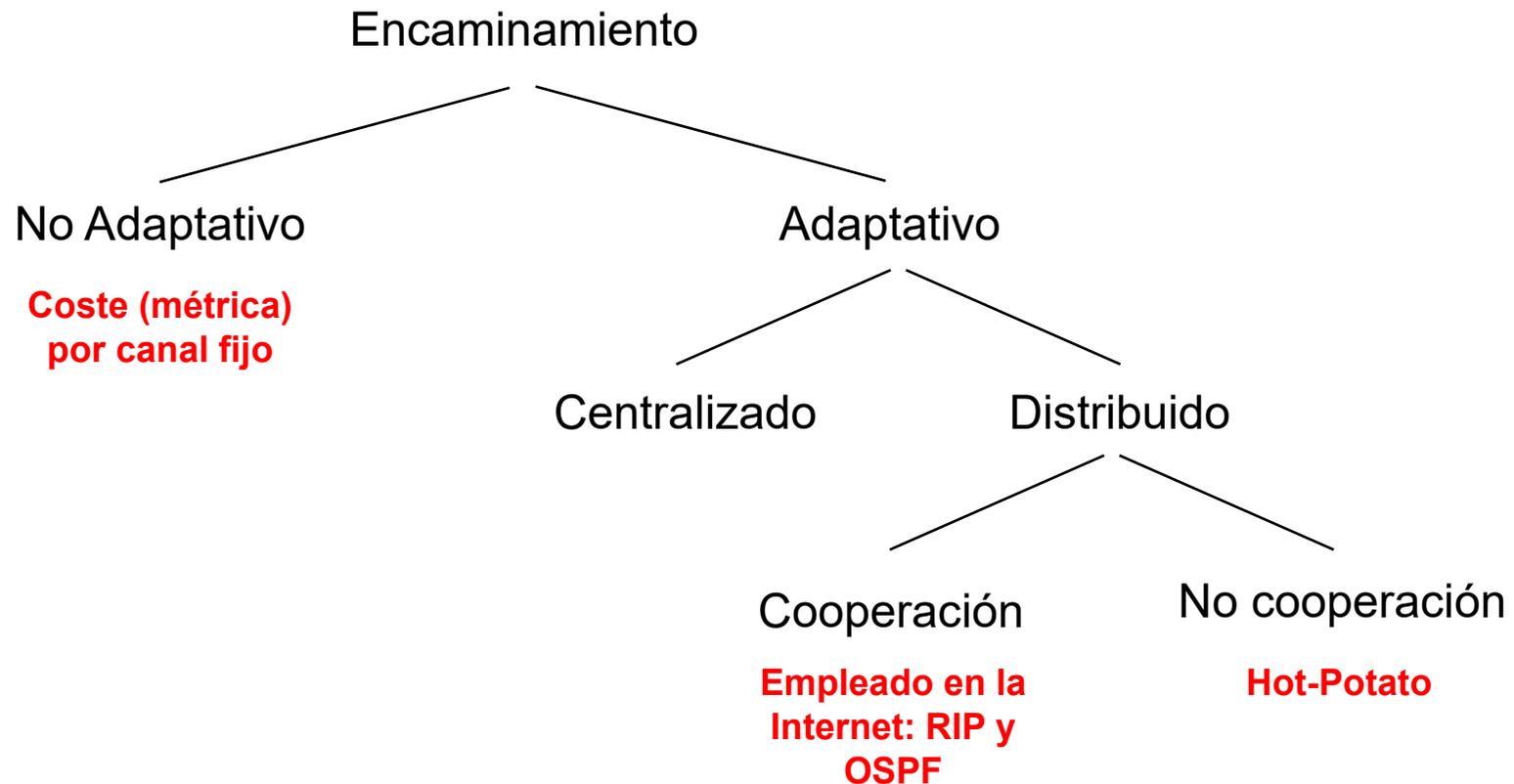
*S1 y S2 quieren enviar 5 y 15 unidades a D
Capacidad de los enlaces: 10 unidades*

- S2 no puede encaminar todo su tráfico por un único camino
 - S2 → 3 → 5 → D [7 unidades]
 - S2 → 2 → D [8 unidades]
- En consecuencia, S1 no empleará la ruta de menor número de saltos
 - S1 → 1 → 4 → D

Tipos de encaminamiento

- **Broadcasting (difusión)**
 - Envío de información a todos los nodos
 - Inundación [*flooding*]
 - Procedimiento sencillo
 - Mucha sobrecarga, por transmisiones y recepciones innecesarias
- **Shortest Path (camino más corto)**
 - Una de las estrategias más empleadas
 - Se minimiza el número de saltos entre origen y destino
 - De manera genérica se podría hablar de coste → Establecimiento de alguna métrica
- **Encaminamiento óptimo**
 - El camino más corto no siempre ofrece el mejor comportamiento
 - Optimización matemática compleja
- **Hot potato (Patata caliente)**
 - Un nodo manda cada paquete por la interfaz menos cargada
 - Se deshace del mismo lo antes posible

Clasificación de encaminamiento



Contenidos

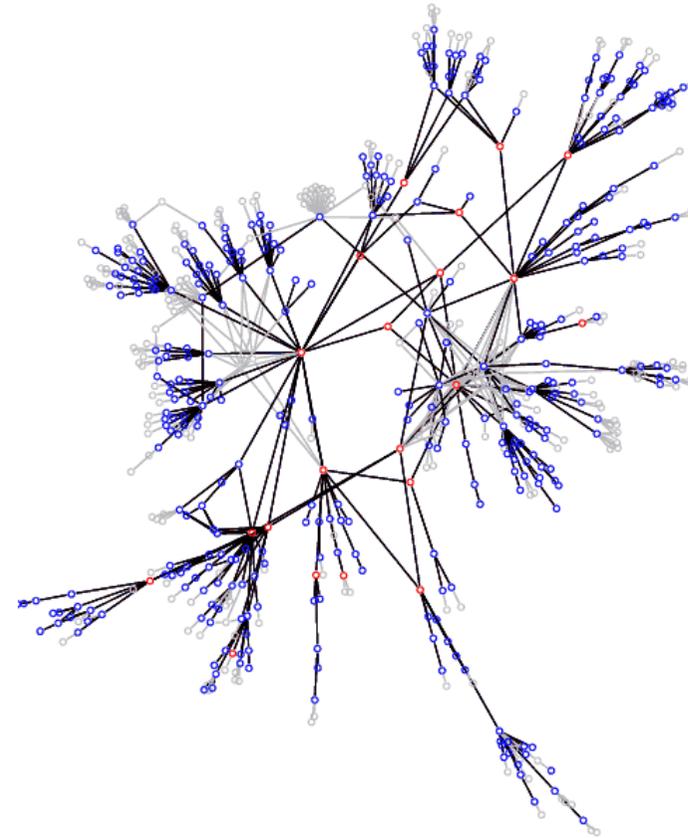
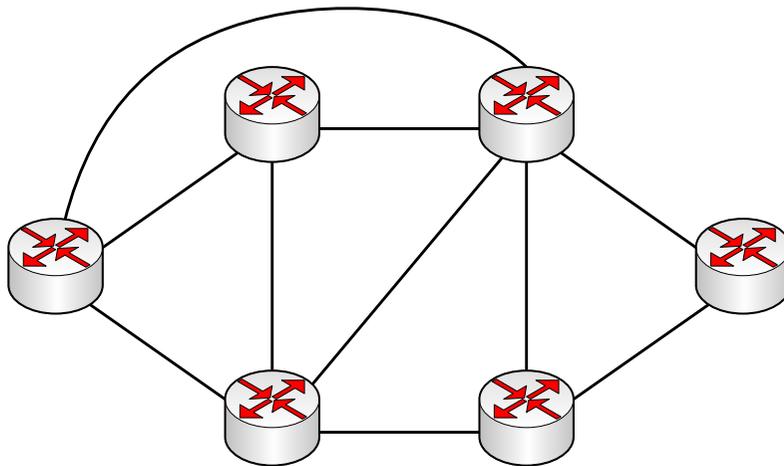
- Introducción
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

¿Qué es un grafo?

- Los grafos son una herramienta matemática que se emplea para formular problemas de encaminamiento
- Definición de un grafo $G=(N, E)$
 - Conjunto de N nodos
 - Colección de E enlaces (*edges*) – cada enlace consta de un par de nodos de N

Los grafos y el encaminamiento

- Al trasladar un grafo a un problema de encaminamiento
 - Los N nodos son los *routers* de la red
 - Los E enlaces se corresponden con los enlaces físicos entre ellos



Conceptos básicos de grafos

- Tipos de grafos
 - Dirigidos: $(u,v) \neq (v,u)$
 - Los enlaces son pares **dirigidos** de nodos
 - No dirigidos: $(u,v) \rightarrow (v,u)$
 - No es necesario establecer un criterio de ordenación a los nodos en cada enlace

- Concatenaciones de enlaces
 - **Walk** (paseo): secuencia de nodos (n_1, n_2, \dots, n_l) tal que cada pareja (n_{i-1}, n_i) es un enlace del grafo
 - **Path** (camino): es un *walk* en el que no hay nodos repetidos
 - **Cycle** (ciclo o bucle): camino con más de un enlace y en el que $n_1 = n_l$

- Grafo conectado
 - Se dice que un grafo está conectado si cualquier par de nodos está conectado por un camino

- En algunas ocasiones puede resultar interesante/necesario asignar costes $c(u,v)$ a los enlaces

Representación de grafos

▪ Lista adyacencia

- Consta de un *array* de \mathbb{N} listas (una por nodo de la red) con punteros a cada nodo con el que tenga un enlace
- Memoria necesaria
 - En un grafo dirigido la suma de punteros coincide con $|E|$
 - En un grafo no dirigido será $2 |E|$
- Ventajas
 - Se pueden asignar costes a los enlaces de manera sencilla
 - Requiere una cantidad menor de memoria, apropiada para grafos sin muchos enlaces (*sparse*)
- Desventajas
 - El proceso de búsqueda puede ser lento

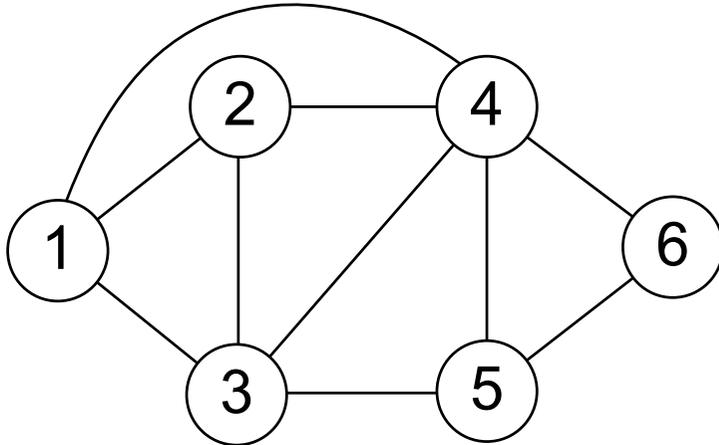
▪ Matriz de adyacencia

- Matriz A de dimensión $\mathbb{N} \times \mathbb{N}$

$$a_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{si } (i, j) \notin E \end{cases}$$

- Con grafos no dirigidos, A es simétrica: $A^T = A$
- El tamaño de A es, para cualquier red, \mathbb{N}^2
- Ventajas
 - La búsqueda es muy rápida
 - Si no se necesitan costes, se puede usar un sólo bit para cada elemento de la matriz
- Desventajas
 - Suele requerir mayor memoria, se usa en grafos más pequeños
 - Si se requieren costes, se necesita mayor capacidad por enlace

Representación de grafos



- Grafo no dirigido

- El número de enlaces en la lista de adyacencia es $2 |E|$
- La matriz de adyacencia es simétrica

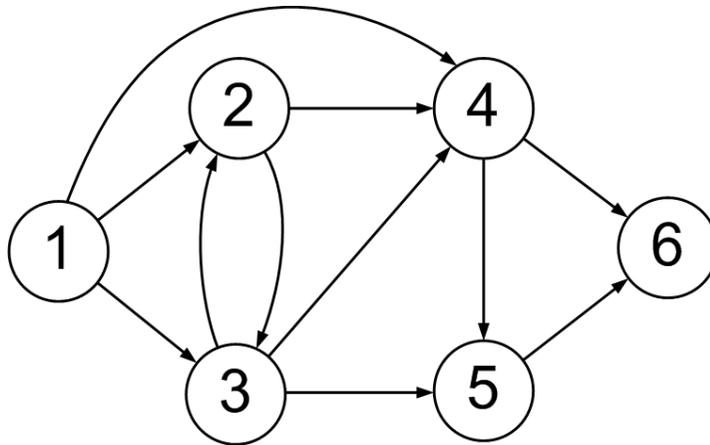
- Lista de adyacencia

- 1 → 2 → 3 → 4
- 2 → 1 → 3 → 4
- 3 → 1 → 2 → 4 → 5
- 4 → 1 → 2 → 3 → 5 → 6
- 5 → 3 → 4 → 6
- 6 → 4 → 5

- Matriz de adyacencia

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Representación de grafos



- Lista de adyacencia

1 → 2 → 3 → 4
2 → 3 → 4
3 → 2 → 4 → 5
4 → 5 → 6
5 → 6
6

- Grafo dirigido

- El número de enlaces en la lista de adyacencia es $|E|$
- La matriz de adyacencia no es simétrica

- Matriz de adyacencia

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Contenidos

- Introducción
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

Búsqueda del camino más corto

- La idea principal es la de encontrar el camino con un coste mínimo entre una fuente (S) y un destino (D)
- Si $c(u,v)$ se mantiene constante para todos los enlaces, la solución es la ruta de menor número de saltos
- Algoritmos con una única fuente: encuentran el camino más corto entre S y el resto de nodos
 - *Dijkstra*
 - *Bellman-Ford*
- Algoritmos para toda la red: encuentran el camino más corto entre todas las posibles parejas de nodos en la red
 - *Floyd-Warshall*
 - *Johnson*

Coste de una ruta

- Métricas aditivas: distancia, tiempo de transferencia, etc
 - Coste camino = $c(l_1) + c(l_2) + c(l_3)$
 - Interesa minimizar el coste total
- Métricas multiplicativas: probabilidad de no fallo, fiabilidad
 - Coste camino = $f(l_1) \cdot f(l_2) \cdot f(l_3)$
 - Interesa maximizar el coste total
- Métricas cuello de botella: capacidad
 - Coste camino = $\min \{ C(l_1), C(l_2), C(l_3) \}$
 - Interesa maximizar el coste total

Algoritmo de Dijkstra

- Encuentra el camino de coste mínimo de una fuente S a todos los nodos en un grafo con costes **NO NEGATIVOS**
- Definiciones previas

- Coste camino $c(p) = \sum_{i=1}^k c(u_{i-1}, u_i)$

- Coste camino mínimo $\delta(u, v) = \begin{cases} \min\{c(p) : u \xrightarrow{p} v\} & \text{si hay camino} \\ \infty & \text{entre } u \text{ y } v \\ & \text{en caso contrario} \end{cases}$

Algoritmo de Dijkstra

■ Variables

- Conjunto de nodos Q para los que no se ha encontrado el camino más corto
- Se mantiene una lista con las distancias a cada nodo $d(u)$

$$\forall u \in \{N - Q\} \quad d(u) = \delta(S, u)$$

■ Algoritmo

- Se busca en Q el nodo cuyo camino de coste mínimo sea el menor

$$d(u) = \min_{v \in Q} d(v)$$

- u se borra de Q
- Si Q es el conjunto vacío ($Q = \emptyset$), se termina el algoritmo
- Para todos los nodos v de Q adyacentes a u

$$d(v) = \min\{d(v), d(u) + c(u, v)\}$$

Algoritmo de Dijkstra

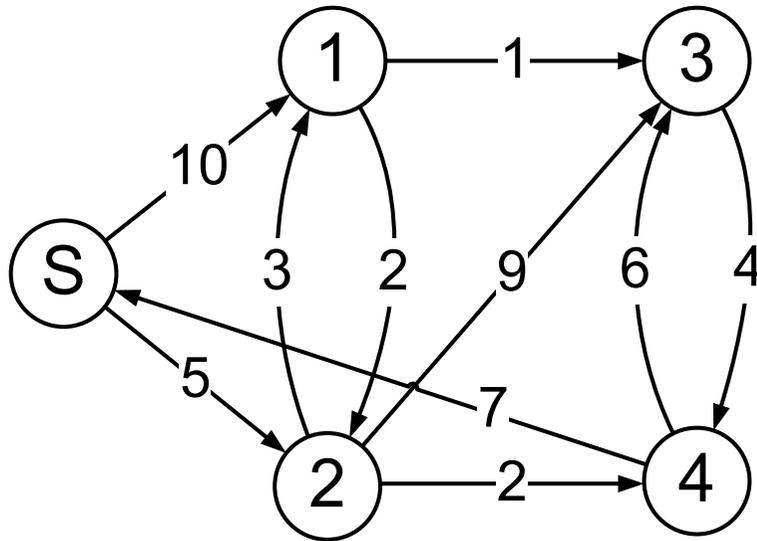
INITIALIZATION

1. $d(S) = 0$
2. for all v in N but S
3. $d(v) = \infty$
4. $Q = N$

MAIN LOOP

5. while $Q \neq \{\emptyset\}$
6. u vertex in Q with $\min\{d(v)\}$
7. delete u from Q
8. for all v in Q adjacent to u
9. if $d(v) > d(u) + c(u, v)$
10. $d(v) = d(u) + c(u, v)$
11. $\text{prev}(v) = u$

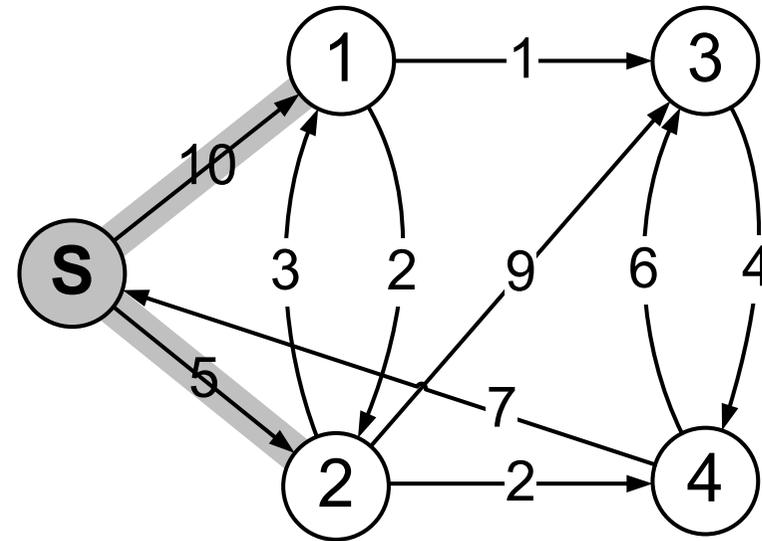
Ejemplo algoritmo de Dijkstra



Inicialización

$$Q = \{S, 1, 2, 3, 4\}$$

$$d = [0, \infty, \infty, \infty, \infty]$$

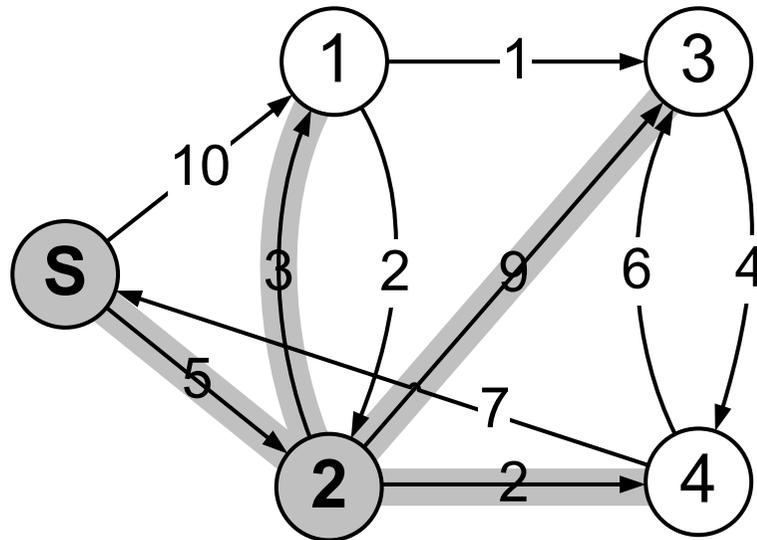


Primera iteración

$$Q = \{1, 2, 3, 4\}$$

$$d = [0, 10, 5, \infty, \infty]$$

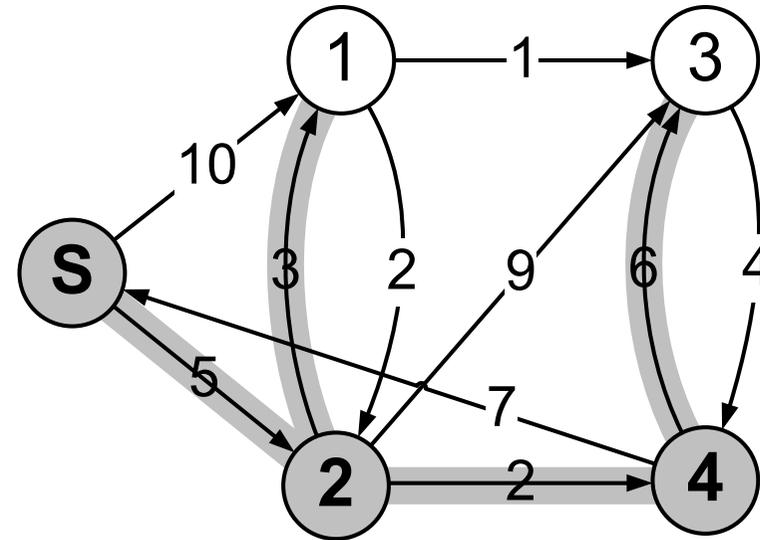
Ejemplo algoritmo de Dijkstra



Segunda iteración

$$Q = \{1, 3, 4\}$$

$$d = [0, 8, 5, 14, 7]$$

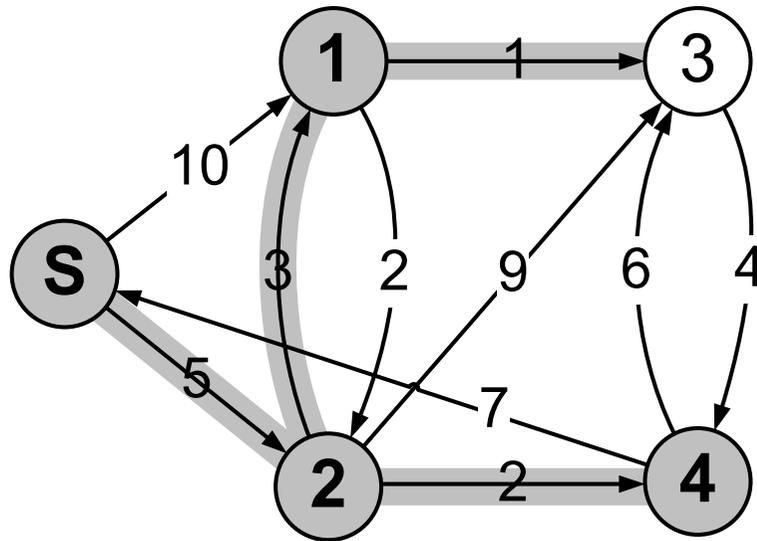


Tercera iteración

$$Q = \{1, 3\}$$

$$d = [0, 8, 5, 13, 7]$$

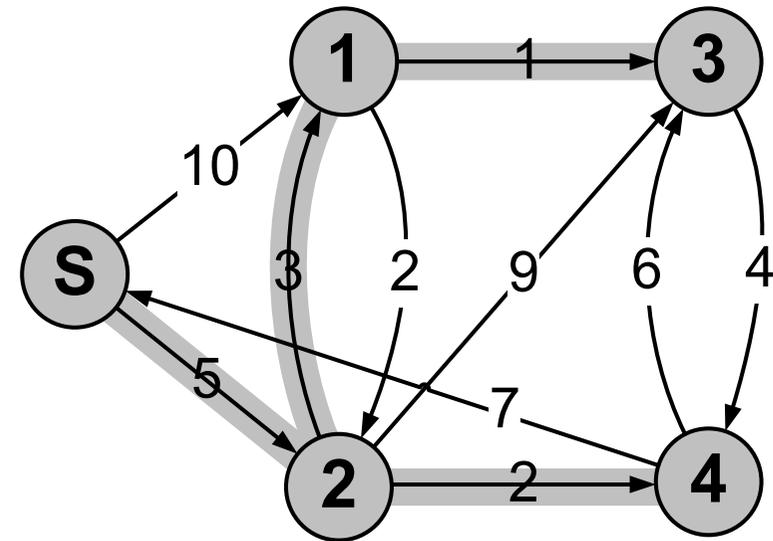
Ejemplo algoritmo de Dijkstra



Cuarta iteración

$$Q = \{3\}$$

$$d = [0, 8, 5, 9, 7]$$



Quinta iteración

$$Q = \{\emptyset\}$$

$$d = [0, 8, 5, 9, 7]$$

Algoritmo de Bellman-Ford

- Al igual que Dijkstra, encuentra el camino más corto de un nodo al resto
- Puede emplearse con redes que tengan enlaces con coste negativo
- Si hay un ciclo negativo en la fuente, Bellman-Ford lo detecta
 - En este caso el camino de coste mínimo **NO** puede solucionarse
- Variables
 - Una lista con los costes de las rutas de S a cualquier nodo $d(u)$
- Algoritmo
 - Se recorre el grafo $N - 1$ veces y se aplica la ecuación de Bellman para los enlaces del grafo

$$d(v) = \min\{d(v), d(u) + c(u, v)\}$$

Algoritmo de Bellman-Ford

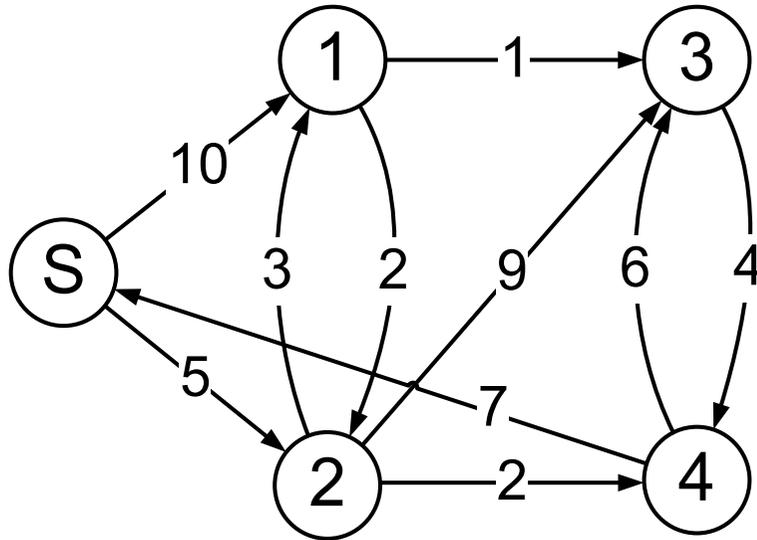
INITIALIZATION

1. $d(S) = 0$
2. for all v in N but S
3. $d(v) = \infty$

MAIN LOOP

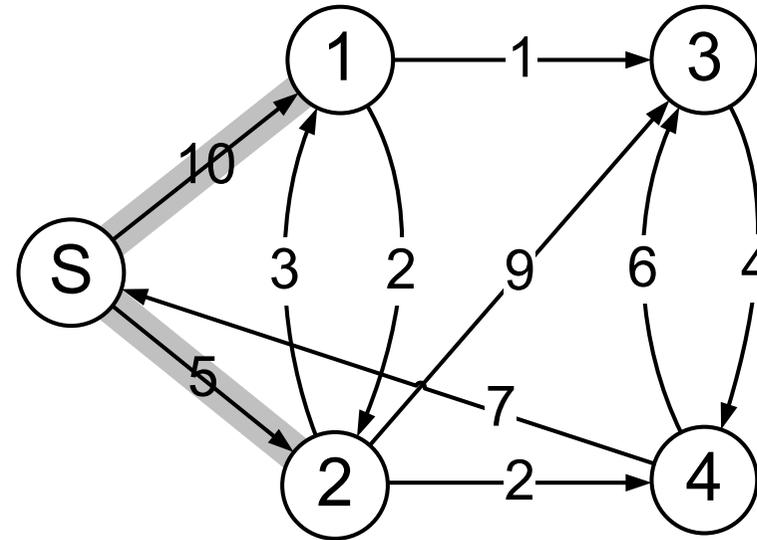
4. for $k = 1$ to $N-1$
5. for each (u, v) in E
6. if $d(v) > d(u) + c(u, v)$
7. $d(v) = d(u) + c(u, v)$
8. $prev(v) = u$

Ejemplo algoritmo de Bellman-Ford



Inicialización

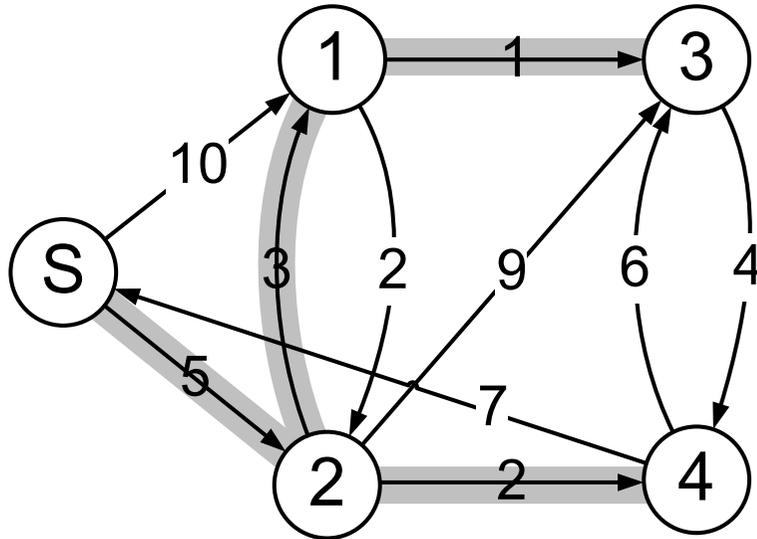
$$d = [0, \infty, \infty, \infty, \infty]$$



Primera iteración

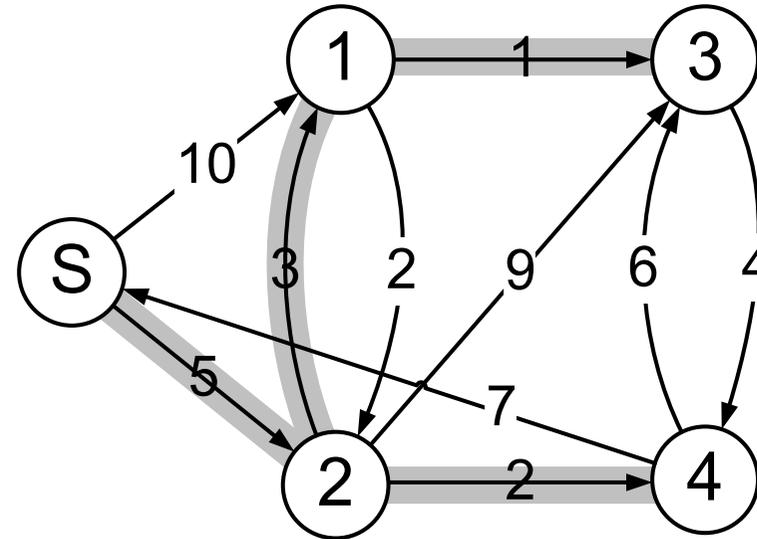
$$d = [0, 10, 5, \infty, \infty]$$

Ejemplo algoritmo de Bellman-Ford



Segunda iteración

$$d = [0, 8, 5, 11, 7]$$



Tercera iteración

$$d = [0, 8, 5, 9, 7]$$

Algoritmo de Floyd Warshall

- Se define $d^k[i,j]$ como el coste del camino más corto entre i y j con la condición de que use únicamente los nodos $1, 2, \dots, k$ como nodos intermedios
- Así, $d^N[i,j]$ representa la distancia del camino más corto entre i y j
- El algoritmo de Floyd Warshall establece iterativamente $d^k[i,j]$ para todas las parejas de nodos (i,j) para $k = 1, 2, \dots, N$
- A partir de $d^{k-1}[i,j]$, el algoritmo calcula $d^k[i,j]$ a partir de la siguiente propiedad

$$d^k[i, j] = \min\{d^{k-1}[i, j], d^{k-1}[i, k] + d^{k-1}[k, j]\}$$

Algoritmo de Floyd Warshall

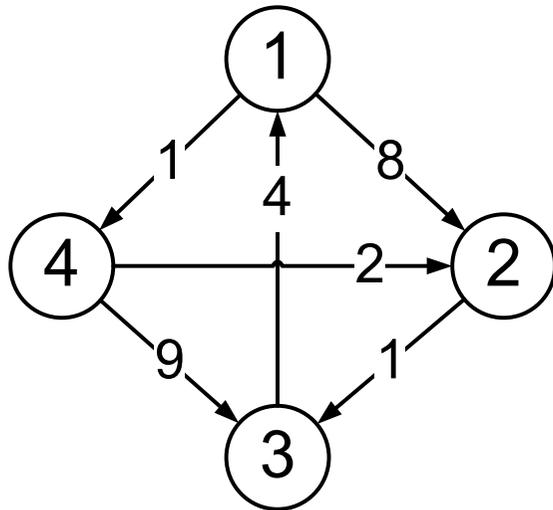
INITIALIZATION

1. for all (u, v) in $N \times N$
2. $d(u, v) = \infty$
3. $\text{pred}(u, v) = \text{NIL}$
4. for all nodes u in N
5. $d(u, u) = 0$
6. for each (u, v) in E
7. $d(u, v) = c(u, v)$
8. $\text{pred}(u, v) = u$

MAIN LOOP

9. for $k = 1$ to N
10. for $u = 1$ to N
11. for $v = 1$ to N
12. if $d(u, v) > d(u, k) + d(k, v)$
13. $d(u, v) = d(u, k) + d(k, v)$
14. $\text{pred}(u, v) = \text{pred}(k, v)$

Ejemplo algoritmo de Floyd Warshall



$$D^{(0)} = \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{pmatrix}$$

$$\Pi^{(0)} = \begin{pmatrix} nil & 1 & nil & 1 \\ nil & nil & 2 & nil \\ 3 & nil & nil & nil \\ nil & 4 & 4 & nil \end{pmatrix}$$

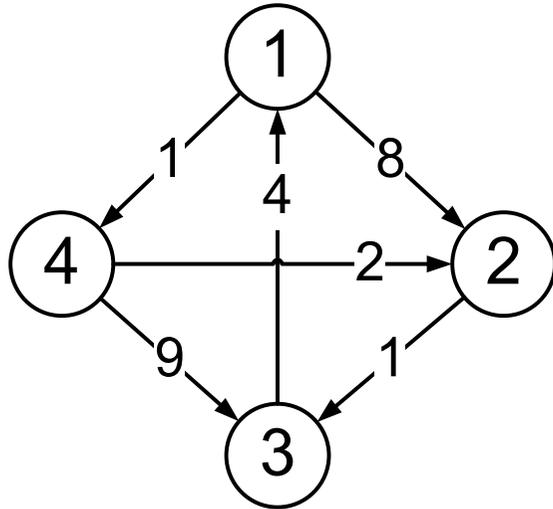
$$D^{(1)} = \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{pmatrix}$$

$$\Pi^{(1)} = \begin{pmatrix} nil & 1 & nil & 1 \\ nil & nil & 2 & nil \\ 3 & 1 & nil & 1 \\ nil & 4 & 4 & nil \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{pmatrix}$$

$$\Pi^{(2)} = \begin{pmatrix} nil & 1 & 2 & 1 \\ nil & nil & 2 & nil \\ 3 & 1 & nil & 1 \\ nil & 4 & 2 & nil \end{pmatrix}$$

Ejemplo algoritmo de Floyd Warshall



$$D^{(3)} = \begin{pmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix}$$

$$\Pi^{(3)} = \begin{pmatrix} nil & 1 & 2 & 1 \\ 3 & nil & 2 & 1 \\ 3 & 1 & nil & 1 \\ 3 & 4 & 2 & nil \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix}$$

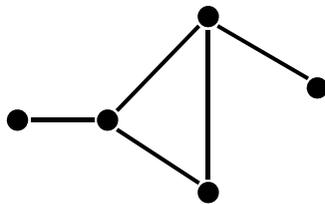
$$\Pi^{(4)} = \begin{pmatrix} nil & 4 & 2 & 1 \\ 3 & nil & 2 & 1 \\ 3 & 4 & nil & 1 \\ 3 & 4 & 2 & nil \end{pmatrix}$$

Contenidos

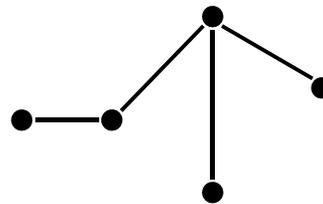
- Introducción
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

Minimum Spanning Tree

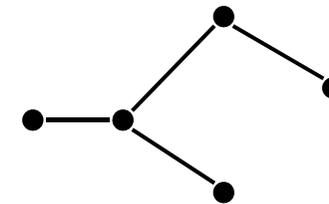
- **Tree** (árbol): es un grafo G , no dirigido, conectado y sin ciclos
 - El número de enlaces es igual al número de nodos menos 1: $|E| = |N| - 1$
 - Cualquier par de nodos están unidos por un único camino
 - G está conectado, pero al eliminar cualquier enlace dejaría de estarlo
 - G no tiene ciclos, pero al añadir un enlace cualquiera aparecería un ciclo



No es un árbol



Sí es un árbol



Sí es un árbol

- Un **Spanning Tree** cubre (se expande por) todos los nodos de un grafo Q
 - Entre los diferentes **Spanning Tree** de un grafo Q el **Minimum Spanning Tree (MST)** es aquel que tiene un menor coste

Minimum Spanning Tree

- Aplicaciones: el MST cubre todos los nodos de una red
 - Procesos de difusión (broadcast) de información
 - Un mensaje para ser enviado a todos los nodos de la red
 - Gran uso en Redes de Área Local: bridges (IEEE 802.1D)
 - Se emplea para eliminar enlaces no necesarios
- Algoritmos
 - *Kruksal*
 - *Prim*

MST: Algoritmo de Kruksal

- Construye el MST incorporando paulatinamente enlaces que unan dos componentes diferentes (dos subgrafos no conectados entre sí)
 - Se puede ver como un proceso de búsqueda de componentes conectados en una red
- Va recorriendo los enlaces (u,v) de \mathbb{E} en orden creciente (por su coste)
 - Añade el enlace actual (u,v) a un subgrafo \mathbb{A} si u y v pertenecen a árboles distintos
 - Siempre se añade aquel que tenga un menor coste
- Variables
 - \mathbb{A} : Conjunto de enlaces que forman el MST
 - L : lista con los enlaces de \mathbb{G} , ordenados según su coste, en orden creciente
- Al recorrer todos los enlaces \mathbb{A} contendrá el MST de \mathbb{G}

MST: Algoritmo de Kruksal

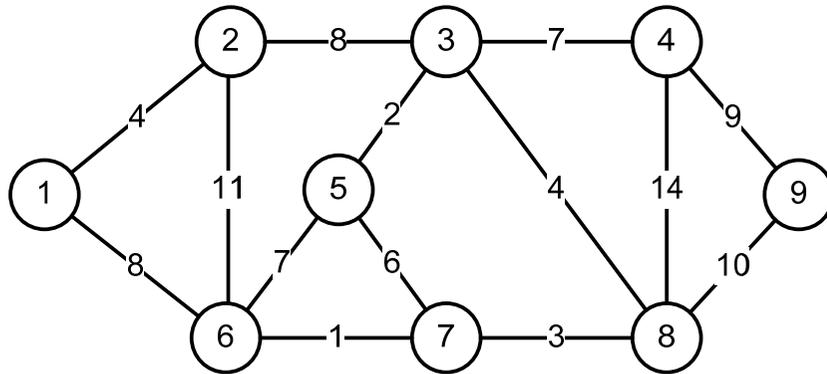
INITIALIZATION

1. $A = \{\emptyset\}$
2. $L = E$
3. `sort(L)`

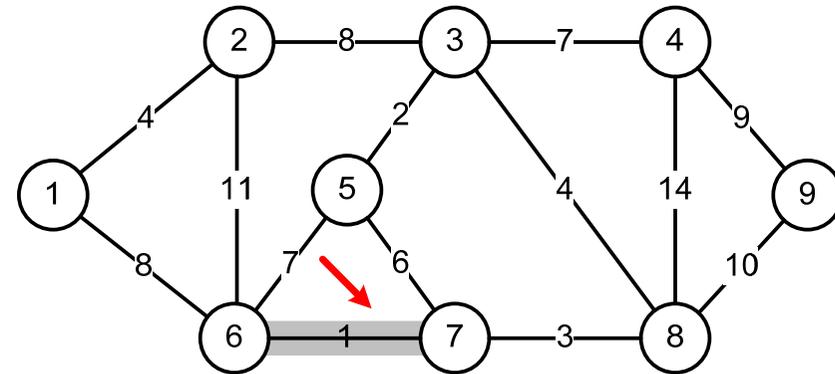
MAIN LOOP

4. for all (u,v) in L (in order)
5. if u & v belong to same tree
6. discard (u,v)
7. else
8. $A = A \cup (u,v)$

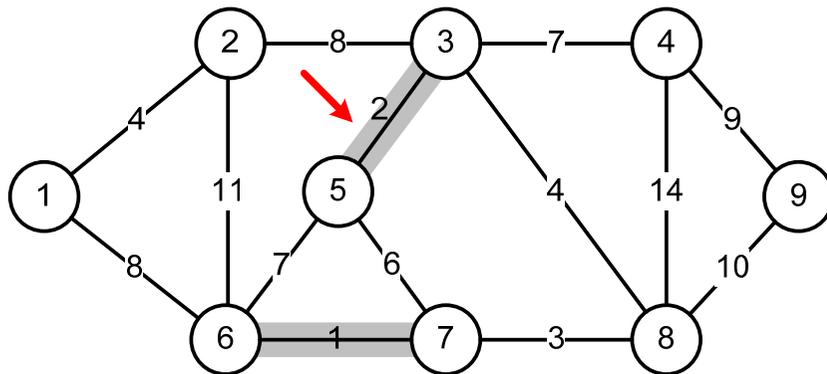
MST: Ejemplo algoritmo de Kruksal



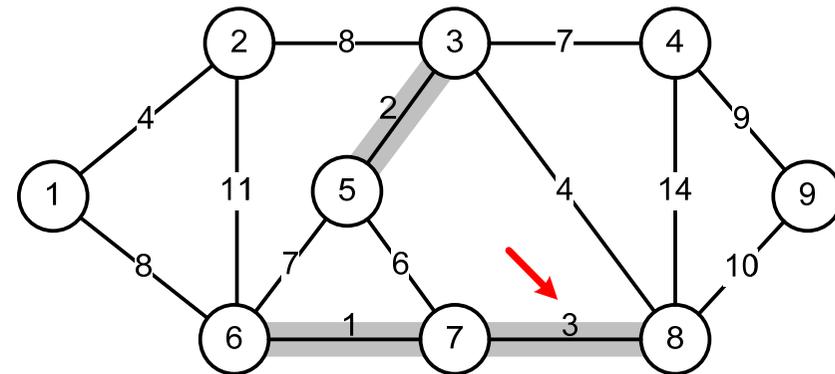
Grafo original



Iteración 1

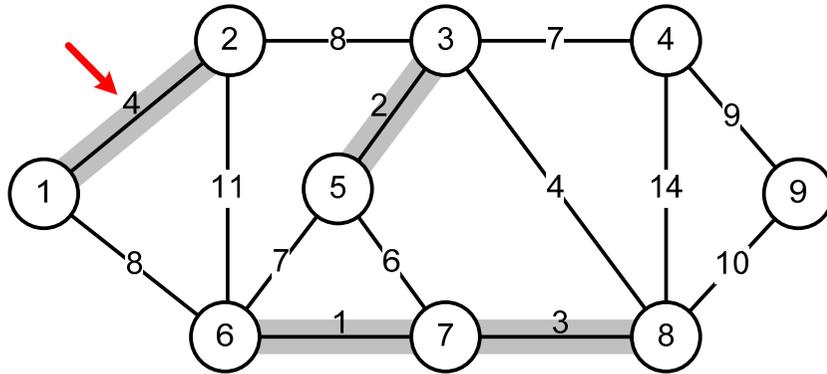


Iteración 2

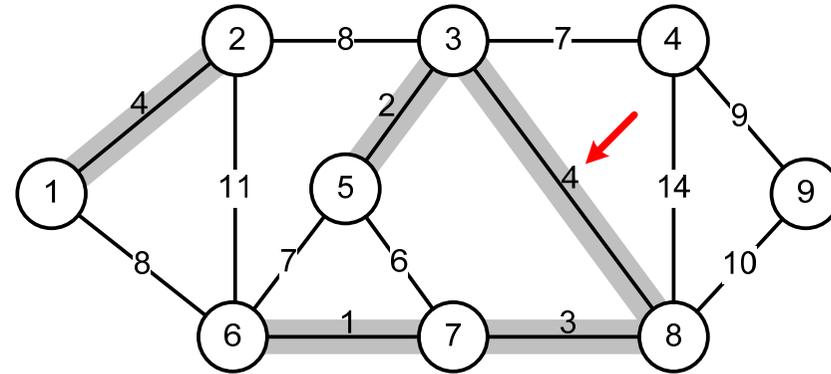


Iteración 3

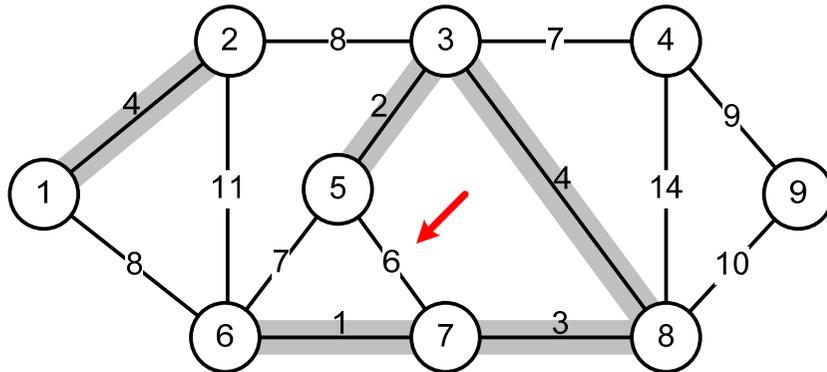
MST: Ejemplo algoritmo de Kruksal



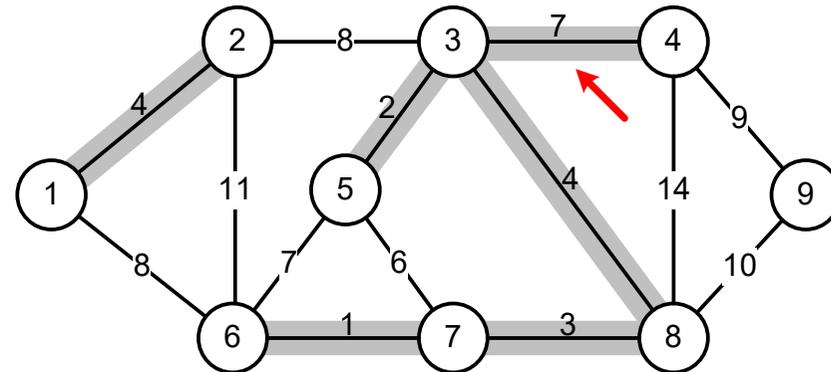
Iteración 4



Iteración 5

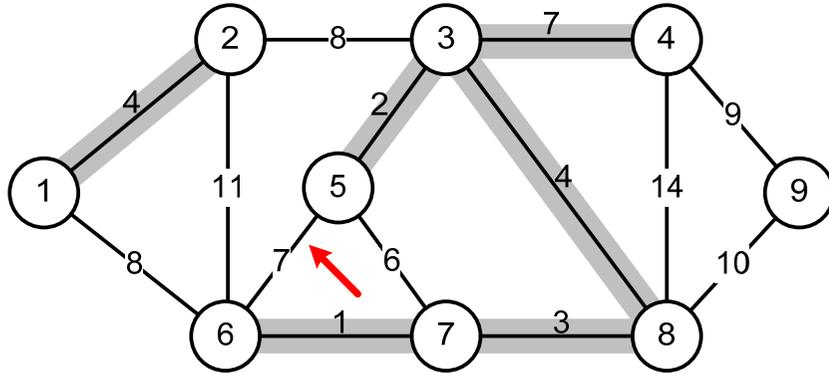


Iteración 6

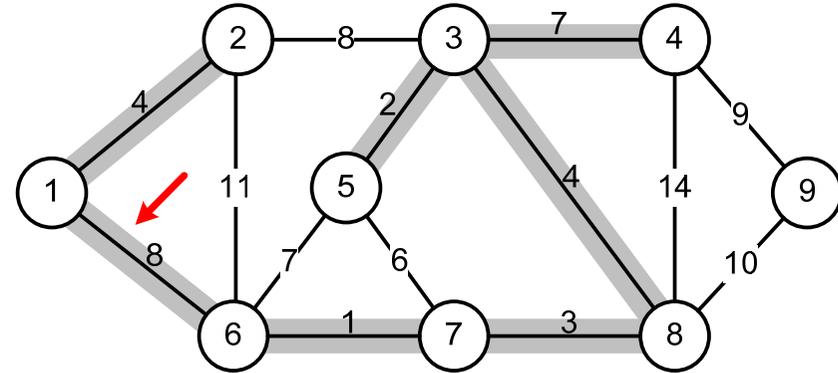


Iteración 7

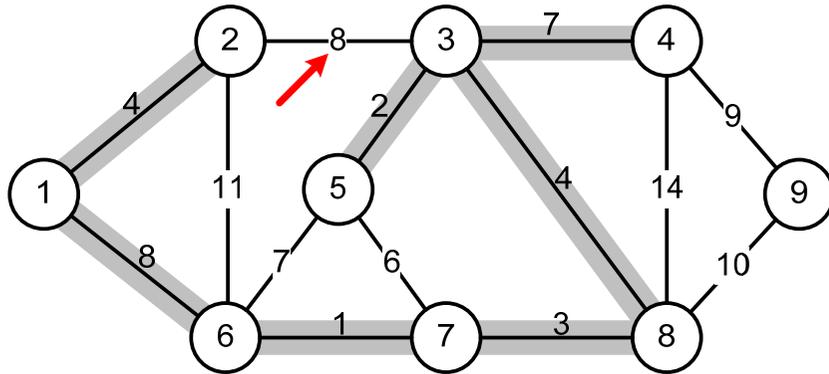
MST: Ejemplo algoritmo de Kruksal



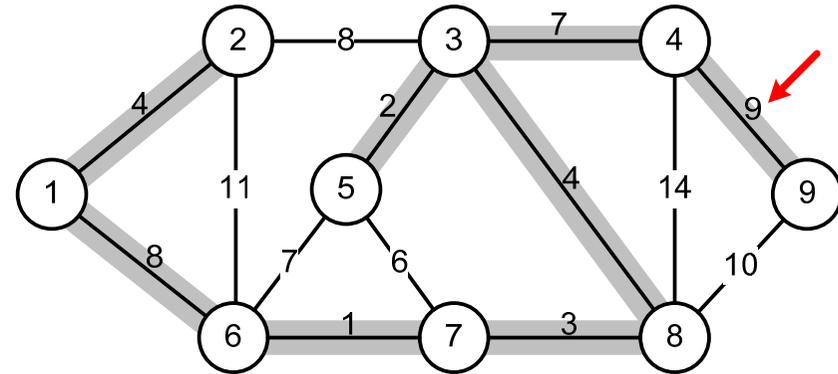
Iteración 8



Iteración 9

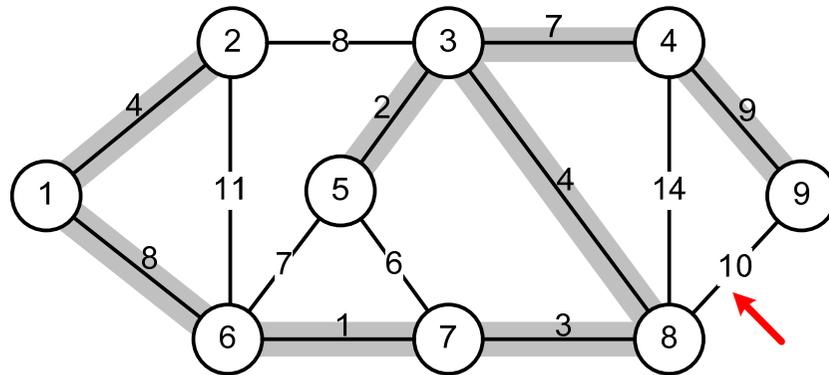


Iteración 10

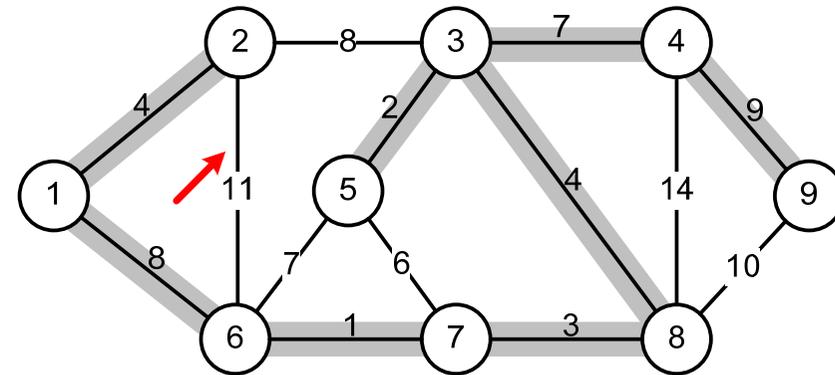


Iteración 11

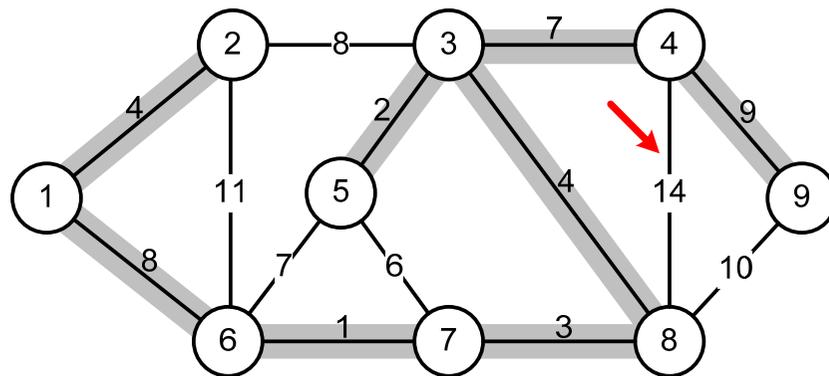
MST: Ejemplo algoritmo de Kruksal



Iteración 12



Iteración 13



Iteración 14

MST: Algoritmo de Prim

- Opera de manera similar al algoritmo de *Dijkstra*
- Comienza con un nodo arbitrario (R), al que paulatinamente se añaden enlaces, hasta que se cubren todos los nodos
- Variables
 - Conjunto de nodos Q que quedan por incorporar al MST
 - Se mantiene una lista con el 'peso' de cada nodo

MST: Algoritmo de Prim

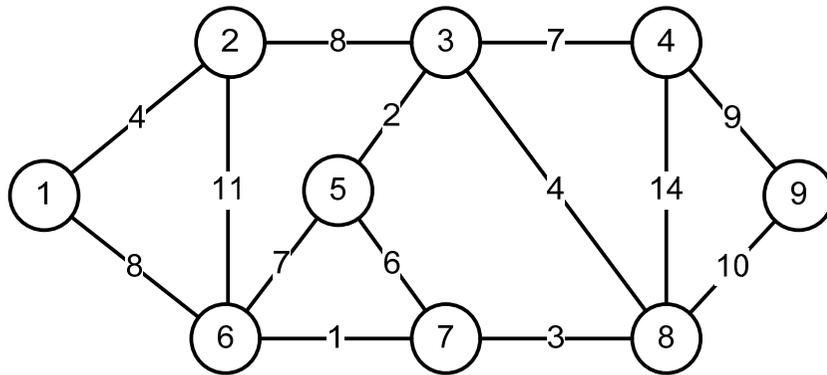
INITIALIZATION

1. $Q = N$; Randomly select R
2. $k(R) = 0$
3. for all v in N but R
4. $k(v) = \infty$

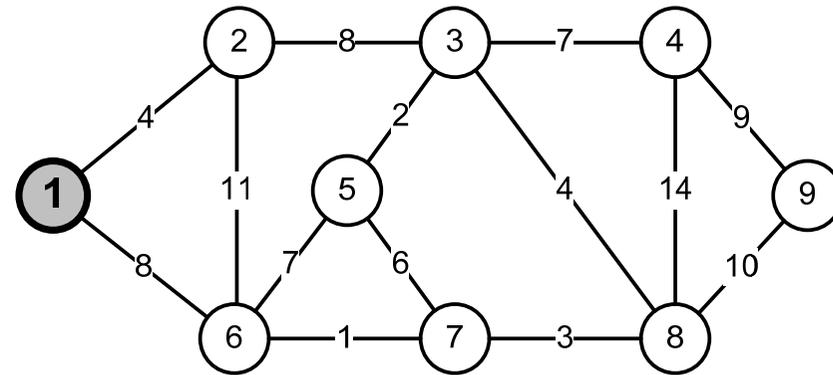
MAIN LOOP

5. while $Q \neq \{\emptyset\}$
6. u vertex in Q with $\min\{k(v)\}$
7. delete u from Q
8. for all v adjacent to u AND v in Q
9. if $k(v) > c(u, v)$
10. $k(v) = c(u, v)$
11. $\text{prev}(v) = u$

MST: Ejemplo algoritmo de Prim



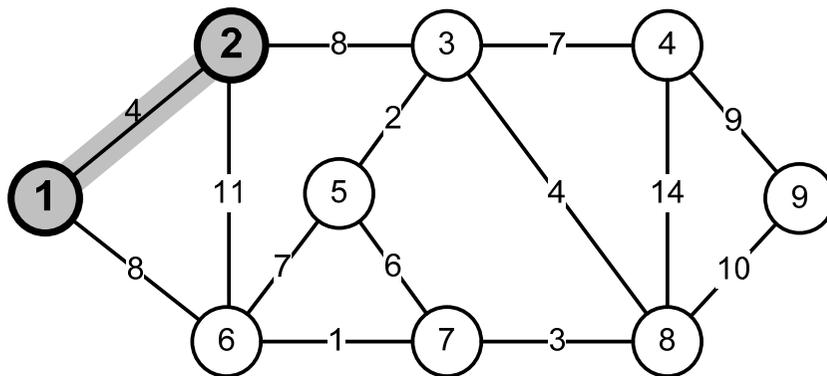
Grafo original



Iteración 1

$Q = \{2,3,4,5,6,7,8,9\}$

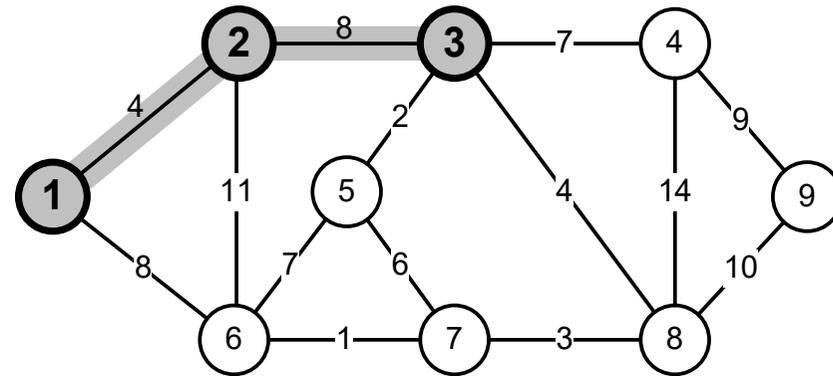
$k = \{0,4,\infty,\infty,\infty,8,\infty,\infty,\infty\}$



Iteración 2

$Q = \{3,4,5,6,7,8,9\}$

$k = \{0, 4,8,\infty,\infty,8,\infty,\infty,\infty\}$

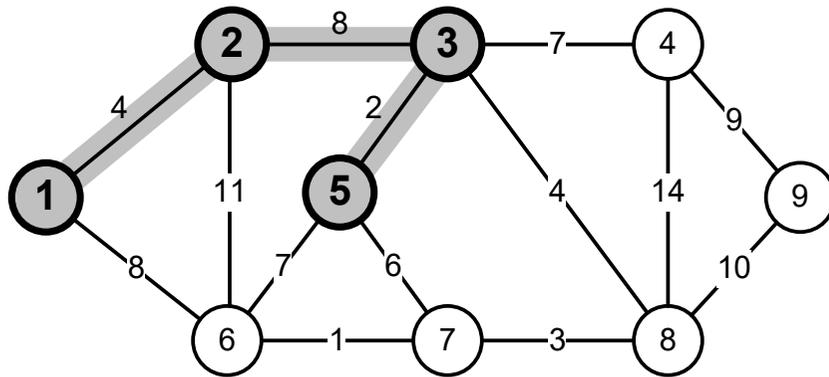


Iteración 3

$Q = \{4,5,6,7,8,9\}$

$k = \{0, 4,8,7,2,8,\infty,4,\infty\}$

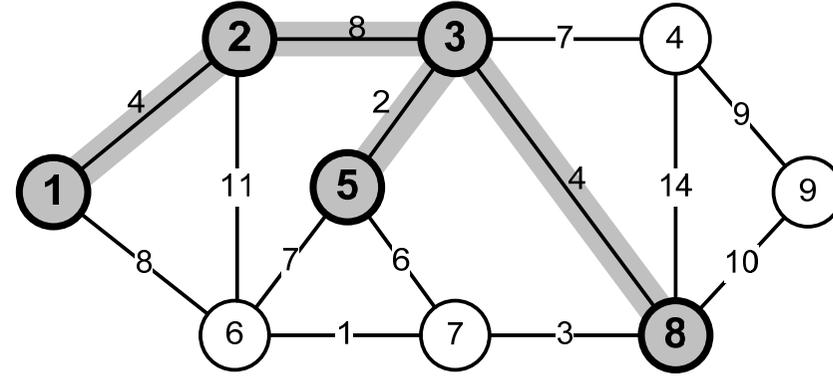
MST: Ejemplo algoritmo de Prim



Iteración 4

$Q = \{4,6,7,8,9\}$

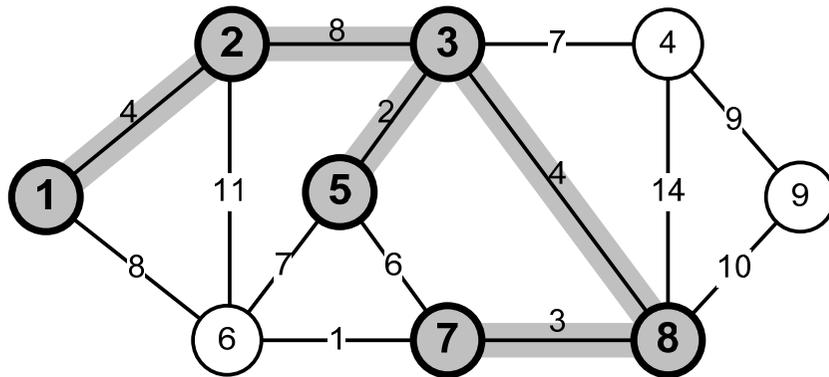
$k = \{0,4,8,7,2,7,6,4,\infty\}$



Iteración 5

$Q = \{4,6,7,9\}$

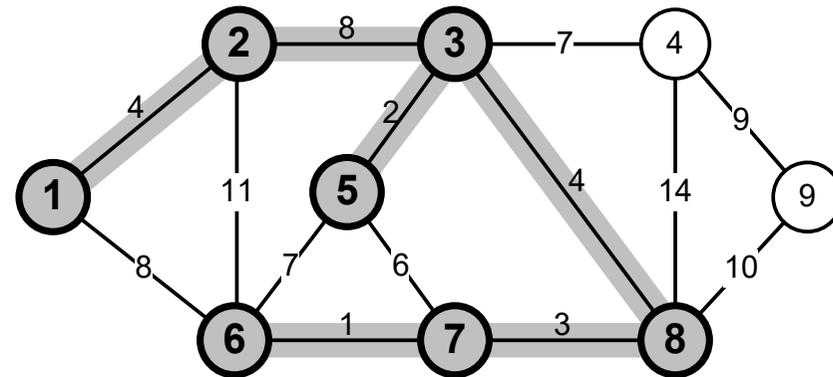
$k = \{0,4,8,7,2,7,3,4,10\}$



Iteración 6

$Q = \{4,6,9\}$

$k = \{0,4,8,7,2,1,3,4,10\}$

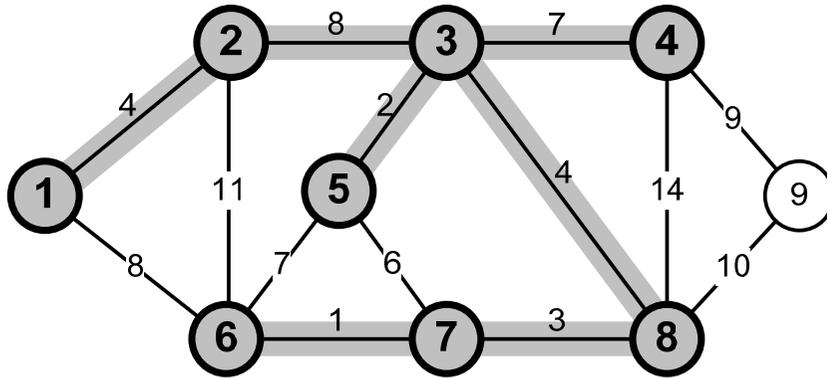


Iteración 7

$Q = \{4,9\}$

$k = \{0,4,8,7,2,1,3,4,10\}$

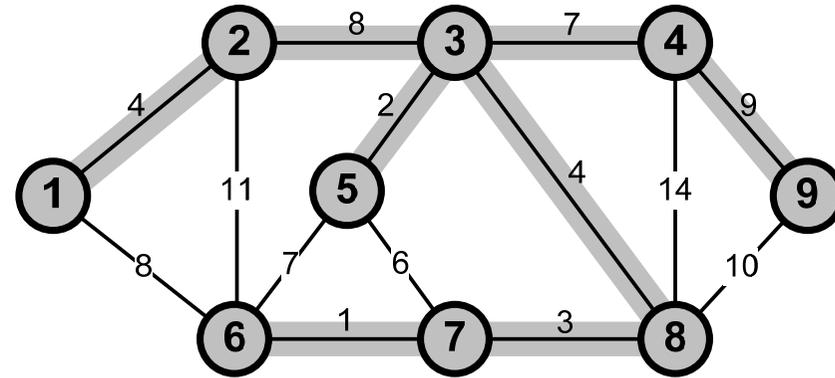
MST: Ejemplo algoritmo de Prim



Iteración 8

$Q = \{9\}$

$k = \{0,4,8,7,2,1,3,4,9\}$



Iteración 9

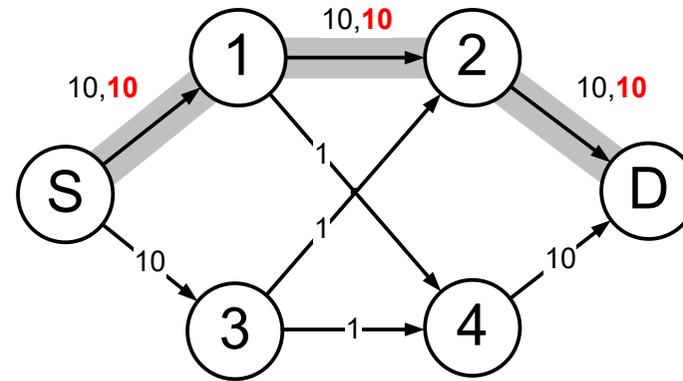
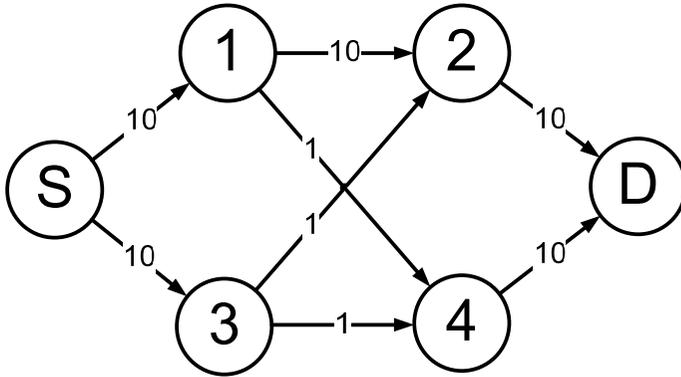
$Q = \{\emptyset\}$

$k = \{0,4,8,7,2,1,3,4,9\}$

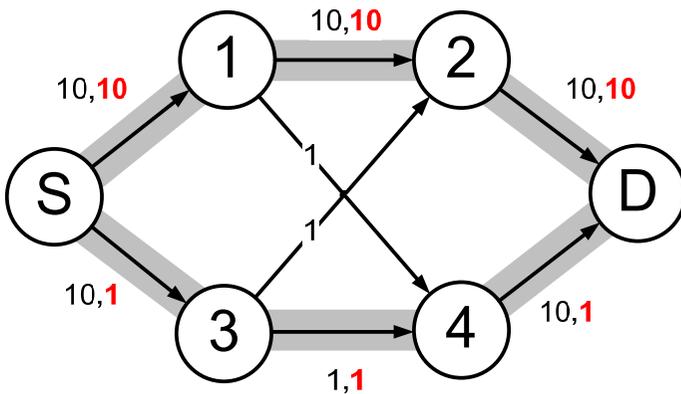
Problema de máximo flujo

- Se trata de maximizar el flujo que se puede “enviar” entre S y D
- Cada enlace tiene una capacidad $p(u,v)$ NO NEGATIVA
- Por un enlace (u,v) se tiene un flujo $f(u,v) \rightarrow f(u,v) \leq p(u,v)$
- Conservación de flujo $\sum_{u,v \in N - \{S,D\}} f(u,v) = 0$
- Simetría $f(u,v) = -f(v,u)$

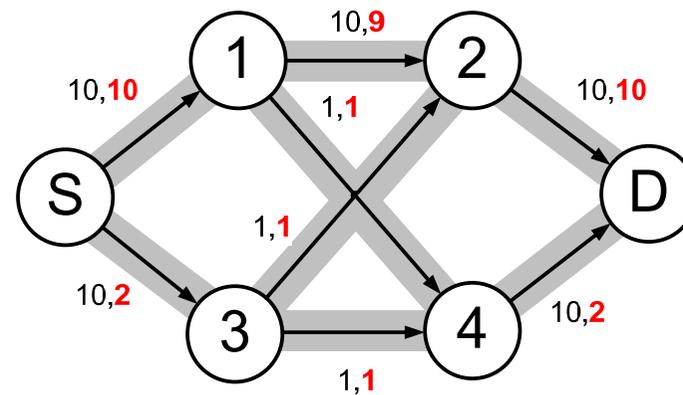
Problema de máximo flujo



Flujo = 10



Flujo = 11



Flujo = 12

(No es obvio)

Máximo flujo: conceptos previos

- Red residual
 - Enlaces que pueden admitir más flujo
- Capacidad residual
 - La capacidad que queda disponible en un enlace $p_f(u, v) = p(u, v) - f(u, v)$
- Augmenting Path
 - Es un camino entre S y D en el grafo con capacidades residuales
 - La red residual puede tener enlaces nuevos
 - Su capacidad residual es la menor de sus enlaces

Máximo flujo: Algoritmo Ford-Fulkerson

- El algoritmo Ford-Fulkerson resuelve el problema del máximo flujo
- En cada iteración
 - Encuentra un **augmenting path** \mathbb{P}
 - Incrementa el flujo entre S y D con la capacidad residual de \mathbb{P}

Máximo flujo: Algoritmo Ford-Fulkerson

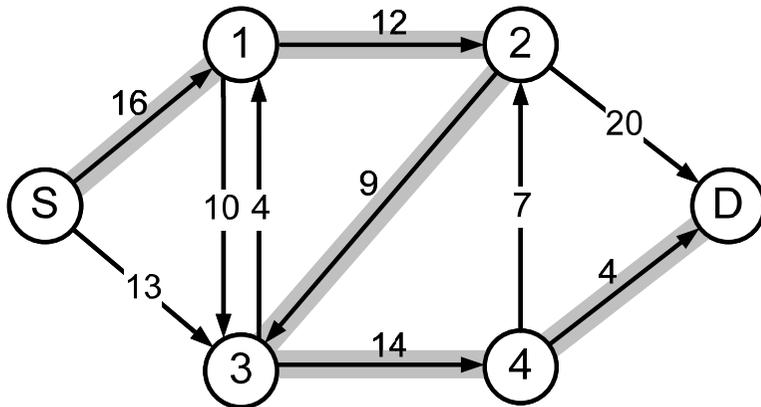
INITIALIZATION

1. for all (u, v) in E
2. $f(u, v) = 0$
3. $f(v, u) = 0$

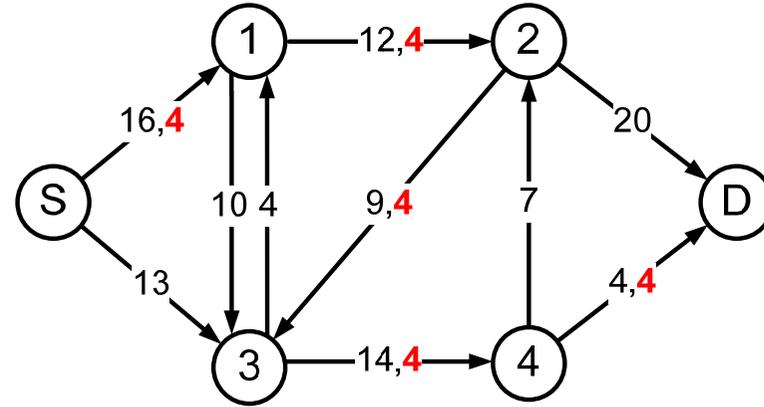
MAIN LOOP

4. while there exists a path p from S
to D in the residual network G_f
5. $cf(p) = \min\{cf(u, v) : (u, v) \text{ in } p\}$
6. for each (u, v) in p
7. $f(u, v) = f(u, v) + cf(p)$
8. $f(v, u) = -f(u, v)$

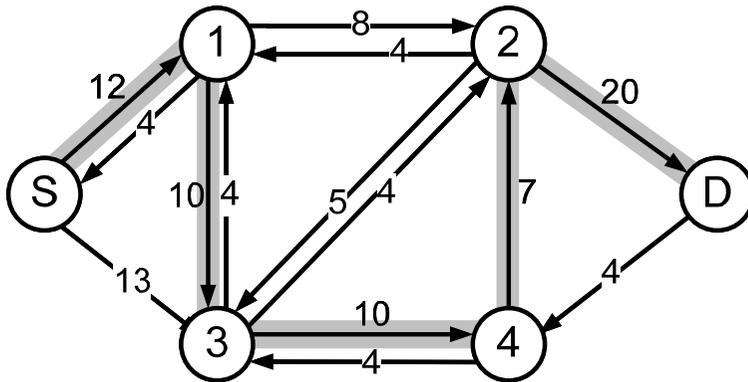
Ejemplo algoritmo Ford-Fulkerson



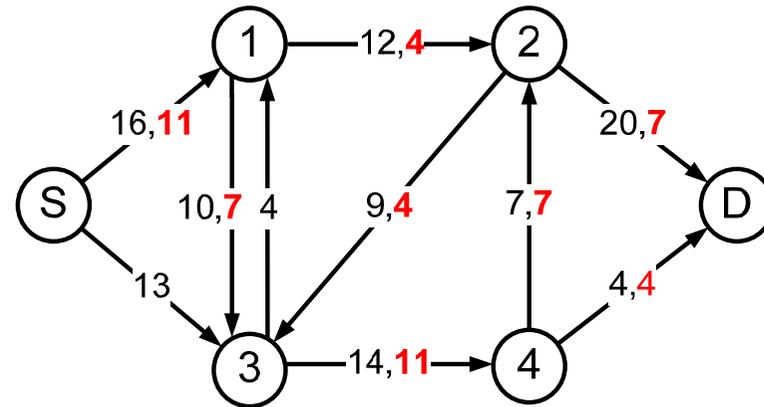
$P = \{(S,1) (1,2) (2,3) (3,4) (4,D)\}$
 $cf(P) = 4$



Flujo = 4

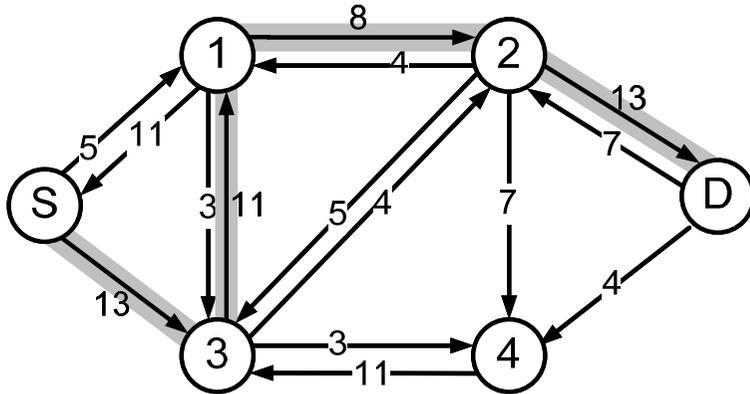


$P = \{(S,1) (1,3) (3,4) (4,2) (2,D)\}$
 $cf(P) = 7$

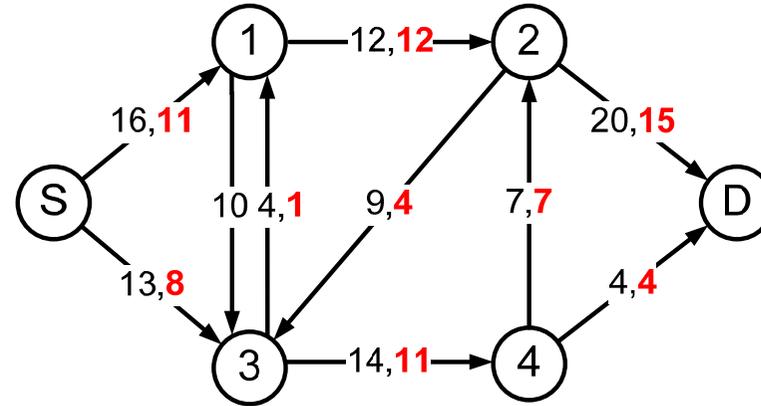


Flujo = 11

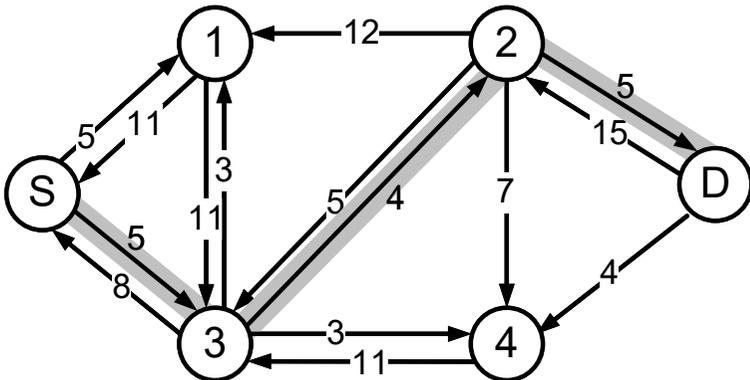
Ejemplo algoritmo Ford-Fulkerson



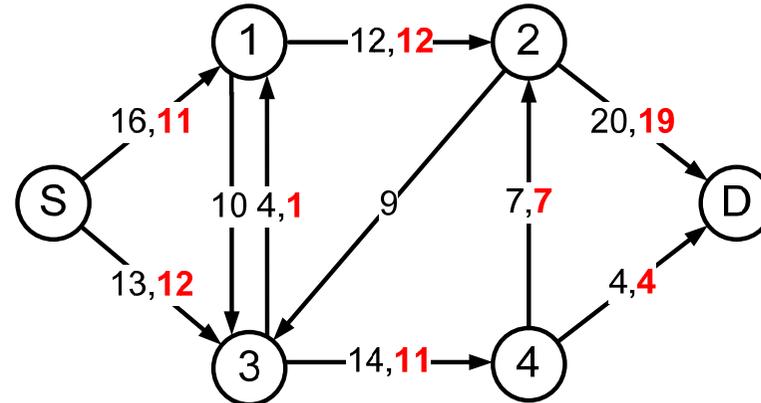
$P = \{(S,3) (3,1) (1,2) (2,D)\}$
 $cf(P) = 8$



Flujo = 19

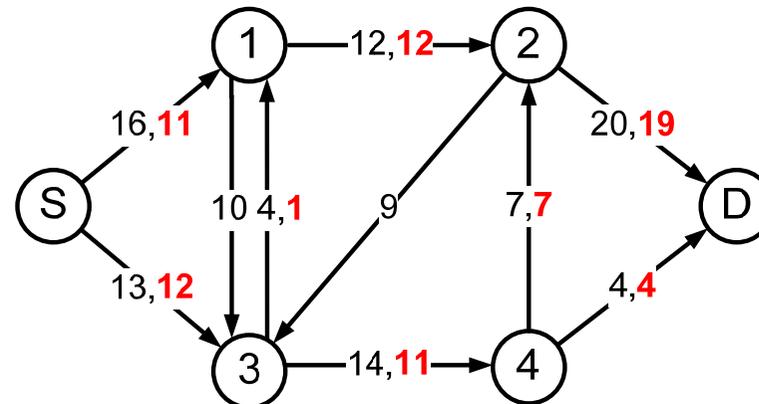
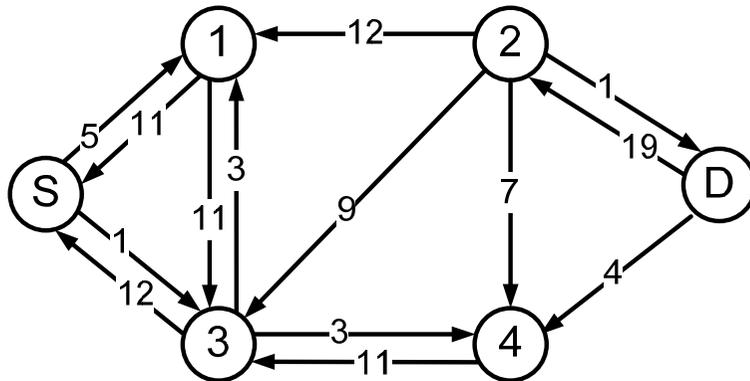


$P = \{(S,3) (3,2) (2,D)\}$
 $cf(P) = 4$



Flujo = 23

Ejemplo algoritmo Ford-Fulkerson



Flujo = 23

- Como ya no existe un “*augmenting path*” en la red residual el algoritmo se da por finalizado
- La eficiencia del algoritmo depende de la manera en la que se “busca” el *augmenting path* en cada iteración
 - Podría incluso no converger

Contenidos

- Introducción
- Teoría de grafos
- Algoritmos de búsqueda de camino más corto
- Otros algoritmos en grafos
- Del algoritmo al protocolo

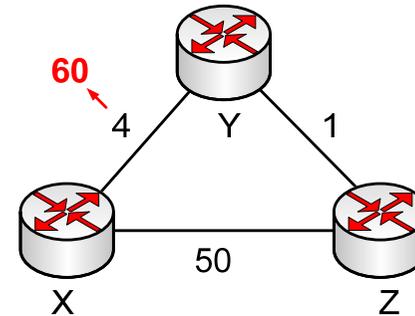
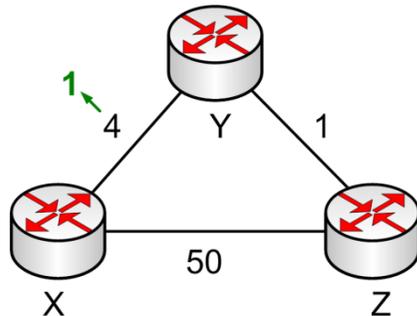
Encaminamiento en redes

- Los nodos (*routers*) toman decisiones en base a cierta información
- Es necesario disponer de un protocolo de señalización
 - Proporciona un método para transportar dicha información por la red
- Protocolos de encaminamiento
 - Vector distancia (**Distance vector, DV**) – usado en la *1ª generación ARPANET*
 - Estado del enlace (**Link state, LS**) – usado en la *2ª generación ARPANET*
- Jerarquía en la red
 - Escalabilidad
 - Autonomía administración de la red
 - Sistemas Autónomos (*Autonomous Systems, AS*)
 - Encaminamiento
 - Intra-AS: dentro de un AS
 - Inter-AS: entre varios AS

Encaminamiento vector distancia

- Se basa en el algoritmo de Bellman-Ford
- Cada nodo de la red mantiene una tabla: una entrada por cada uno del resto de nodos
 - Distancia: “métrica o coste” del camino hacia dicho nodo
 - Interfaz de salida necesaria para alcanzarle
- Periódicamente cada nodo intercambia la información de la tabla con sus vecinos
- Problemas
 - Convergencia lenta
 - Propagación rápida de “buenas noticias” y lenta de las “malas”
 - Count-to-infinity
- Protocolo RIP (*Routing Information Protocol*)
 - Definido en los RFC 1058 y 2453 (RIP Version 2)
 - Es un protocolo Intra-AS que, a pesar de que se sigue empleando, es obsoleto

DV: Count-to-infinity



- El enlace XY cambia a 1
 - [t0] Y detecta el cambio
 - [t1] Z modifica su tabla
 - [t2] Y recibe la actualización de Z, pero no necesita hacer ningún cambio

	$D_Y(X)$	$D_Z(X)$
t0	1	5
t1	1	2
t2	1	2

- El enlace XY cambia a 60
 - [t0] Y detecta el cambio, pero actualiza a 6 (cree que puede ir a través de Z)
 - [t1] Z modifica su tabla a 7 (6+1)
 - [t2] Y recibe la actualización de Z, y cambia su información a 8
 - ...
 - Son necesarias **44 iteraciones**

	$D_Y(X)$	$D_Z(X)$
t0	6	5
t1	6	7
t2	8	7
t3	8	9

Encaminamiento estado de enlace

- Sustituye paulatinamente a DV a partir de 1980
- Se pueden establecer cinco elementos diferenciados
 - Descubrimiento de vecinos: uso de paquetes **HELLO**
 - Estimación del coste con los vecinos
 - Por ejemplo, en base al retardo con cada uno de ellos: paquetes **ECHO**
 - Construcción de un paquete con la información correspondiente
 - ¿Cuándo se tiene que construir dicho paquete?
 - Envío del paquete al resto de nodos (*routers*)
 - Difusión [*broadcast*] de información: inundación
 - Cálculo de la ruta
 - Uso del algoritmo de **Dijkstra** – necesidad de disponer de información global

Encaminamiento estado de enlace

- Desventajas
 - Necesidad de información global
 - Envío de un número mayor de mensajes: SOBRECARGA
 - Un evento de “cambio” en la topología de la red se tiene que notificar a todos los nodos
- Protocolo OSPF (Open Shortest Path First)
 - Está definido (en su segunda versión) en el RFC 2328
 - Es un protocolo Intra-AS

Encaminamiento jerárquico

- Los sistemas autónomos se comunican entre ellos a través de nodos denominados *Gateway*
 - Un AS puede tener más de un Gateway
- Actualmente el encaminamiento entre AS es soportado por el protocolo BGP (Border Gateway Protocol)
 - Está definido (en su versión 4) en el RFC 4271
 - Es el protocolo de encaminamiento en el *core* de Internet, y es utilizado por los proveedores de servicios (ISP)
- Funcionamiento básico de BGP
 - Se basa en un encaminamiento basado en el estado del camino (*path state*)
 - Cada Gateway obtiene la “alcanzabilidad” de los AS vecinos
 - Propaga dicha información a los *routers* de su AS
 - Determina las rutas “óptimas” en función de la información adquirida y de las políticas y reglas establecidas