Camino mínimo en grafos

Melanie Sclar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

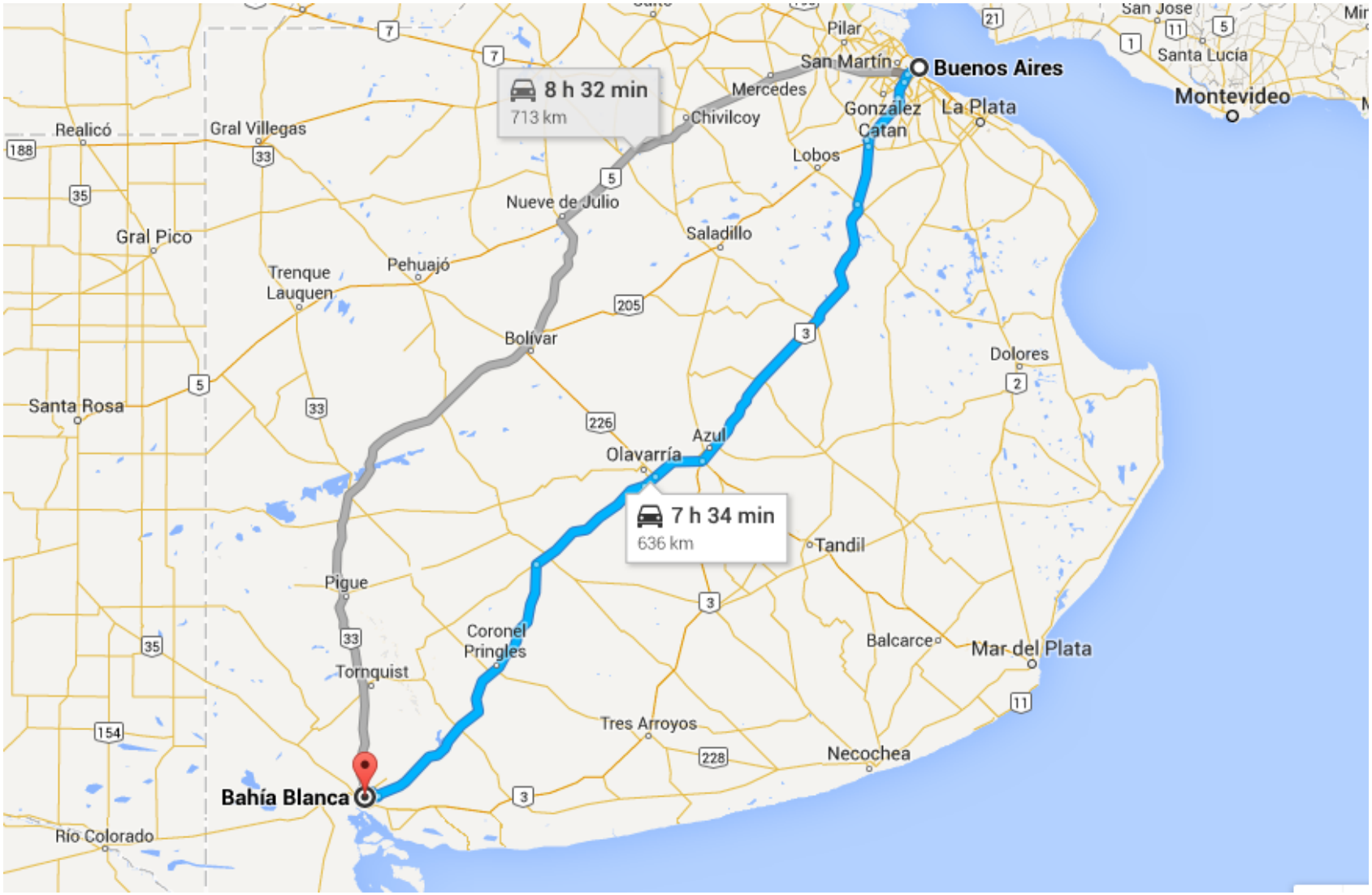
Nacional OIA 2016

Contenidos

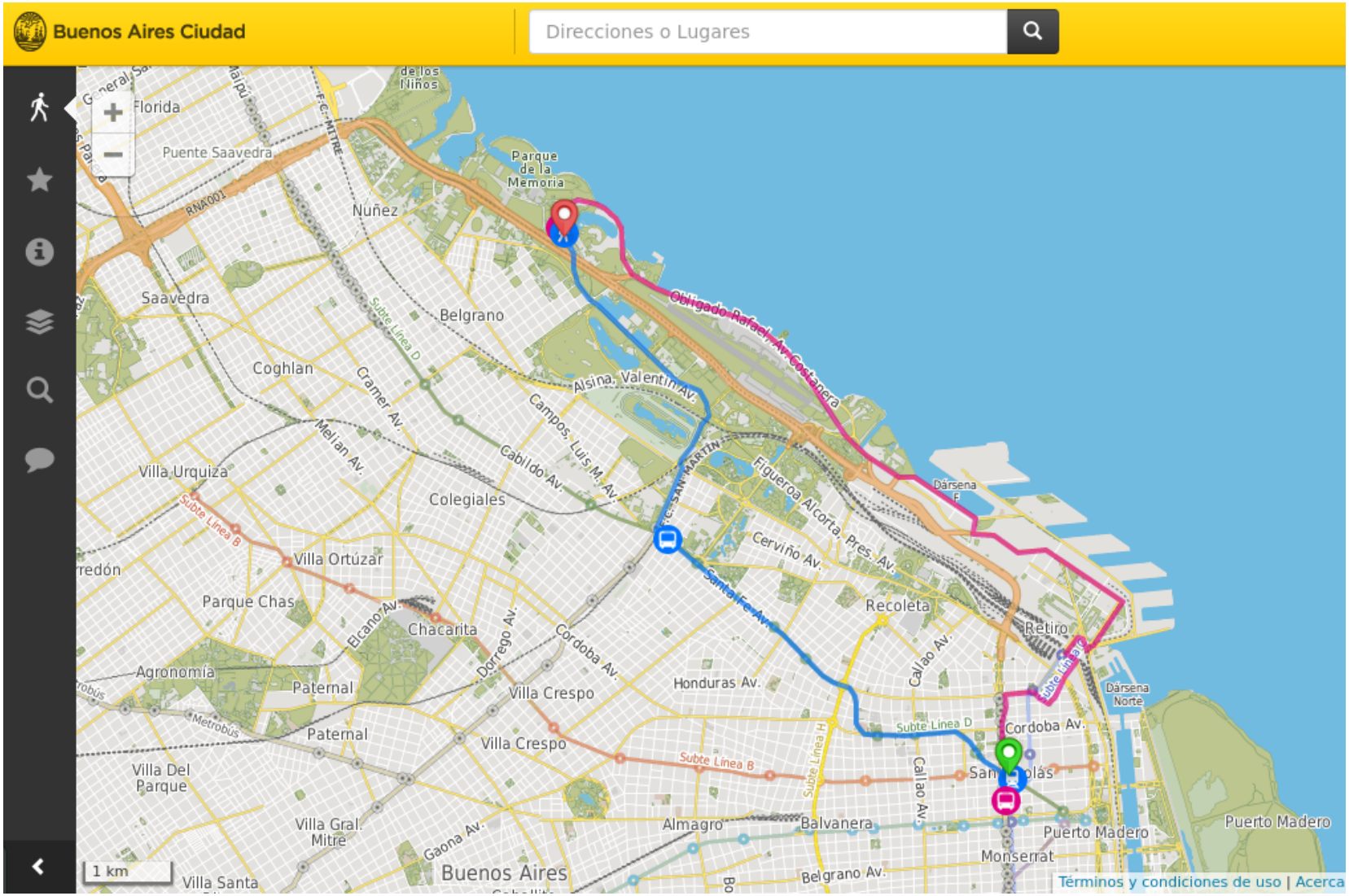
Esta charla tratará sobre uno de los problemas más renombrados de la computación: **el problema del camino mínimo**. Este problema consiste en hallar la mejor forma de ir desde un punto a otro (o a varios otros) minimizando la distancia recorrida, el tiempo invertido, entre varias posibilidades.

Es un problema muy útil en las olimpíadas, pero no solamente aquí. Veamos dos ejemplos de la vida real que deben resolver el problema que trataremos hoy.

Camino mínimo



Camino mínimo



Representando el problema

Para poder resolver estos problemas debemos poder representarlos de manera concisa y abstracta, desprendiéndonos de las particularidades de cada caso de uso posible.

La representación más usual de este tipo de problemas (ya sea para aplicarlo en problemas de camino mínimo o no) es la de **grafos**.

Contenidos

¿Qué es un grafo?

> "Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en inglés) que pueden ser orientados (dirigidos) o no."
>
> — Wikipedia

¿Qué es un grafo?

"Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en inglés) que pueden ser orientados (dirigidos) o no."

— Wikipedia

"Un grafo es un conjunto de puntos y líneas que unen pares de esos puntos"

— La Posta

Grafos dirigidos y no dirigidos

En los grafos no dirigidos las aristas son doble mano (se puede ir en ambos sentidos).

En los dirigidos en cambio, las aristas se recorren en un único sentido: desde el origen al destino de la flecha. Si queremos representar que la calle que une las esquinas u y v es doble mano deberemos poner dos aristas: una que vaya de u a v y otra que vaya de v a u.

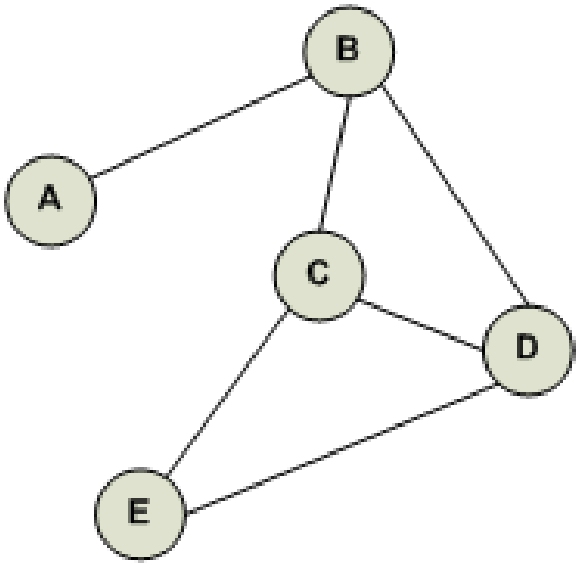


Fig 1. Grafo no dirigido

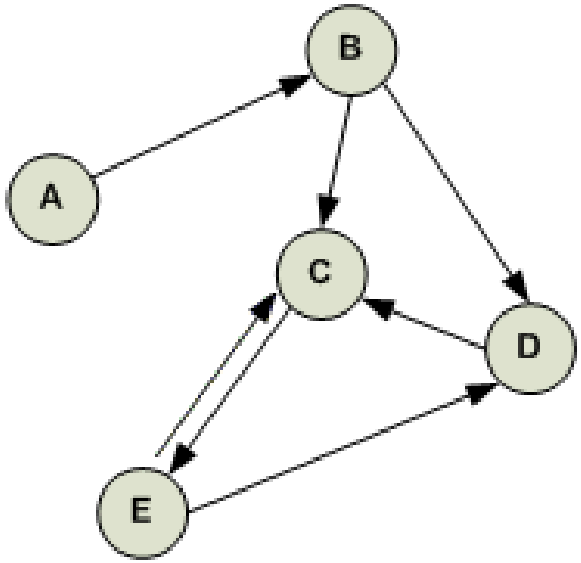


Fig 2. Grafo dirigido

¿Para qué podemos usar los grafos?

Mediante un grafo podemos representar, por ejemplo, una ciudad. Las esquinas serían los vértices y las conexiones por medio de una calle entre dos esquinas serían los ejes. Si todas las calles son doble mano, el grafo es no dirigido. Si algunas calles son mano única el grafo es dirigido: ¿cómo modelamos una calle doble mano aquí?

A medida que los problemas se dificultan, puede suceder que sea difícil que a uno se le ocurra modelar el problema con un grafo, pero que una vez que lo hayamos hecho, el problema se vuelva sencillo utilizando los algoritmos que veremos hoy.

Formas de representar un grafo

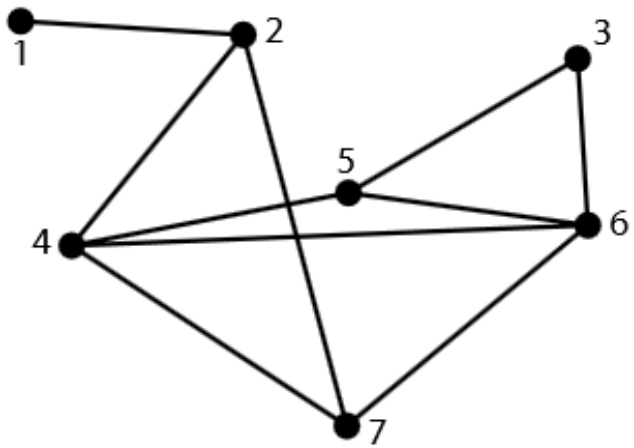
Existen varias maneras de guardar un grafo en memoria para poder luego consultar cosas (por ejemplo, recorrer el grafo, que es lo que haremos hoy).
Veamos las dos más populares.

Matriz de adyacencia

Matriz de adyacencia

La matriz de adyacencia es una matriz de $n \times n$ donde n es la cantidad de nodos del grafo, que en la posición (i, j) tiene un 1 (o true) si hay una arista entre los nodos i y j y 0 (o false) si no.

Ej:



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

Matriz de adyacencia

Esta es una de las representaciones más utilizadas. Si bien el ejemplo es para un grafo no dirigido, también se puede utilizar la misma estructura para grafos dirigidos y grafos con pesos.

Ventajas

Matriz de adyacencia

Esta es una de las representaciones más utilizadas. Si bien el ejemplo es para un grafo no dirigido, también se puede utilizar la misma estructura para grafos dirigidos y grafos con pesos.

Ventajas

- Permite saber si existe o no arista entre dos nodos cualesquiera en O(1).
- Es muy fácil de implementar, $matrizAdy[i][j]$ guarda toda la información sobre la arista.

Matriz de adyacencia

Esta es una de las representaciones más utilizadas. Si bien el ejemplo es para un grafo no dirigido, también se puede utilizar la misma estructura para grafos dirigidos y grafos con pesos.

Ventajas

- Permite saber si existe o no arista entre dos nodos cualesquiera en O(1).
- Es muy fácil de implementar, $matrizAdy[i][j]$ guarda toda la información sobre la arista.

Desventajas

Matriz de adyacencia

Esta es una de las representaciones más utilizadas. Si bien el ejemplo es para un grafo no dirigido, también se puede utilizar la misma estructura para grafos dirigidos y grafos con pesos.

Ventajas

- Permite saber si existe o no arista entre dos nodos cualesquiera en O(1).
- Es muy fácil de implementar, $matrizAdy[i][j]$ guarda toda la información sobre la arista.

Desventajas

- La complejidad espacial: se necesitan n^2 casillas para representar un grafo de n nodos.

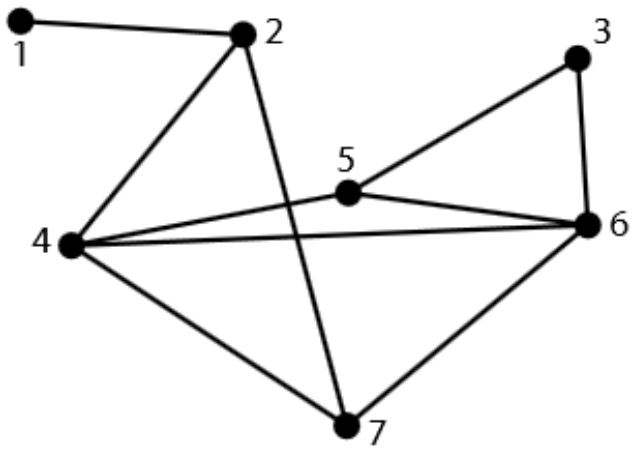
Lista de adyacencia

Lista de adyacencia

La lista de adyacencia es un vector de vectores de enteros, que en el *i*-ésimo vector tiene el número *j* si hay una arista entre los nodos *i* y *j*.

Coloquialmente la llamamos *lista de vecinos* pues para cada nodo guardamos la lista de nodos para los que existe una arista que los conecta (o sea, los vecinos).

Ej:



$$L_1 : 2$$
$$L_2 : 1 \rightarrow 4 \rightarrow 7$$
$$L_3 : 5 \rightarrow 6$$
$$L_4 : 2 \rightarrow 5 \rightarrow 6 \rightarrow 7$$
$$L_5 : 3 \rightarrow 4 \rightarrow 6$$
$$L_6 : 3 \rightarrow 4 \rightarrow 5 \rightarrow 7$$
$$L_7 : 2 \rightarrow 4 \rightarrow 6$$

Lista de adyacencia

Nuevamente, con la misma idea también se pueden modelar grafos dirigidos y con pesos.

La complejidad espacial de esta representación será posiblemente mucho menor. ¿Cuánta memoria necesitaremos para un grafo de n nodos y m aristas?

Lista de adyacencia

Nuevamente, con la misma idea también se pueden modelar grafos dirigidos y con pesos.

La complejidad espacial de esta representación será posiblemente mucho menor. ¿Cuánta memoria necesitaremos para un grafo de n nodos y m aristas? **O(m+n)**

Formalización del problema de camino mínimo

Precisemos mejor nuestro problema, usando los términos de grafos que acabamos de aprender.

Problema de Camino mínimo

Dado un grafo G con pesos en las aristas, el problema de camino mínimo entre dos nodos u y v consiste en encontrar un camino entre esos nodos cuyo peso sea menor o igual que el peso de cualquier otro camino entre u y v.

Según qué tipo de grafo analicemos, la solución al problema será diferente. Por ejemplo, si el grafo no tuviera pesos (o si todos los pesos fueran iguales, que a efectos prácticos es lo mismo), nos conviene usar otro algoritmo (BFS).

Contenidos

Algoritmo de Dijkstra

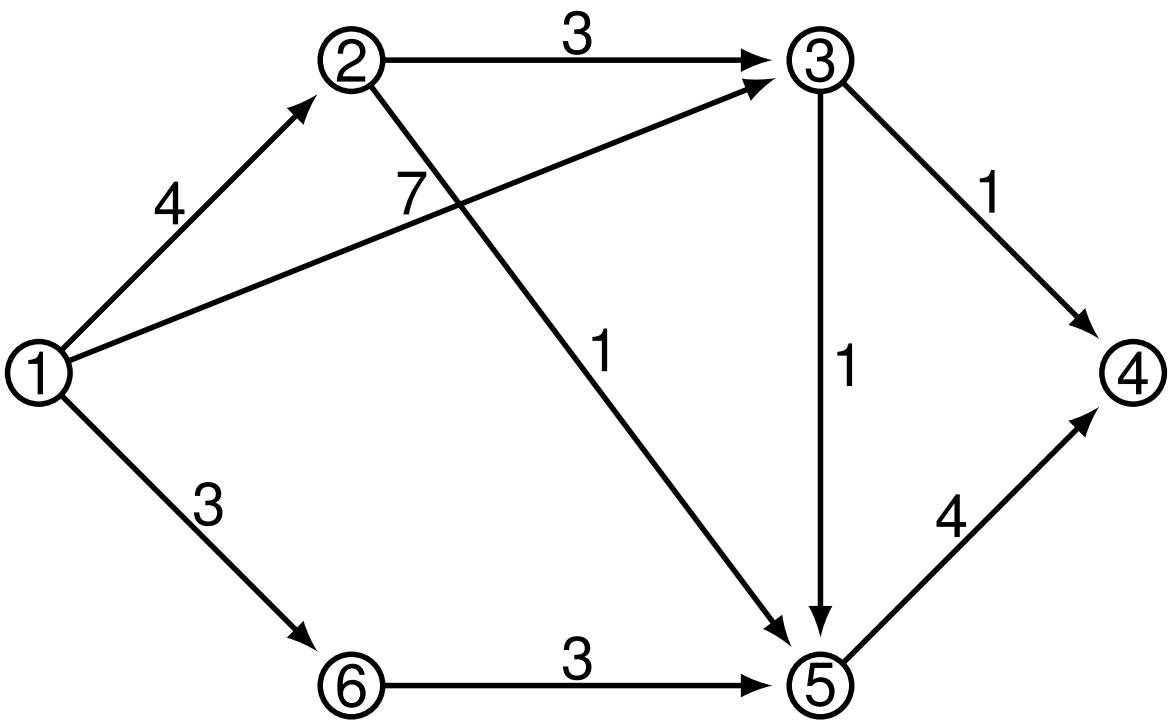
Este algoritmo fue creado por uno de los padres de la computación, Edger W. Dijkstra, en 1956. Sirve para cualquier grafo con pesos (dirigido o no) **siempre y cuando sus pesos no sean negativos**.

Idea del algoritmo de Dijkstra

- El algoritmo calcula las distancias mínimas desde un nodo inicial a todos los demás. Para hacerlo, en cada paso se toma el nodo más cercano al inicial que aún no fue visitado (le diremos v). Este nodo tiene calculada la menor distancia al nodo inicial (¿por qué?).
- Luego, recalculamos todas los caminos mínimos, teniendo en cuenta a v como camino intermedio.
- Así, en cada paso tendremos un subconjunto de nodos que ya tienen calculada su mínima distancia y los demás tienen calculada su mínima distancia si solo puedo usar los nodos del conjunto como nodos intermedios.
- Con cada iteración agregaremos un nodo más a nuestro conjunto, hasta resolver el problema en su totalidad.

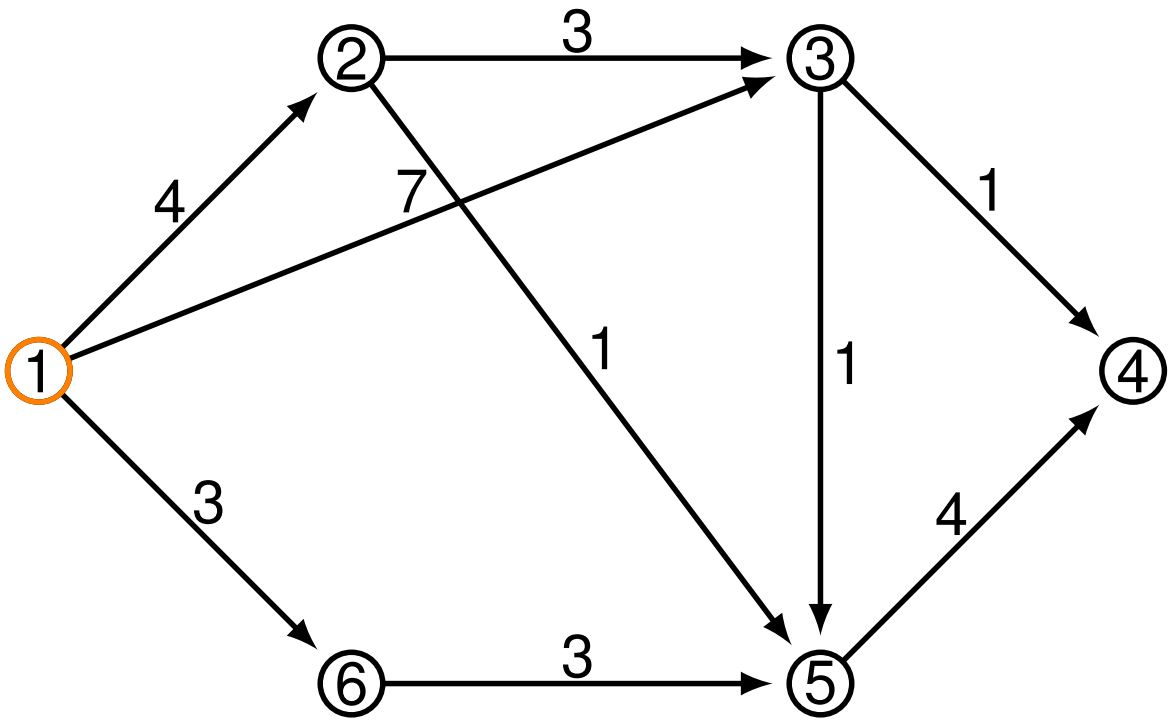
Veamos un ejemplo.

Algoritmo de Dijkstra - Ejemplo



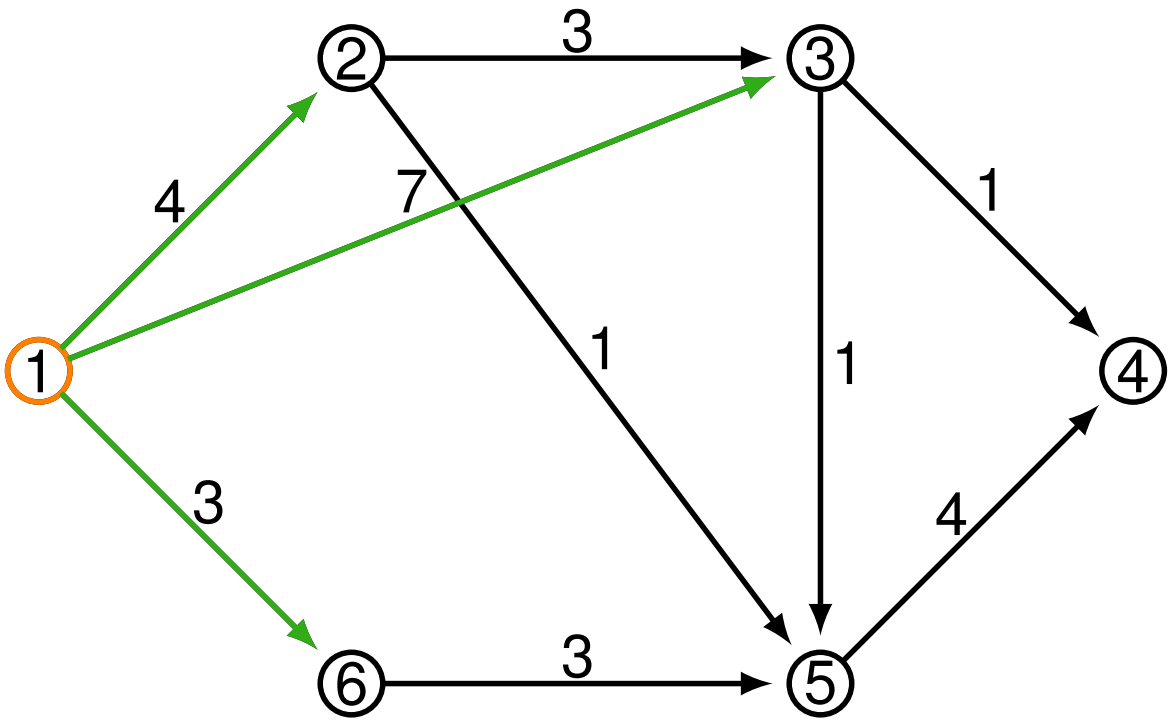
Algoritmo de Dijkstra - Ejemplo

$S = \{1\}$ $\pi = (0, \infty, \infty, \infty, \infty, \infty)$



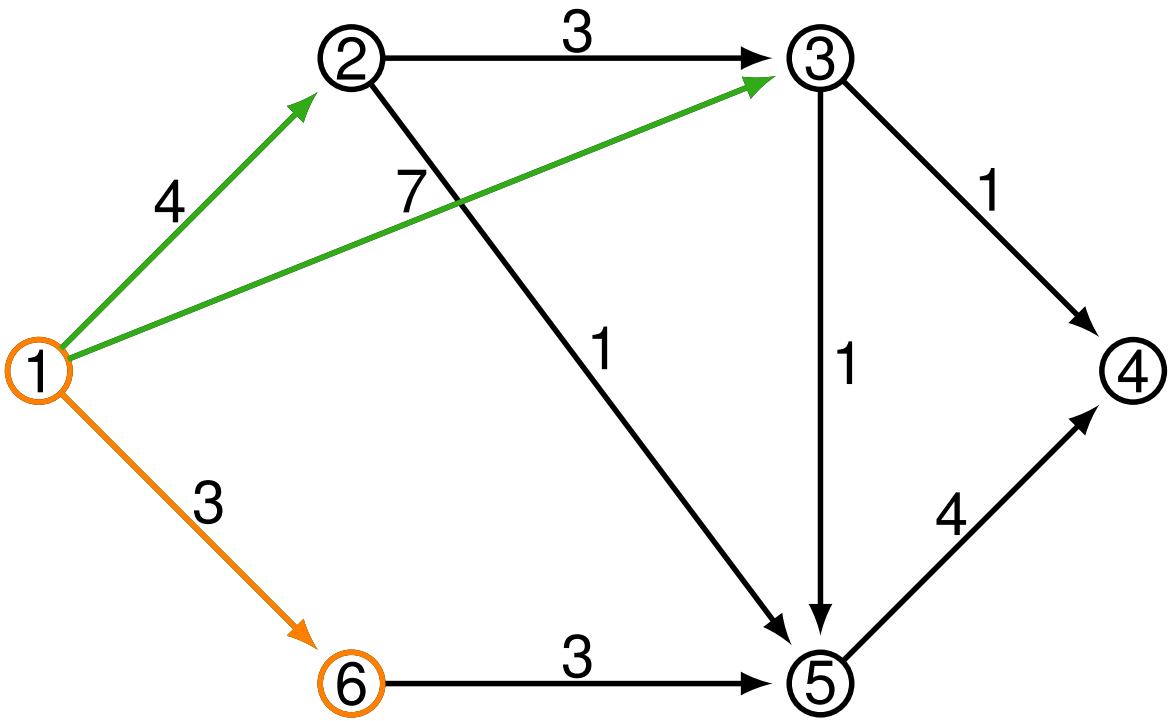
Algoritmo de Dijkstra - Ejemplo

$S = \{1\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



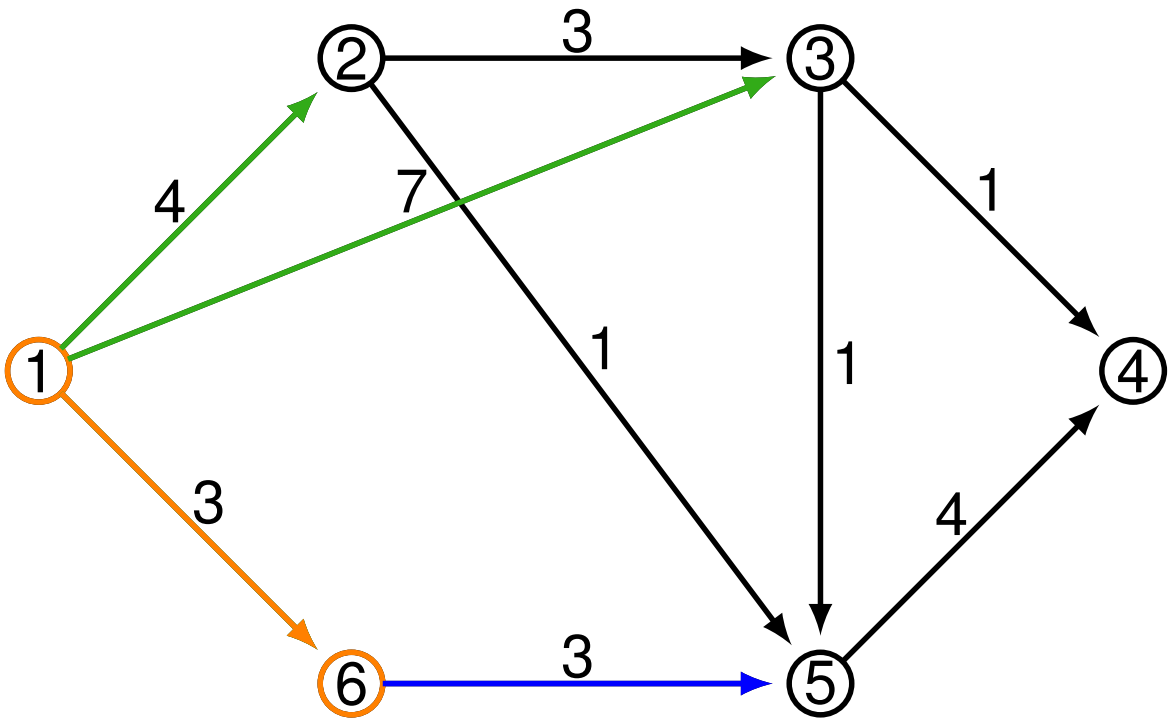
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



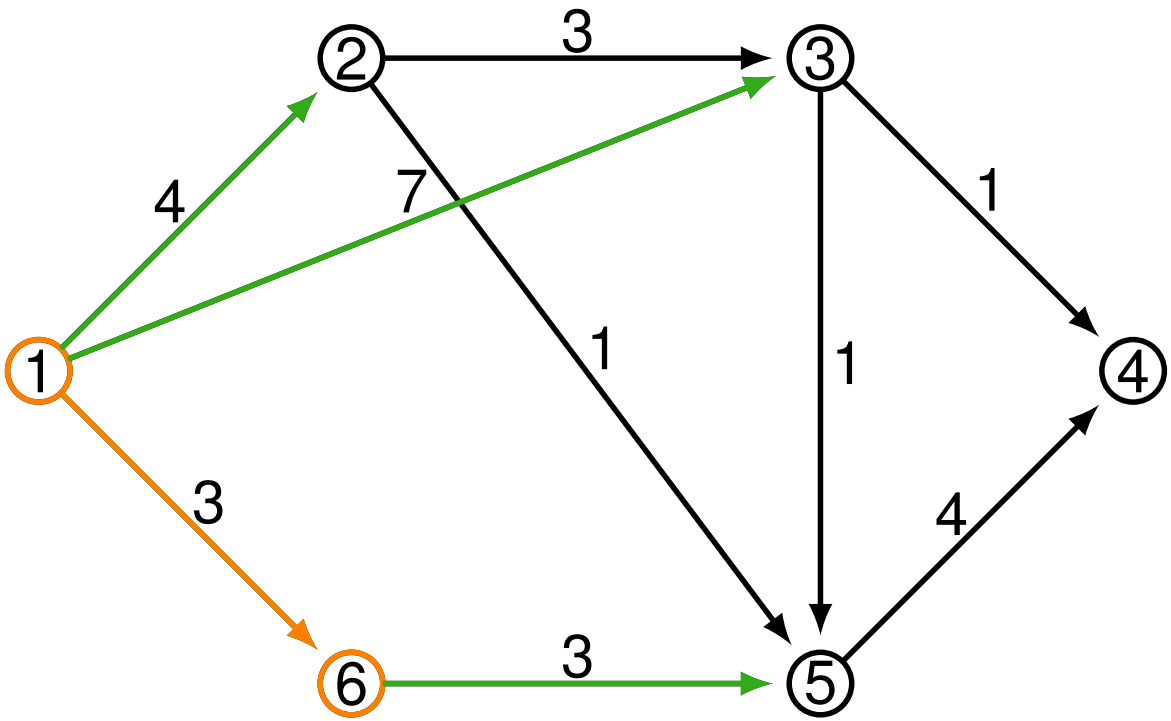
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



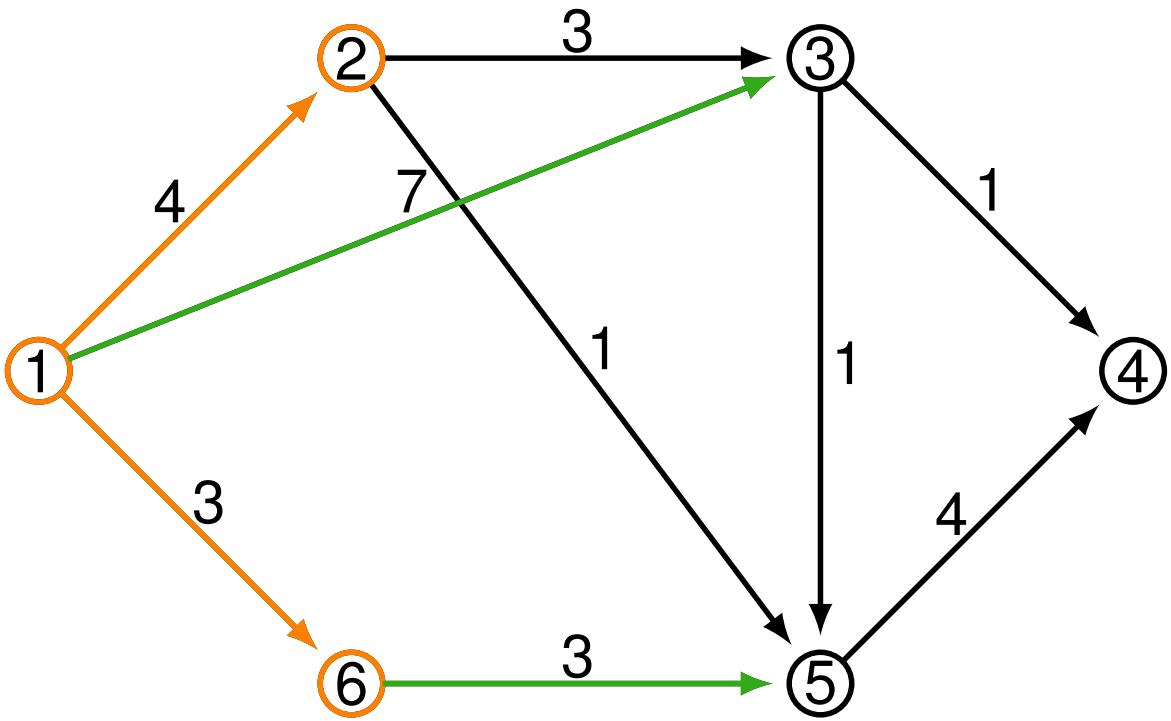
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



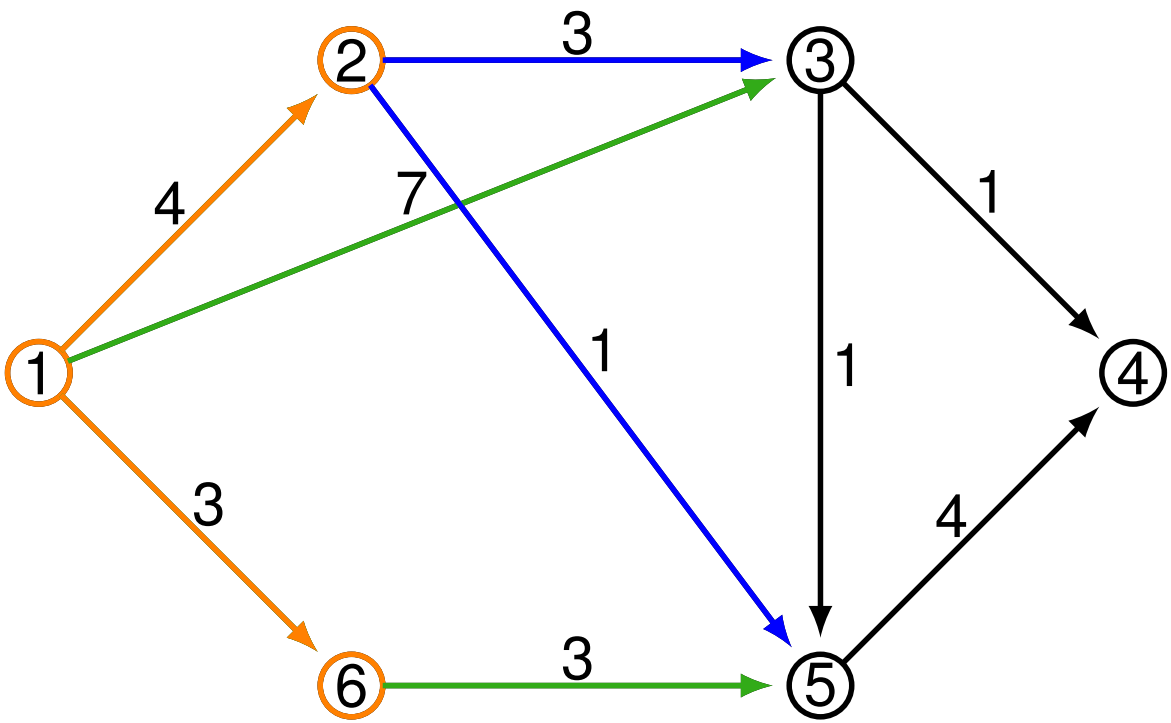
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



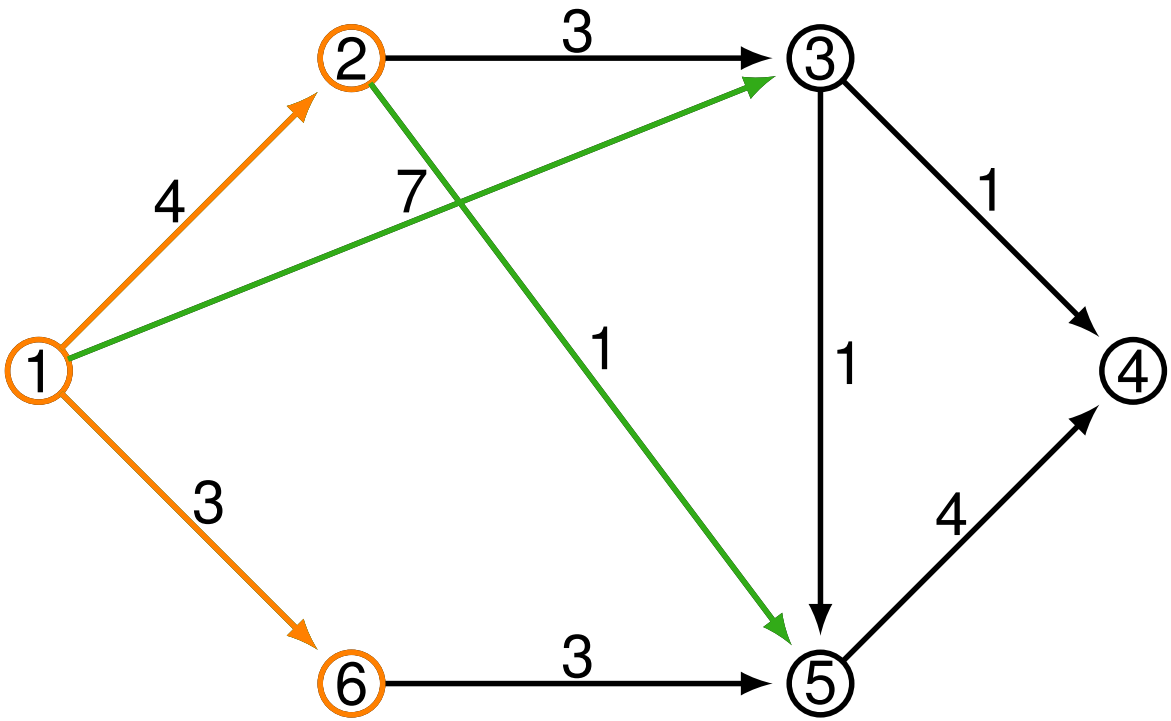
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



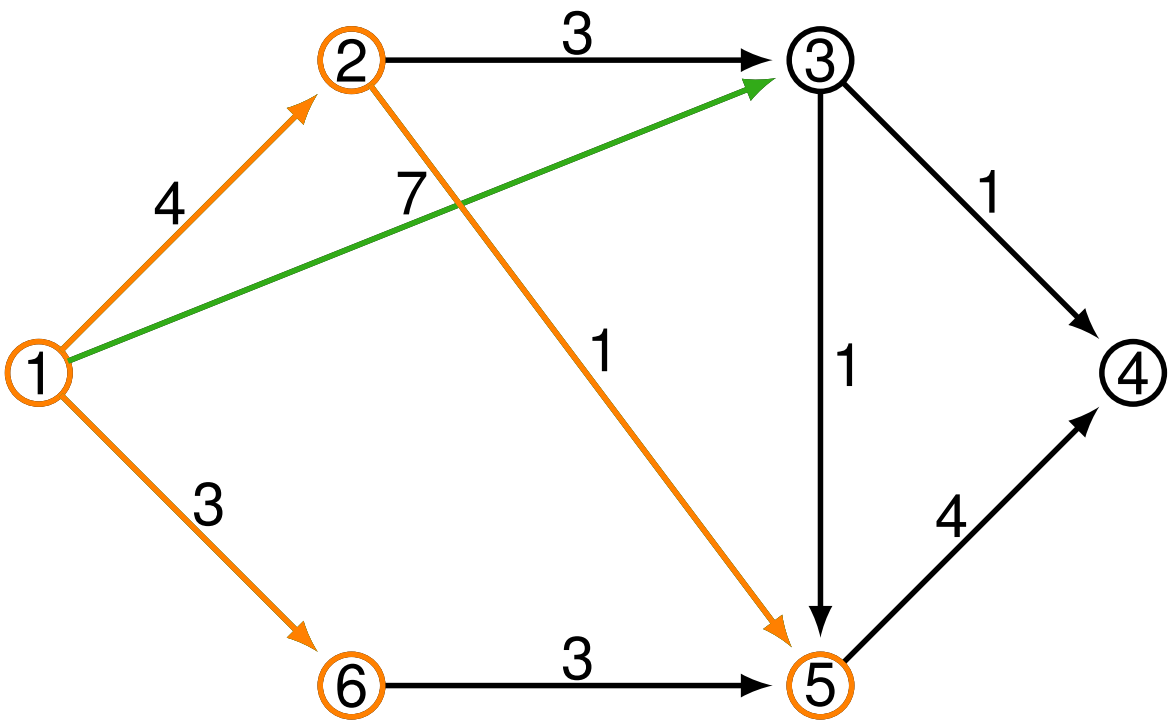
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



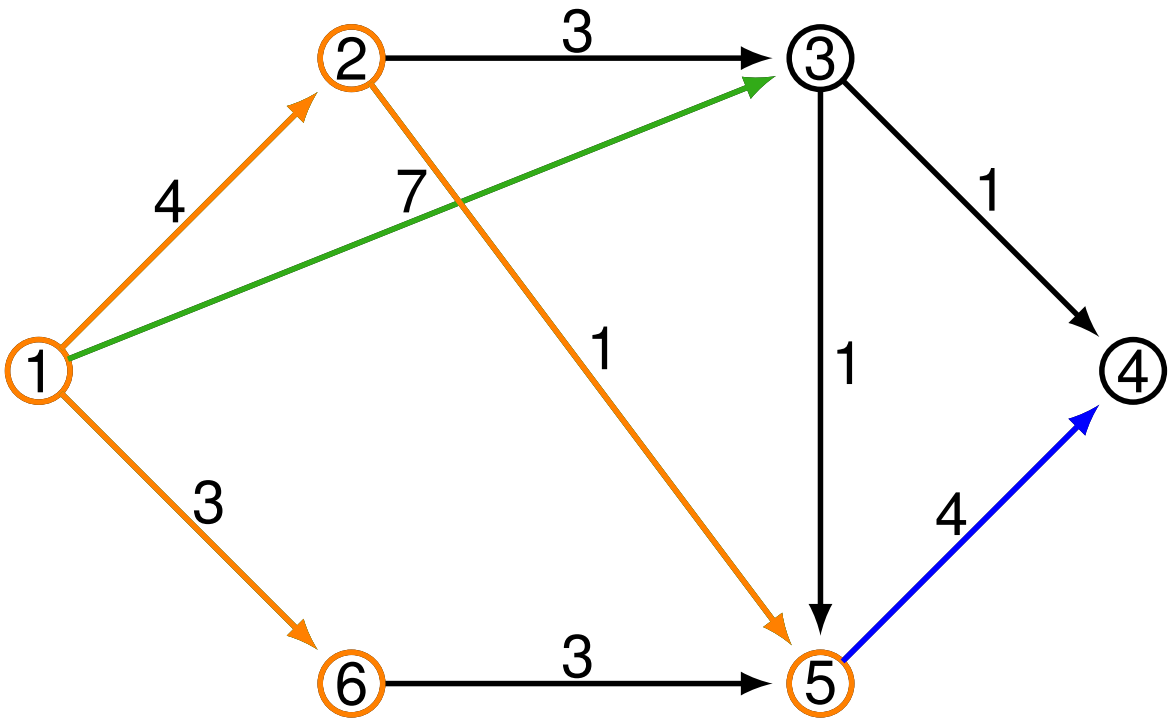
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



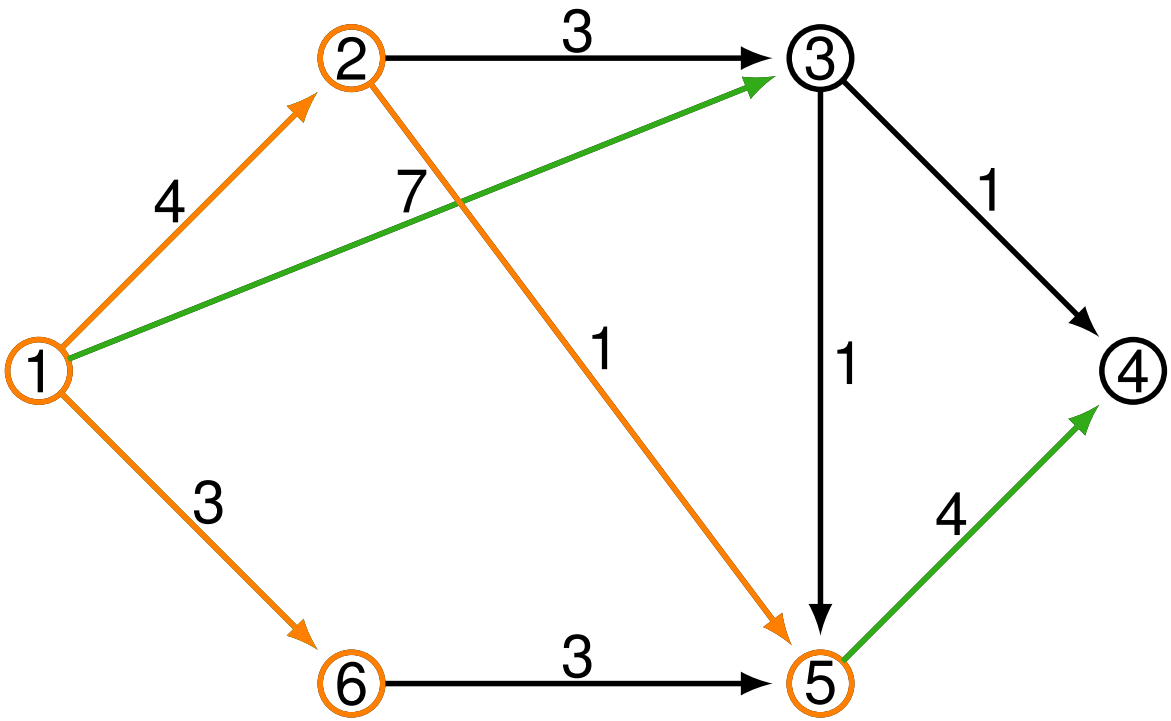
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



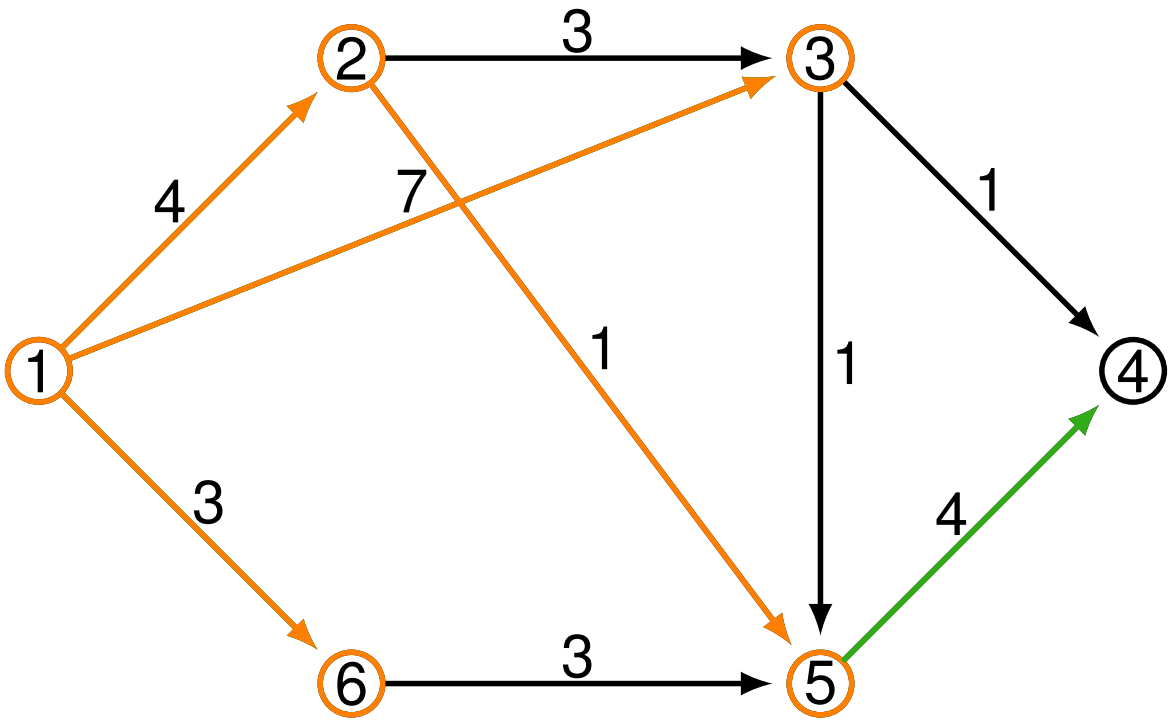
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, 9, 5, 3)$



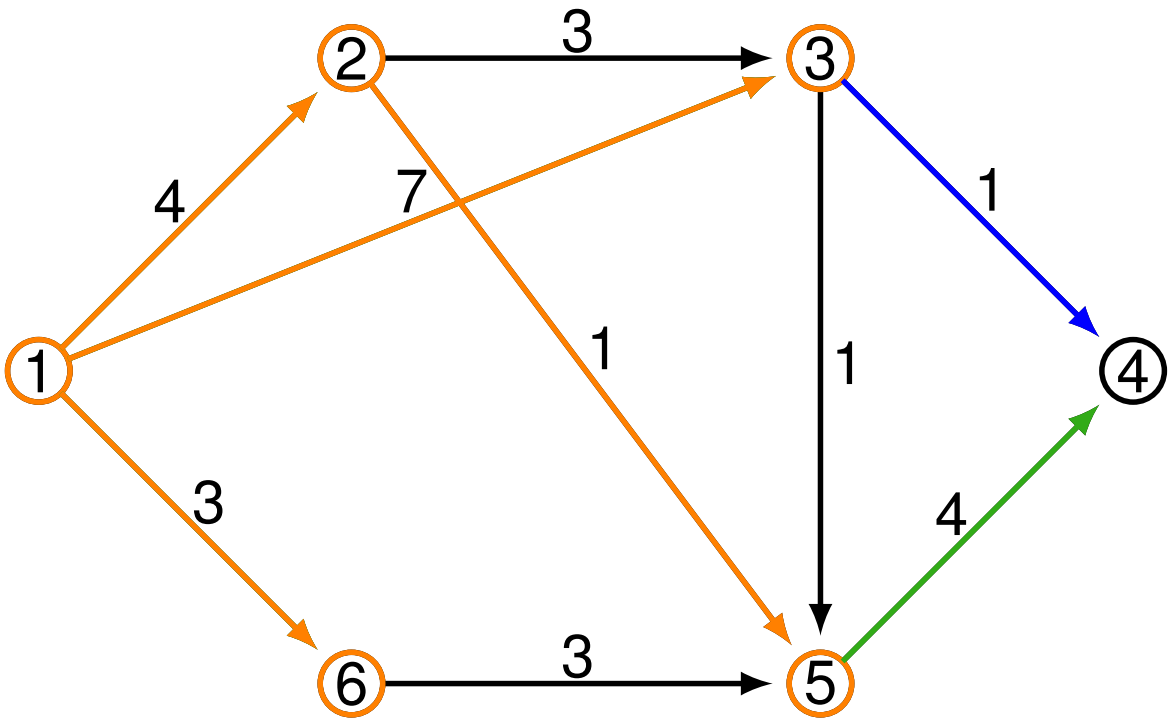
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$



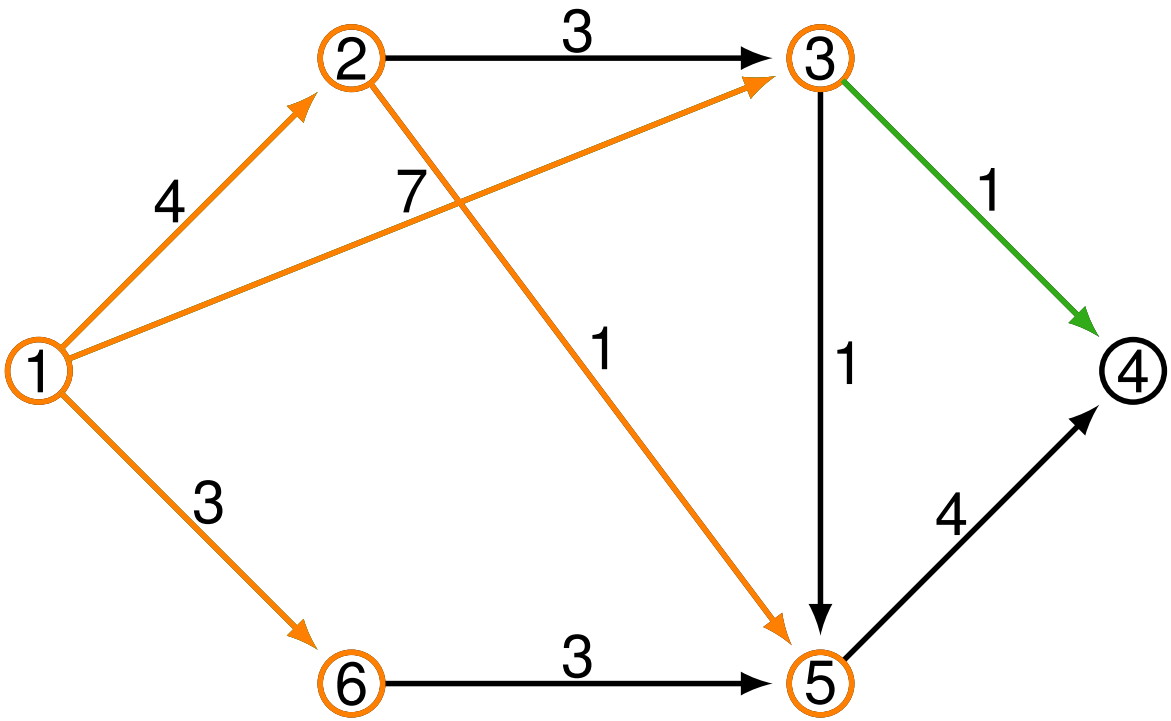
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$



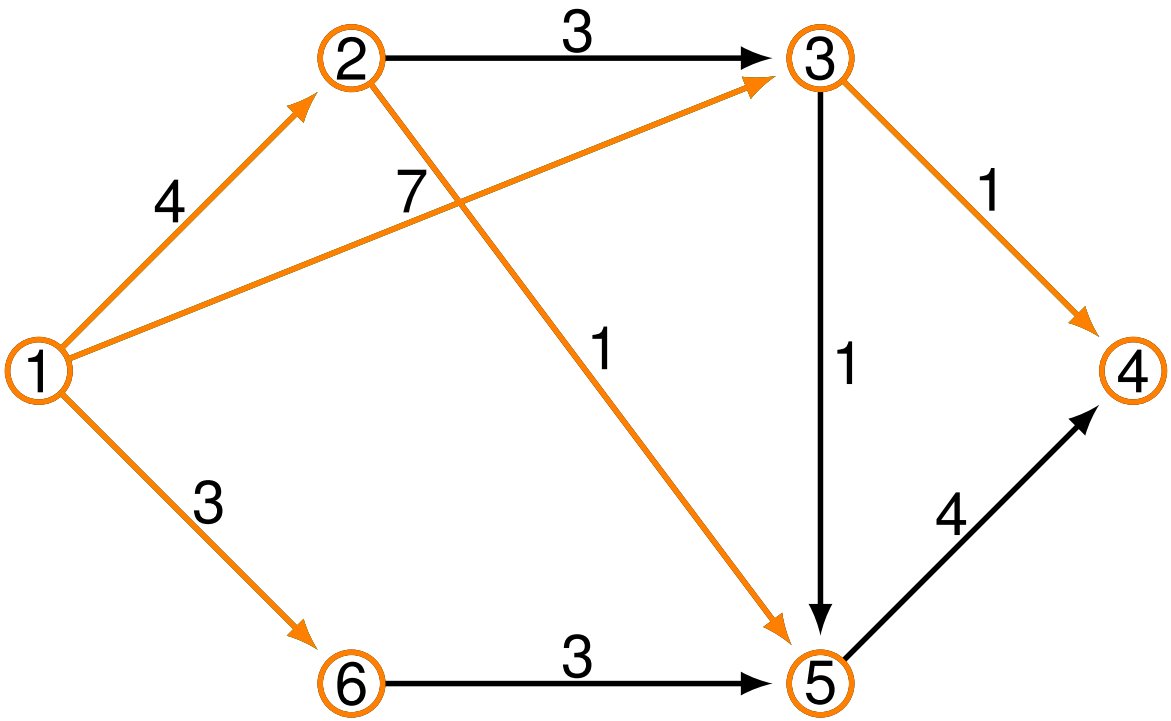
Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 8, 5, 3)$



Algoritmo de Dijkstra - Ejemplo

$S = \{1, 6, 2, 5, 3, 4\}$ $\pi = (0, 4, 7, 8, 5, 3)$



Contenidos

¿Por qué funciona este algoritmo?

Si bien no veremos la demostración formal, la clave está en que:

- En cada paso, para todos los nodos u que ya fueron visitados, el algoritmo tiene calculada la mínima distancia del nodo inicial a u. Para los nodos v que aún no fueron visitados, el algoritmo tiene calculada la distancia mínima si solo podemos utilizar nodos ya visitados como puntos intermedios del camino.
- El próximo nodo a ser visitado es el más cercano al nodo inicial que aún no fue visitado. Entonces, este ya tiene calculada la distancia correcta (si no, habría una mejor solución usando nodos no visitados como nodos intermedios, pero este nodo es el más cercano al origen entre todos los no visitados!).
- Al comienzo, las distancias están bien calculadas.

Pseudocódigo de Dijkstra sin cola de prioridad

```
1  Dijkstra (Grafo G, nodo inicial s)
2    visitado[n] = {false, ..., false} // guarda si un nodo ya fue visitado
3    distancia[n] = {Infinito, ..., Infinito} // guarda las distancias del nodo
         salida al resto
4
5    para cada w en V[G] hacer
6        si existe arista entre s y w entonces
7            distancia[w] = peso (s, w)
8
9    distancia[s] = 0
10   visitado[s] = true
11
12   mientras que no esten visitados todos hacer
13       v = nodo de menor distancia a s que no fue visitado aun
14       visitado[v] = true
15       para cada w en sucesores (G, v) hacer
16           si distancia[w] > distancia[v] + peso (v, w) entonces
17               distancia[w] = distancia[v] + peso (v, w)
18               padre[w] = v
```

Costo de esta implementación de Dijkstra

Podemos ver que utiliza $O(n^2)$ operaciones, siendo n la cantidad de nodos del problema.

Otra implementación

Existen otras implementaciones del algoritmo de Dijkstra. La que veremos a continuación solo difiere de la anterior en la forma de guardar los nodos candidatos a ser visitados.

Pseudocódigo de Dijkstra con cola de prioridad

```
1  Dijkstra (Grafo G, nodo_fuente s)
2      para todo u en V[G] hacer
3          distancia[u] = INFINITO
4          padre[u] = NULL
5          visitado[u] = false
6
7      distancia[s] = 0
8      adicionar (cola, (s, distancia[s]))
9
10     mientras que cola no sea vacia hacer
11         u = extraer_minimo(cola)
12         visitado[u] = true
13         para todo v en adyacencia[u] hacer
14             si no visitado[v] y distancia[v] > distancia[u] + peso (u, v)
                   hacer
15                 distancia[v] = distancia[u] + peso (u, v)
16                 padre[v] = u
17                 adicionar(cola,(v, distancia[v]))
```

Costo de esta implementación de Dijkstra

Podemos ver que utiliza $O(m \log n)$ operaciones, siendo n la cantidad de nodos del problema y m la cantidad de aristas. No veremos hoy la demostración, pero se basa en que las estructuras que usamos para la cola de prioridad extraen e insertan un elemento en tiempo logarítmico.

Existen varias estructuras en C++ para implementar cola de prioridad. Dos ejemplos son `priority_queue` y `set`.

¿Qué implementación uso?

- Si el grafo es **ralo**, o sea, tiene pocas aristas, conviene utilizar

¿Qué implementación uso?

- Si el grafo es **ralo**, o sea, tiene pocas aristas, conviene utilizar la implementación con cola de prioridad ($O(m \log n)$)

¿Qué implementación uso?

- Si el grafo es **ralo**, o sea, tiene pocas aristas, conviene utilizar la implementación con cola de prioridad ($O(m \log n)$)
- Si el grafo es **denso**, o sea, tiene muchas aristas, conviene utilizar

¿Qué implementación uso?

- Si el grafo es **ralo**, o sea, tiene pocas aristas, conviene utilizar la implementación con cola de prioridad ($O(m \log n)$)
- Si el grafo es **denso**, o sea, tiene muchas aristas, conviene utilizar la implementación básica ($O(n^2)$)

Contenidos

Hallemos el camino mínimo entre dos casillas (a, b) y (c, d) de una matriz de $m \times n$. Al programar nos puede servir usar los siguientes trucos de los que hablaremos:

- Vector de movimientos (dx, dy)
- Representar una dirección (0, 1, 2, 3 en los arreglos de arriba) y un giro (sumar mod 4)
- Bordes (hacer padding)

Contenidos

¿Qué es un árbol?

Árboles

Un árbol es un tipo especial de grafo no dirigido. Es un grafo donde entre cualquier par de nodos existe un único camino que los conecta (notar que en particular, todos los nodos tienen que estar conectados entre sí).

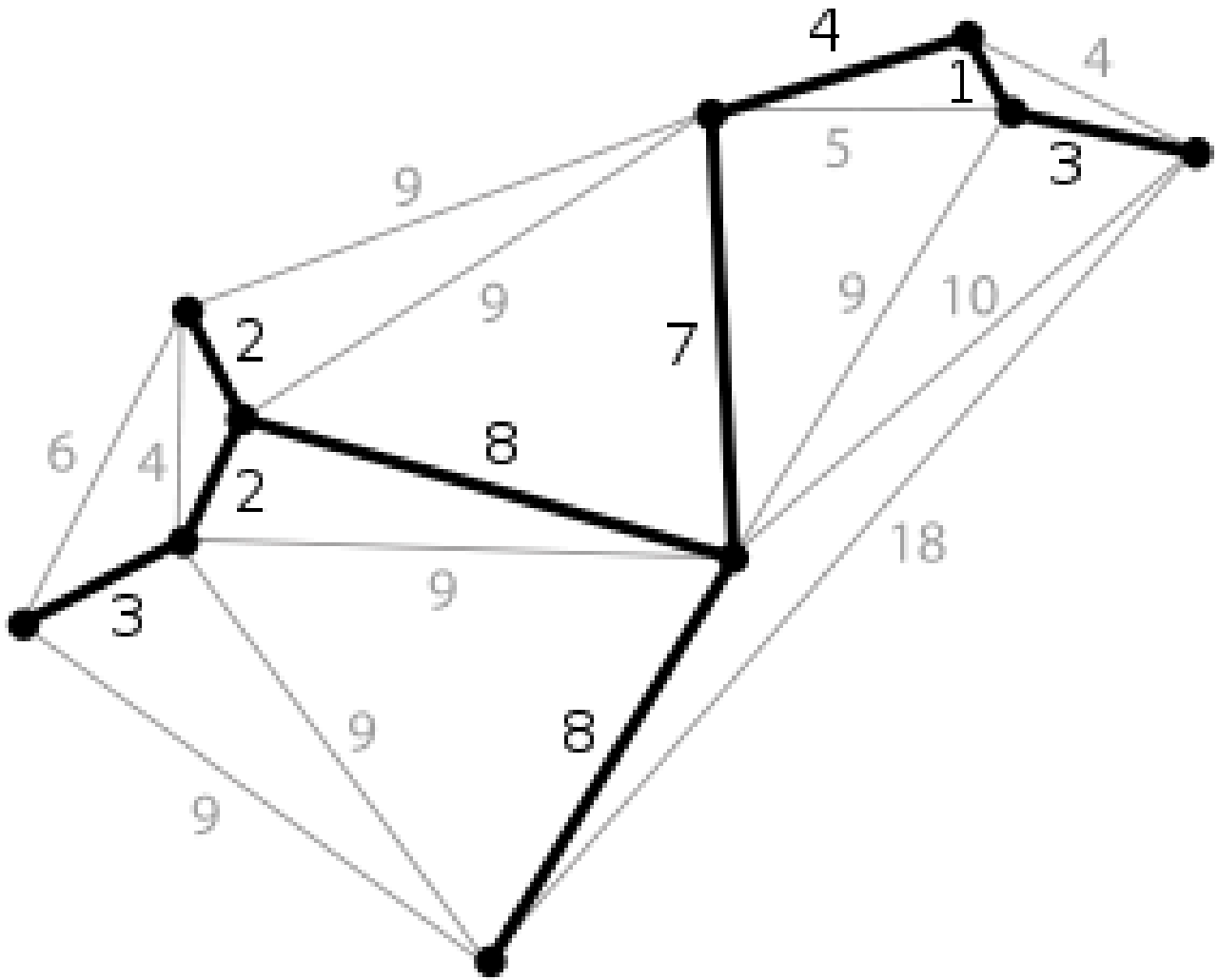
Los árboles tienen mil propiedades interesantes, como $m = n - 1$, que no tiene ciclos y más cosas sobre las que hoy no hablaremos.

¿Qué es un árbol generador mínimo?

Un **árbol generador mínimo** es un árbol que utiliza todos los nodos del grafo, de manera tal que el costo total de sumar las aristas del grafo es mínimo.

Veamos un ejemplo.

Ejemplo de árbol generador mínimo



¿Para qué sirve encontrar un árbol generador mínimo?

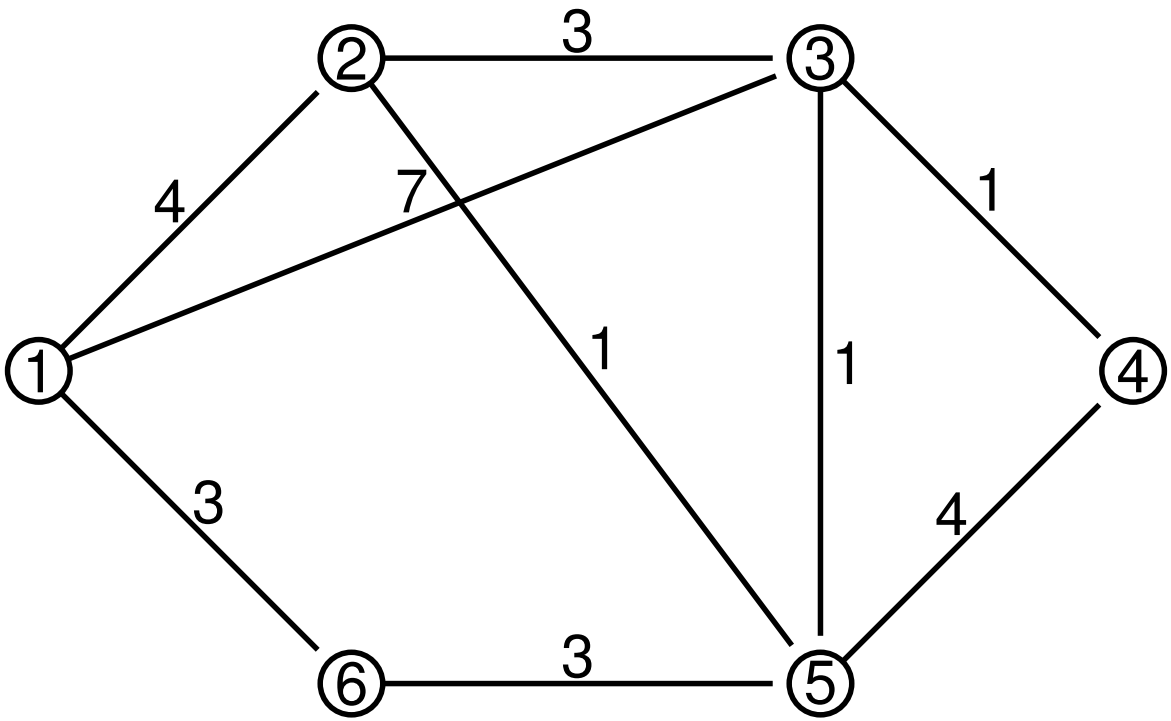
Este problema es muy relevante, porque en diversas situaciones hallarlo resuelve nuestro problema. Muchas veces, a simple vista no es nada obvio que se resuelve con esto.

En un país con muchas ciudades, nos piden que creemos rutas de manera tal que se pueda viajar de cualquier ciudad a cualquier otra usando la menor cantidad de dinero posible. Para algunos pares de ciudades (que son las ciudades entre las que tenemos permitido crear una ruta) sabemos el costo de crear una ruta entre ellos. ¿Qué rutas deberíamos construir para lograr el objetivo?

Contenidos

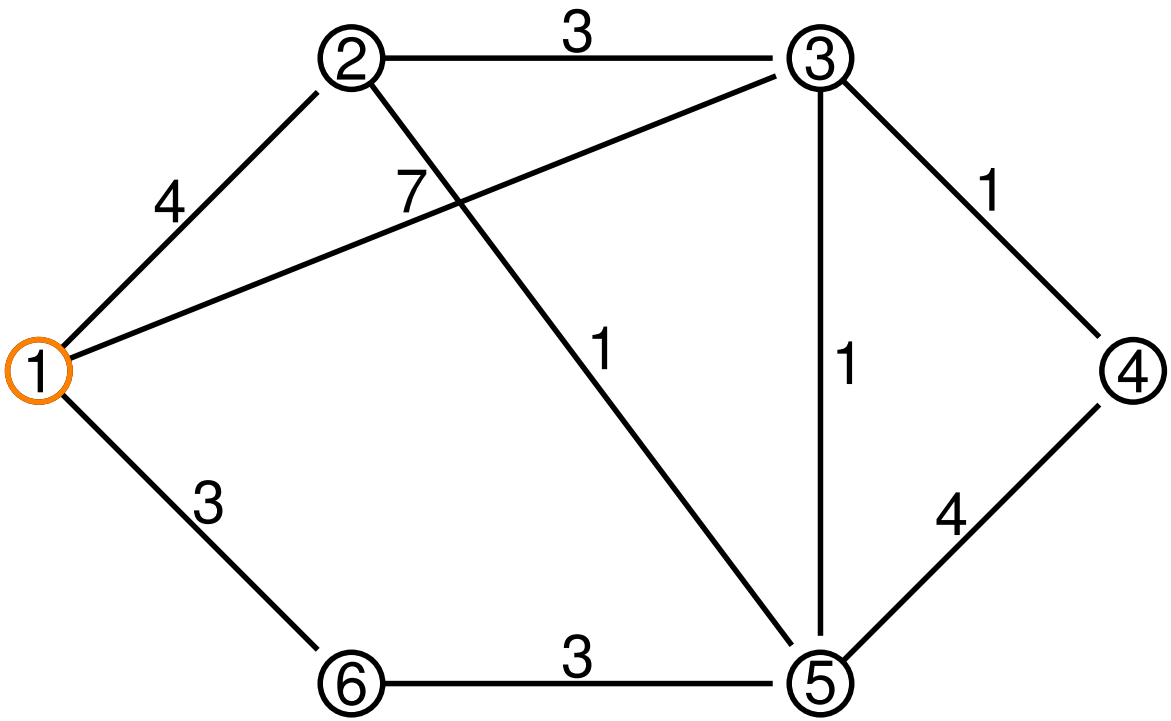
- El algoritmo de Prim mantiene un conjunto de nodos C que son los nodos que ya fueron conectados entre sí.
- En cada paso, elige la arista más barata entre algún nodo de C y un nodo no agregado todavía. Al elegir la arista, se agrega el nodo nuevo a C.
- De esta forma, en cada paso hay un nodo nuevo sumándose al conjunto de nodos ya conectados entre sí.
- Luego de $n - 1$ pasos, habremos agregado todos los nodos de la forma más barata posible: obtendremos un árbol generador mínimo.

Algoritmo de Prim - Ejemplo



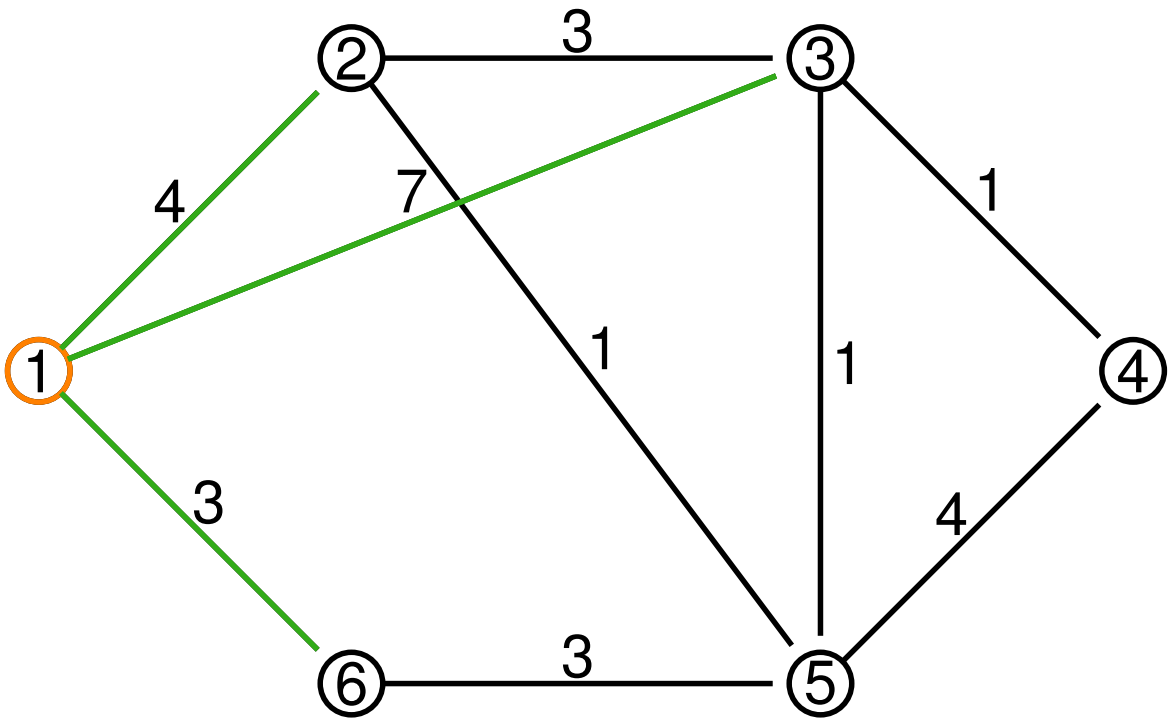
Algoritmo de Prim - Ejemplo

$S = \{1\}$ $\pi = (0, \infty, \infty, \infty, \infty, \infty)$



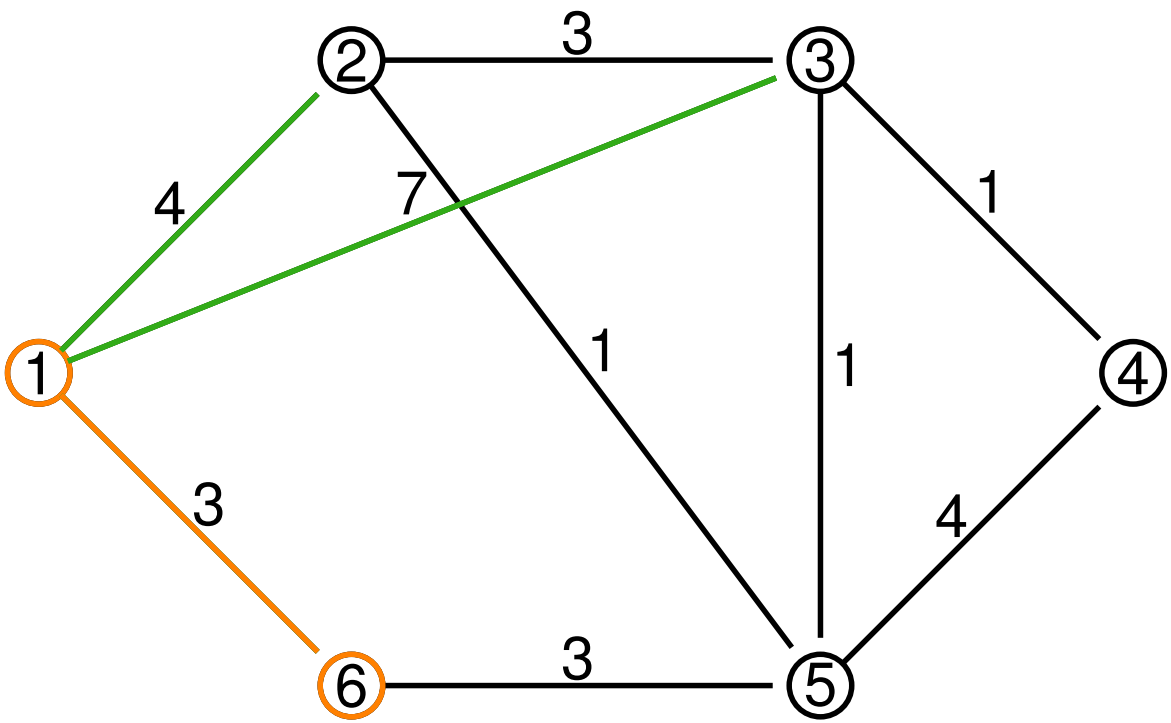
Algoritmo de Prim - Ejemplo

$S = \{1\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



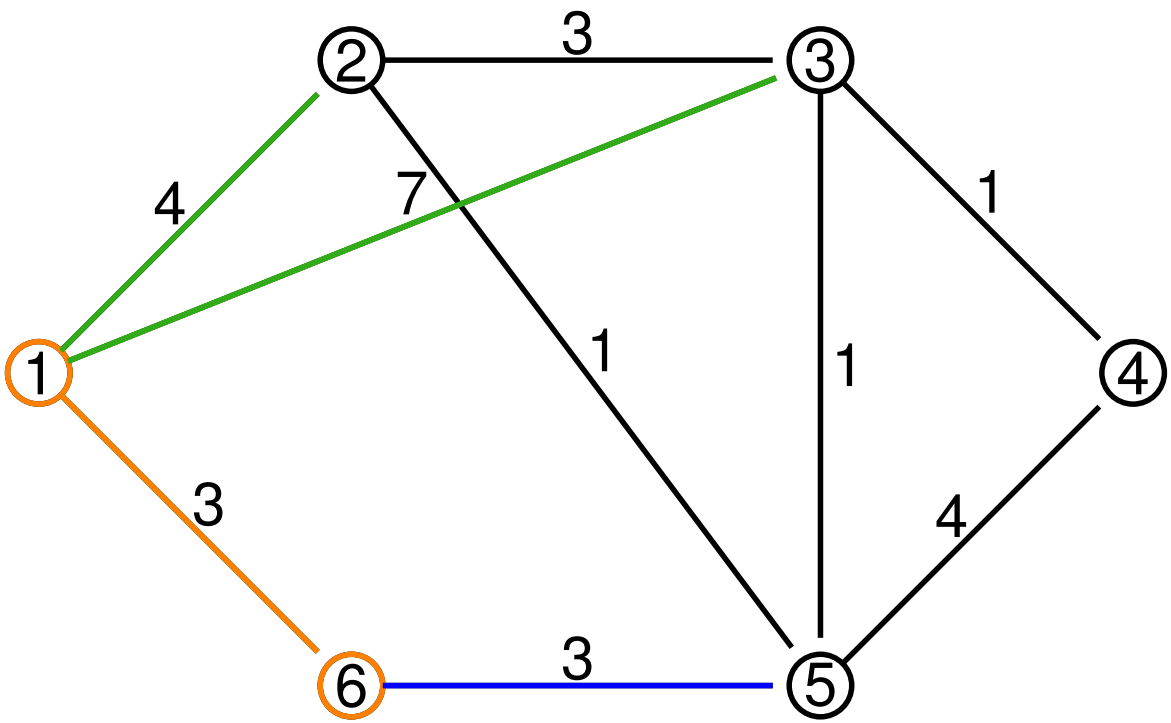
Algoritmo de Prim - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



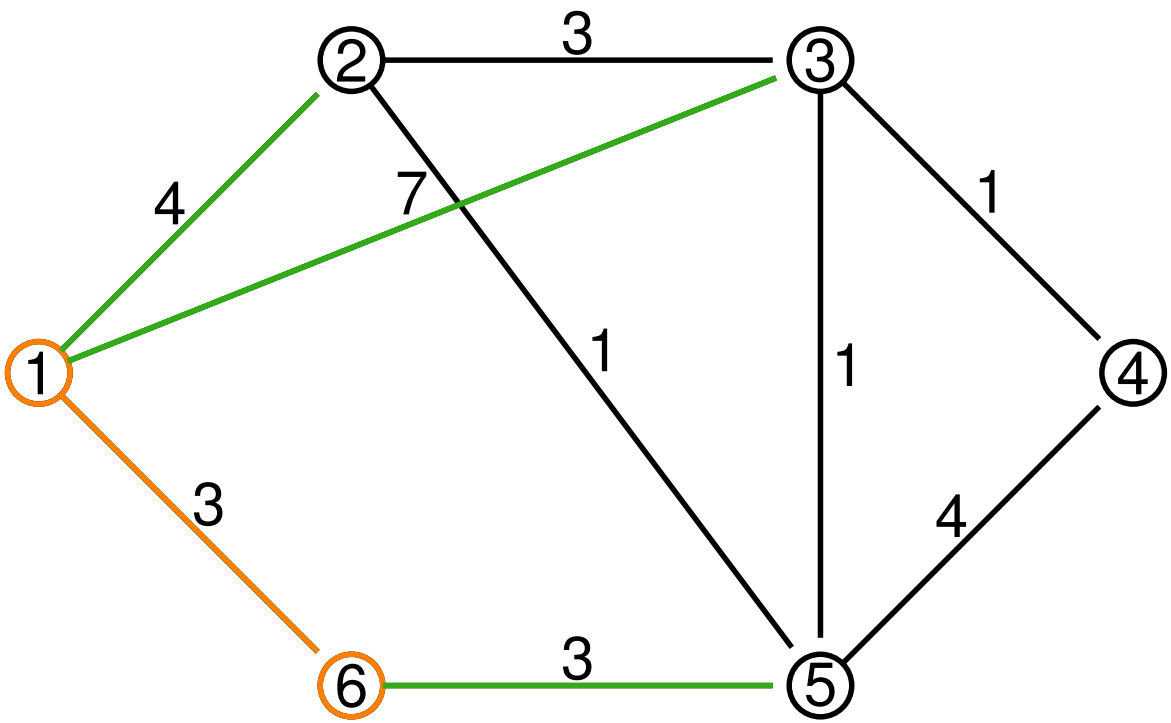
Algoritmo de Prim - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, \infty, 3)$



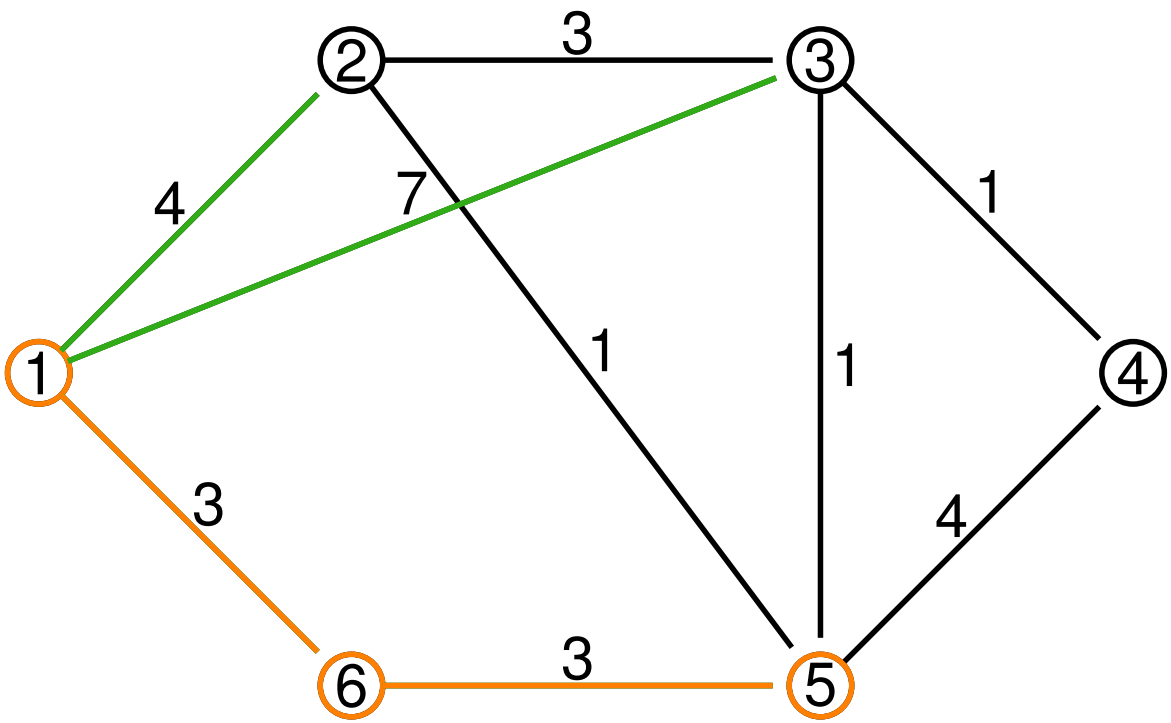
Algoritmo de Prim - Ejemplo

$S = \{1, 6\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



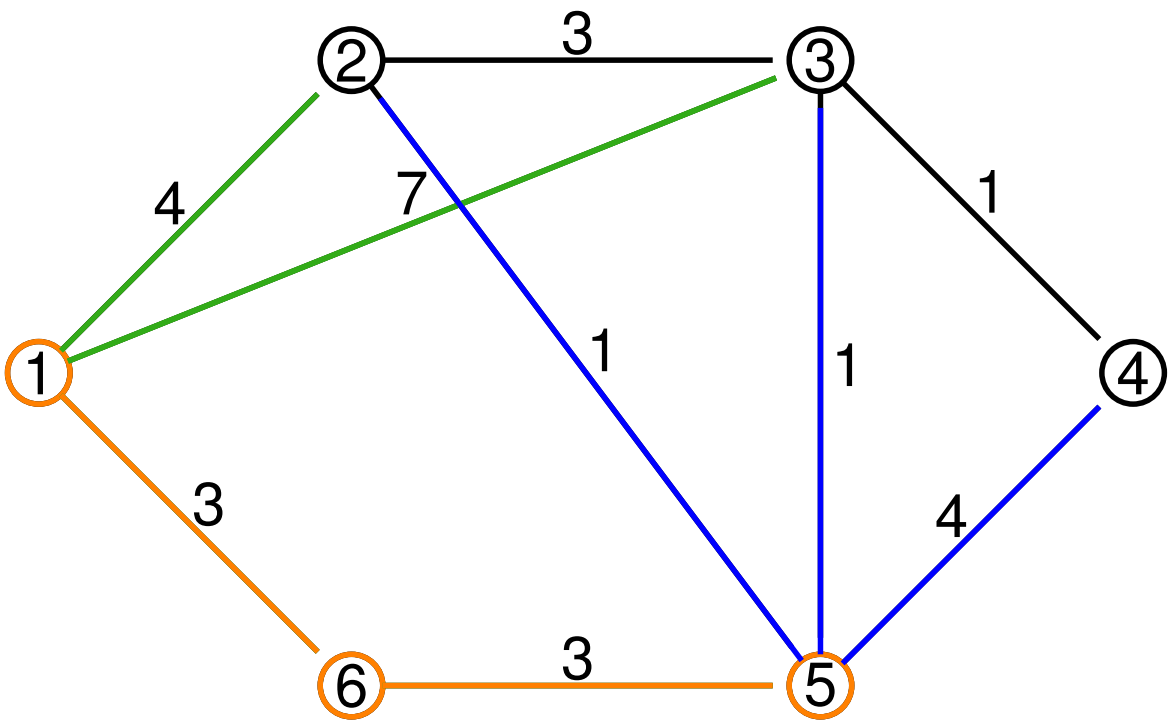
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



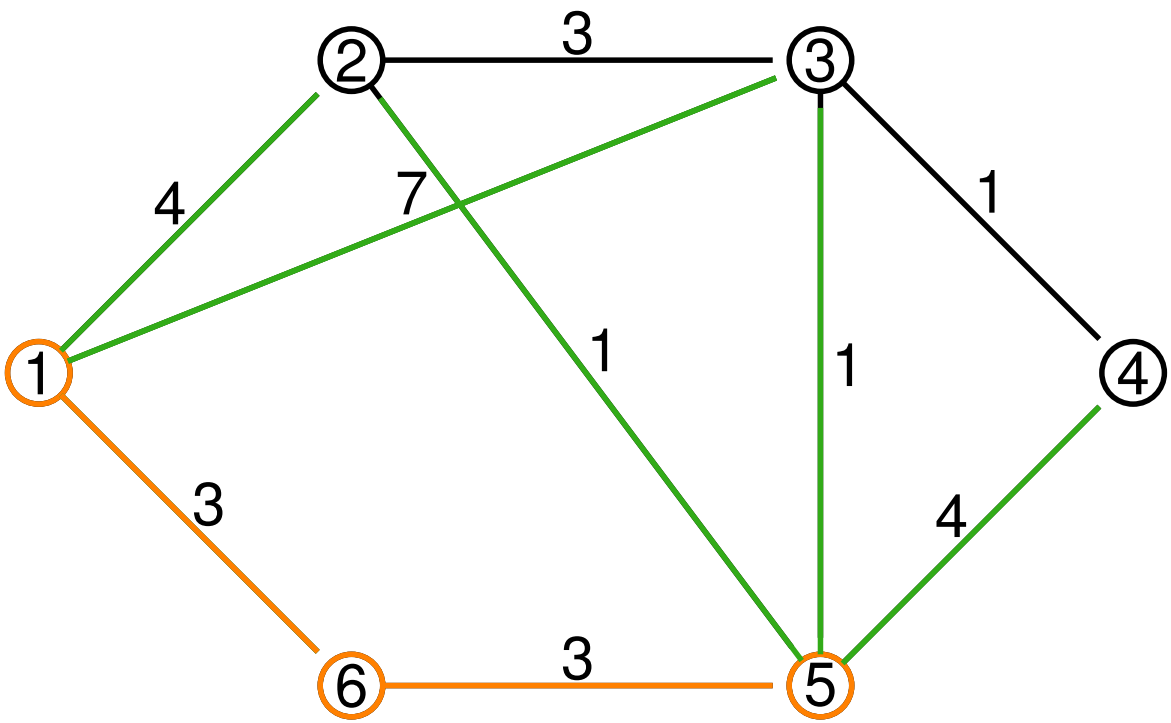
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 6, 3)$



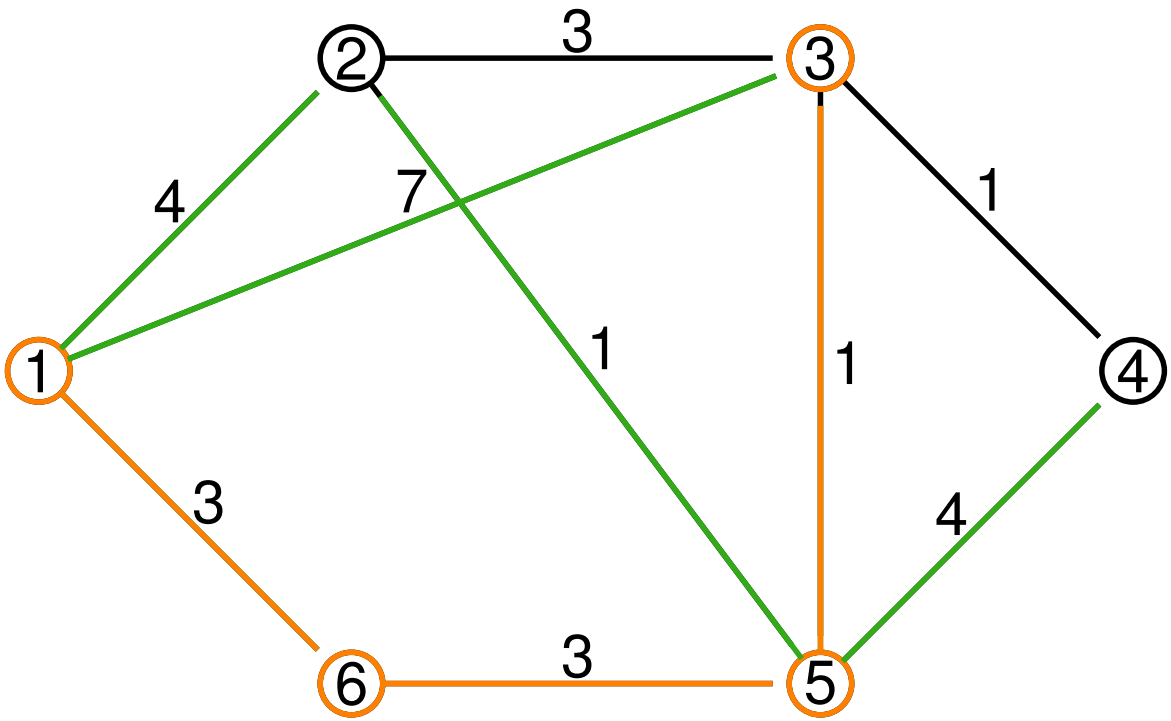
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



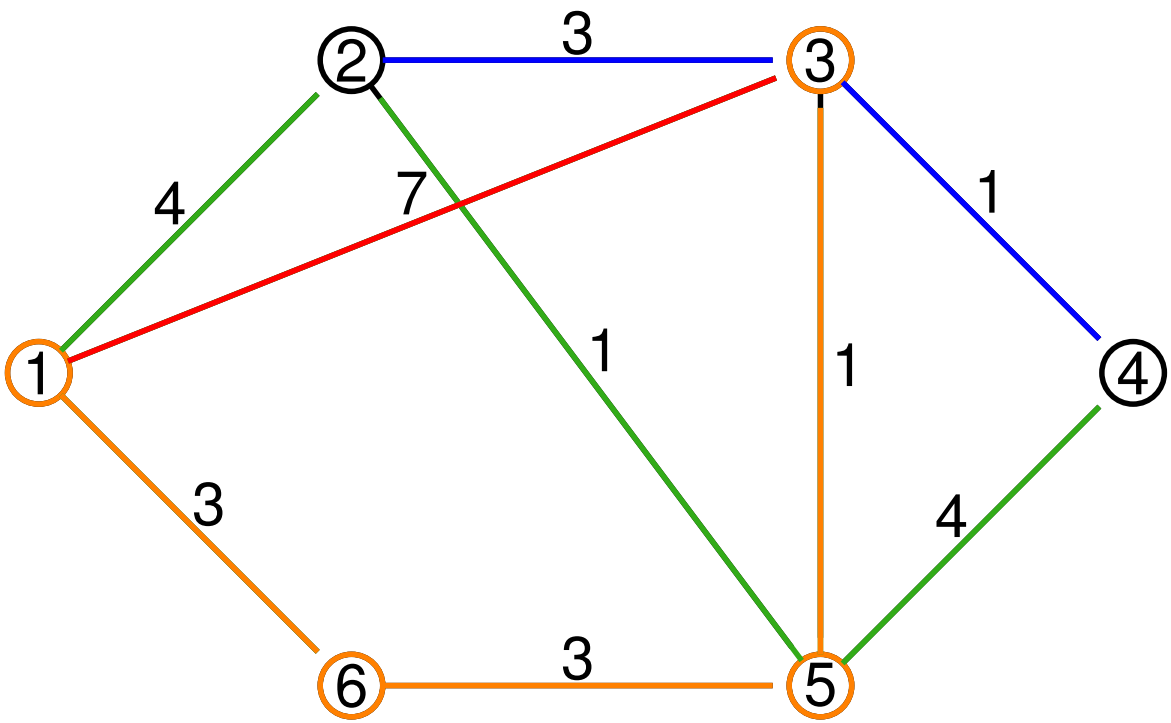
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



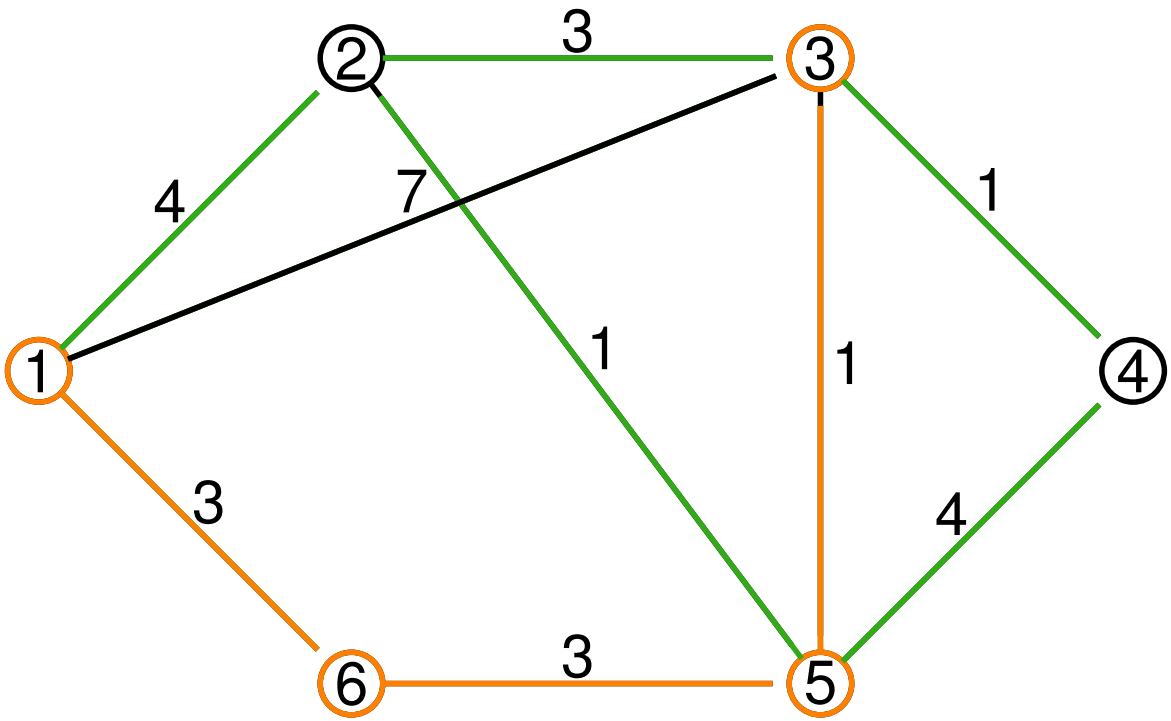
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, \infty, 5, 3)$



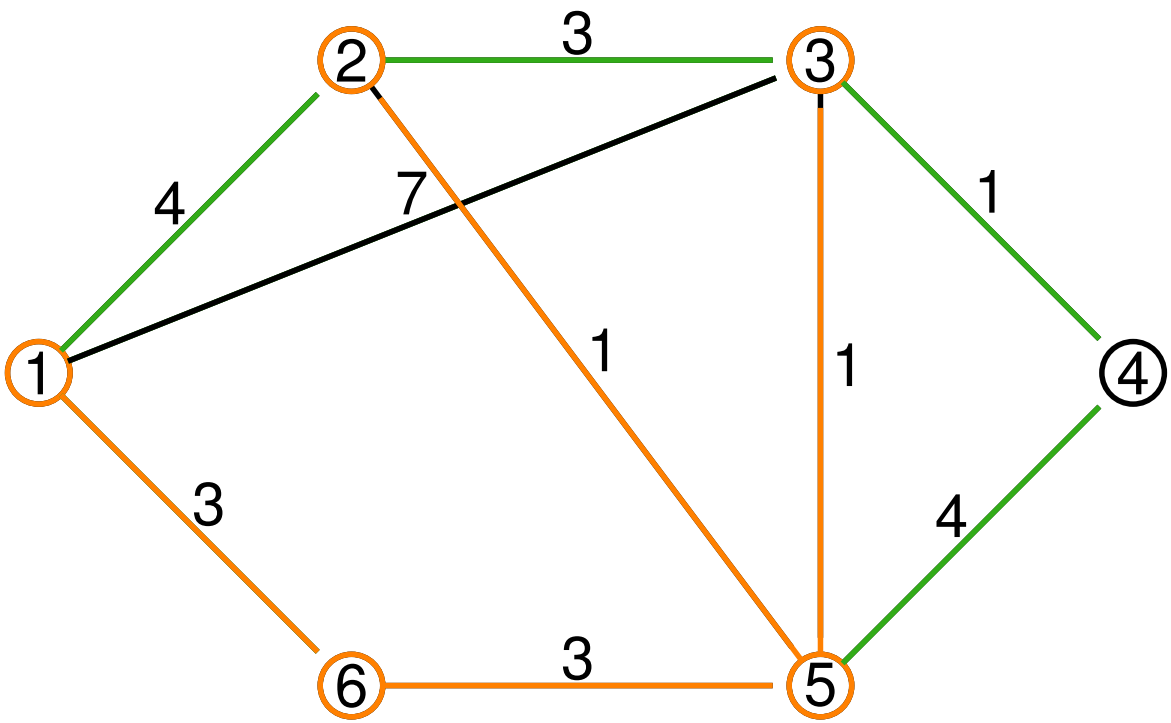
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5\}$ $\pi = (0, 4, 7, 9, 5, 5, 3)$



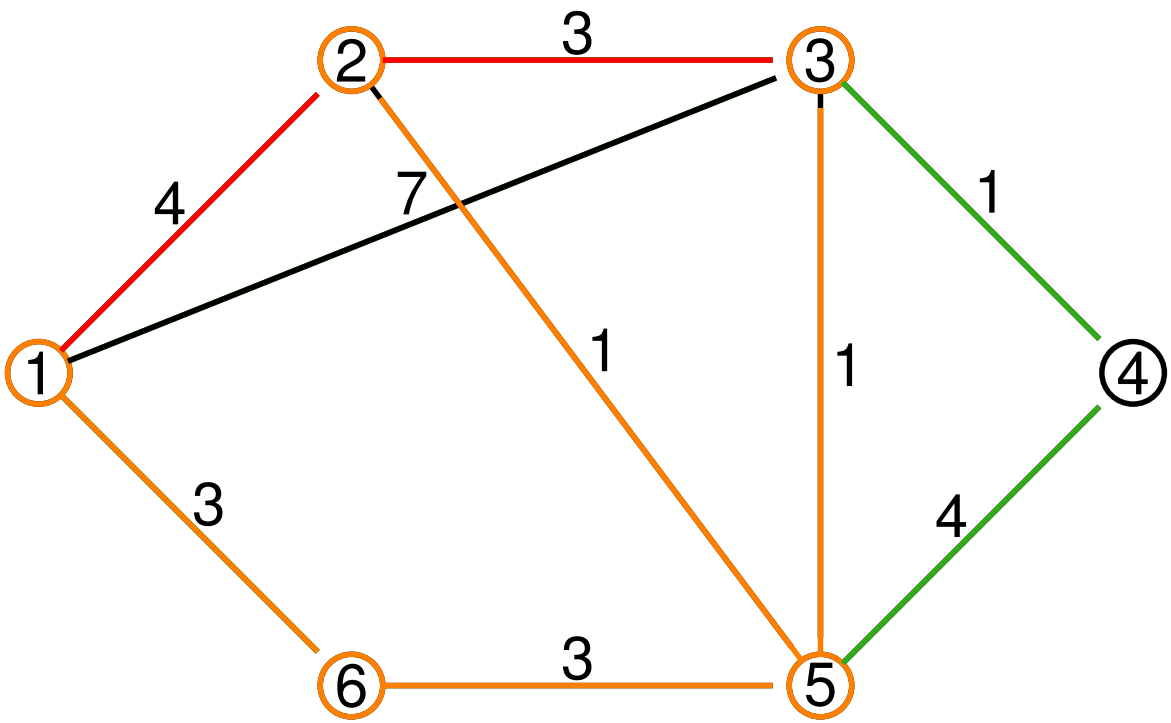
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$



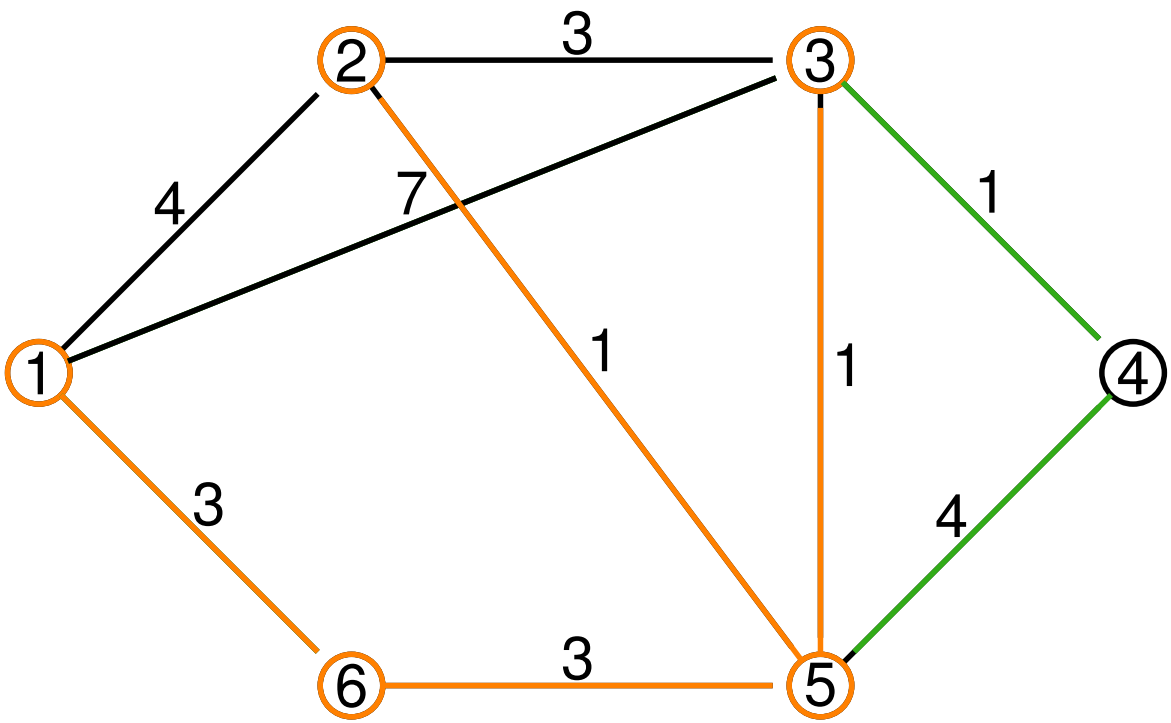
Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5, 3\}$ $\pi = (0, 4, 7, 9, 5, 3)$

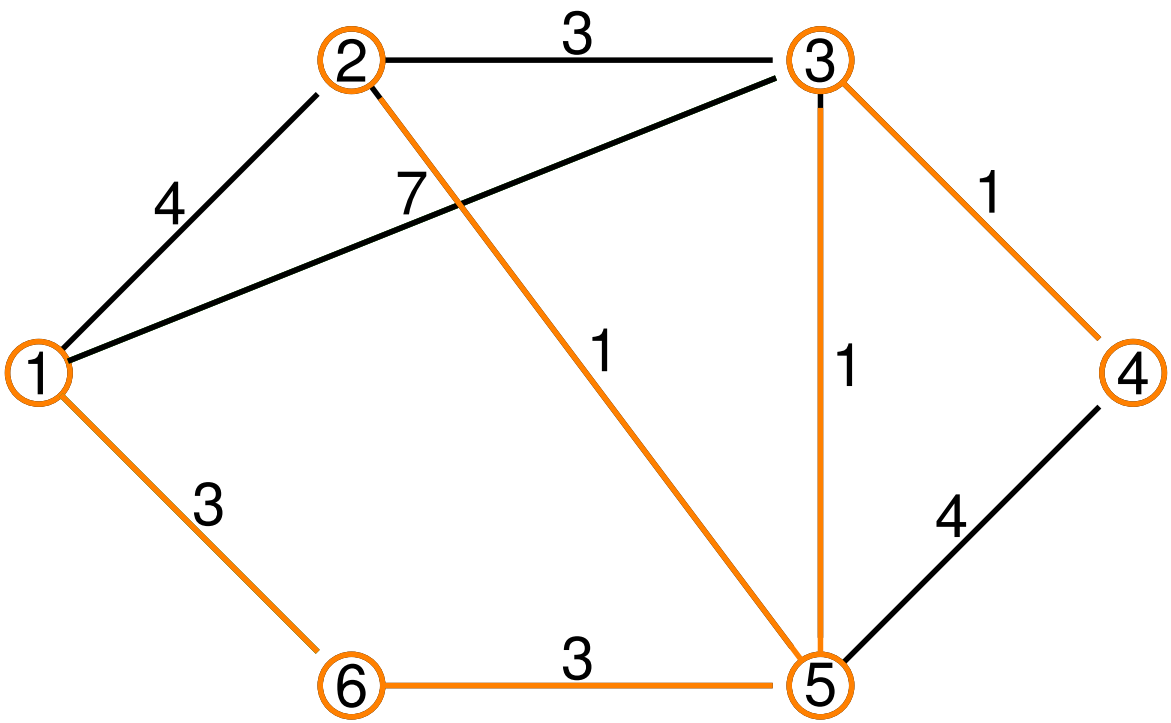


Algoritmo de Prim - Ejemplo

$S = \{1, 6, 2, 5, 3\}$



Algoritmo de Prim - Ejemplo



Pseudocódigo de Prim sin cola de prioridad

```
1  Prim (Grafo G, nodo inicial s)
2    visitado[n] = {false, ..., false} // guarda si un nodo ya fue visitado
3    distancia[n] = {Infinito, ..., Infinito} // guarda las distancias de cada
         nodo al conjunto visitado
4
5    para cada w en V[G] hacer
6        si existe arista entre s y w entonces
7            distancia[w] = peso (s, w)
8
9    distancia[s] = 0
10   visitado[s] = true
11   mientras que no esten visitados todos hacer
12       v = nodo de menor distancia del conjunto que no fue visitado aun (itero
             el arreglo)
13       visitado[v] = true
14       para cada w en los vecinos de v hacer
15           si distancia[w] > peso (v, w) entonces
16               distancia[w] = peso (v, w)
17               padre[w] = v
```

Pseudocódigo de Prim con cola de prioridad

```
1  Prim (Grafo G, nodo_fuente s)
2      para todo u en V[G] hacer
3          distancia[u] = INFINITO
4          padre[u] = NULL
5          visitado[u] = false
6
7      distancia[s] = 0
8      adicionar (cola, (distancia[s], s))
9
10     mientras que cola no sea vacia hacer
11         u = extraer_minimo(cola)
12         visitado[u] = true
13         para todo v en vecinos de u hacer
14             si no visitado[v] y distancia[v] > peso(u, v) hacer
15                 distancia[v] = peso(u, v)
16                 padre[v] = u
17                 adicionar(cola, (distancia[v], v))
```

¿Preguntas?