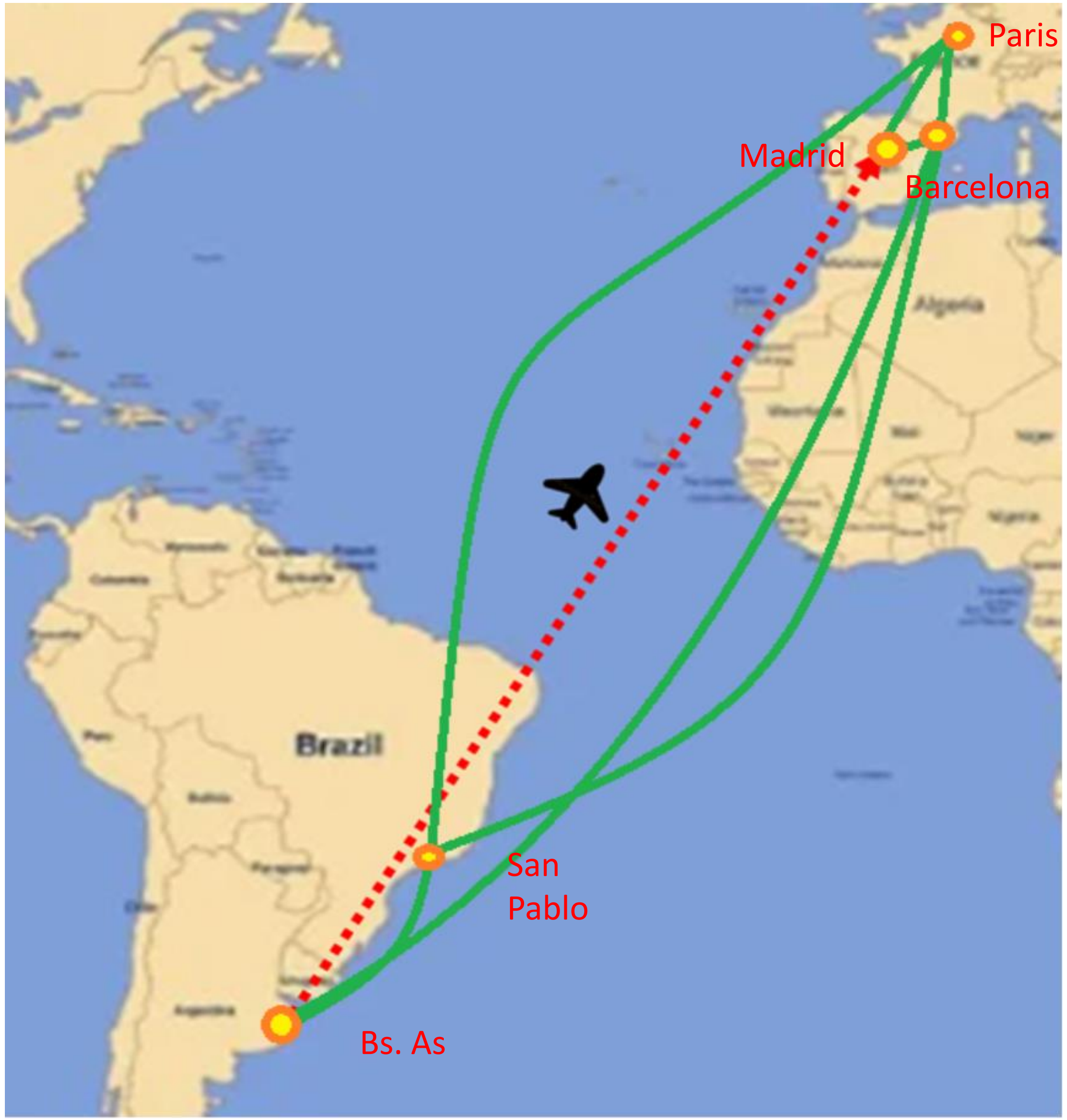
Problema de los caminos más cortos

Algoritmo de Dijkstra

¿CUÁL ES LA FORMA MÁS ECONÓMICA DE LLEGAR A MADRID?

Alternativas de viaje

De las distintas ofertas seleccioné las que me parecían más favorables.

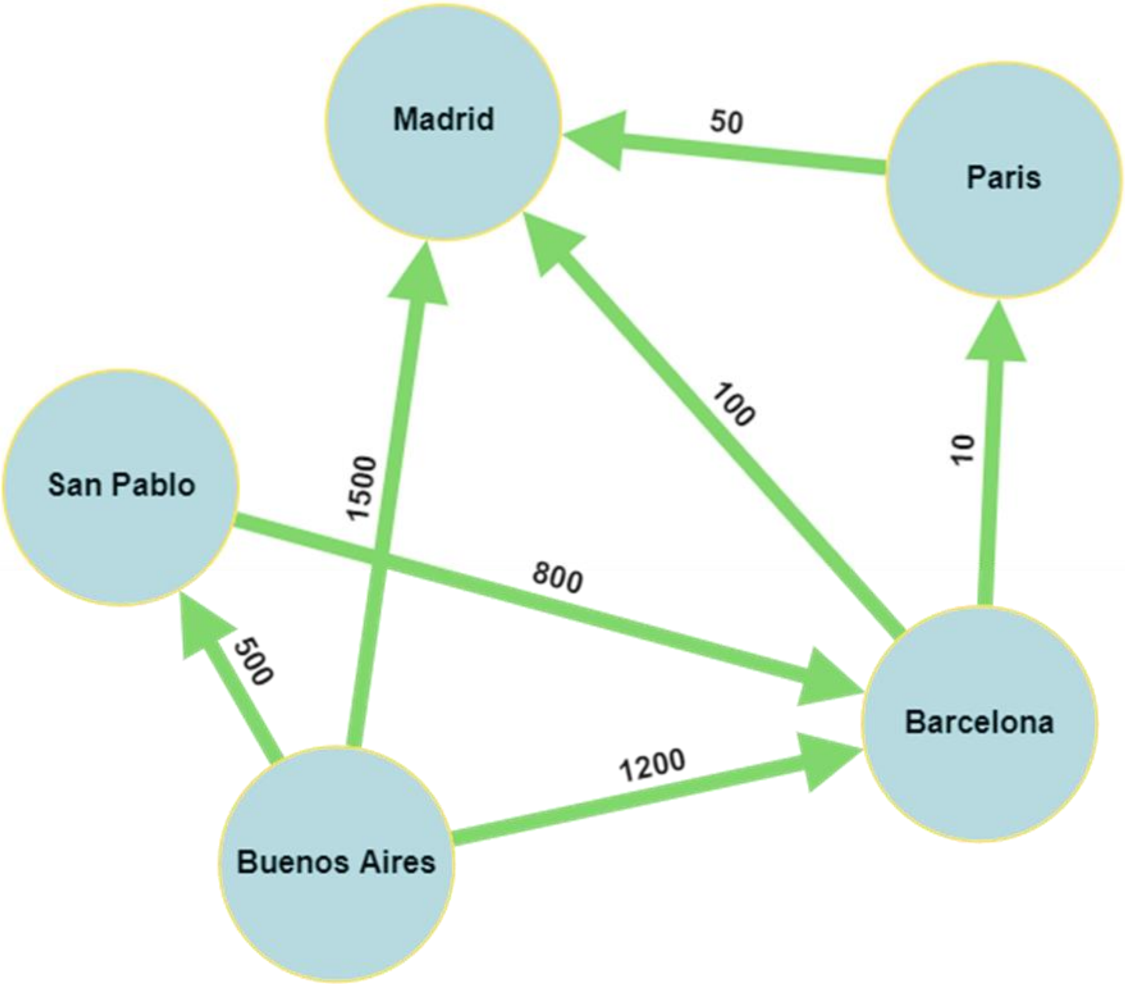


Enunciado del problema

Dados los costos de los vuelos entre las distintas ciudades consideradas.

¿Cuál es el camino más económico para ir de Buenos Aires a Madrid?

Podemos modelar el Grafo

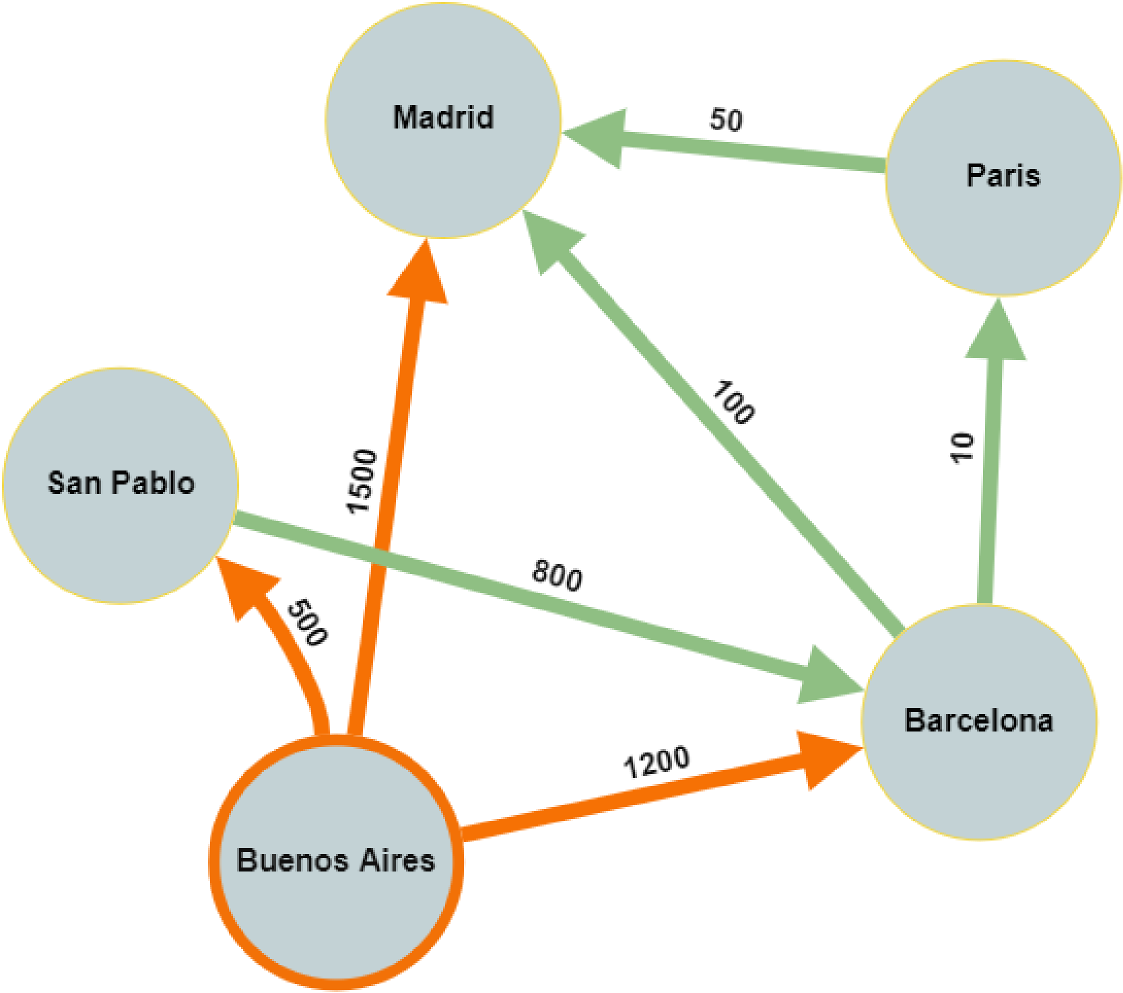


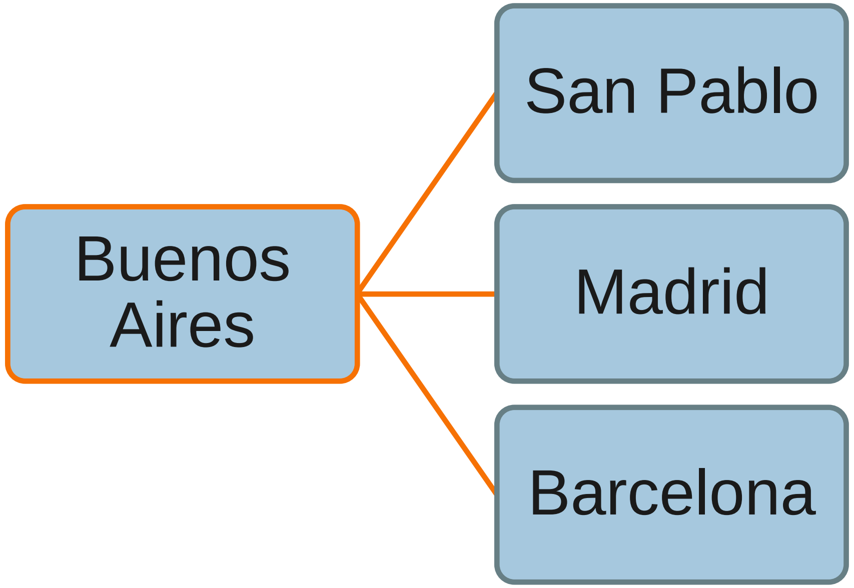
G (V,A) es el mapa de vuelos

V conjunto de vértices (Ciudades)
A conjunto de aristas (vuelos entre ciudades)

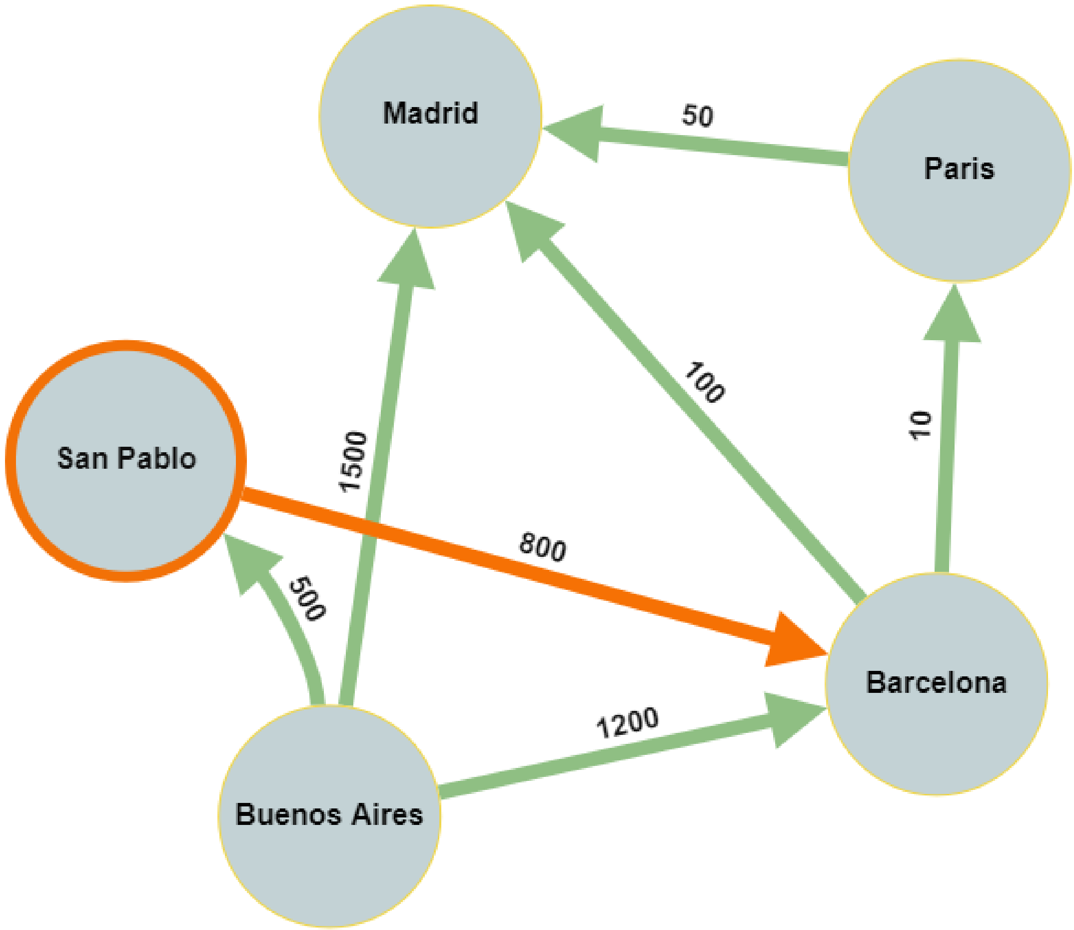
Peso de la arista es costo del vuelo

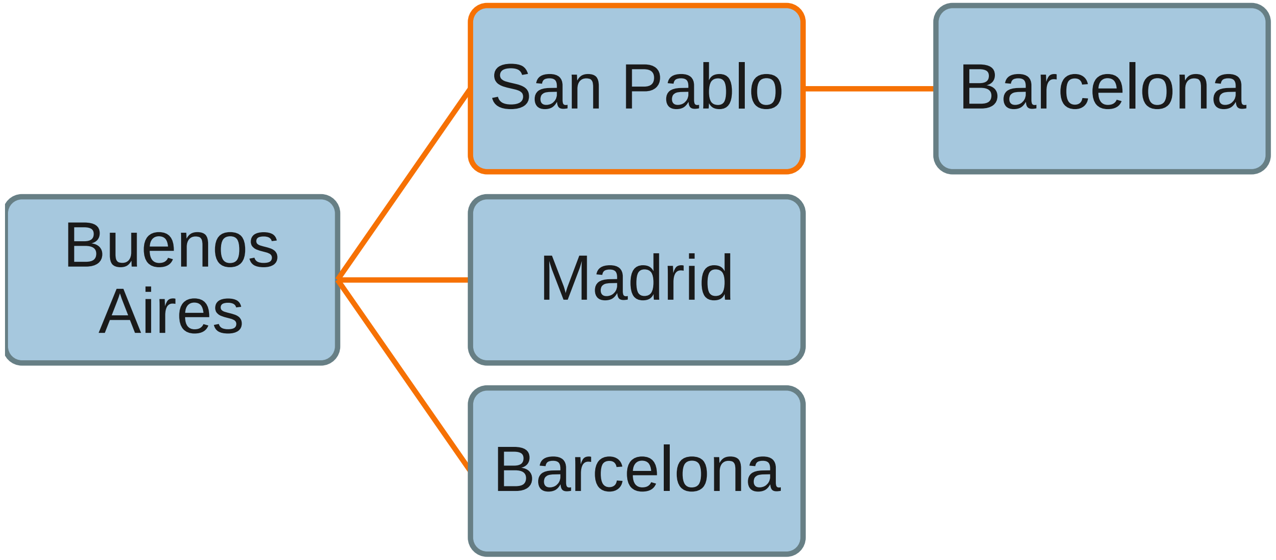
Una solución posible



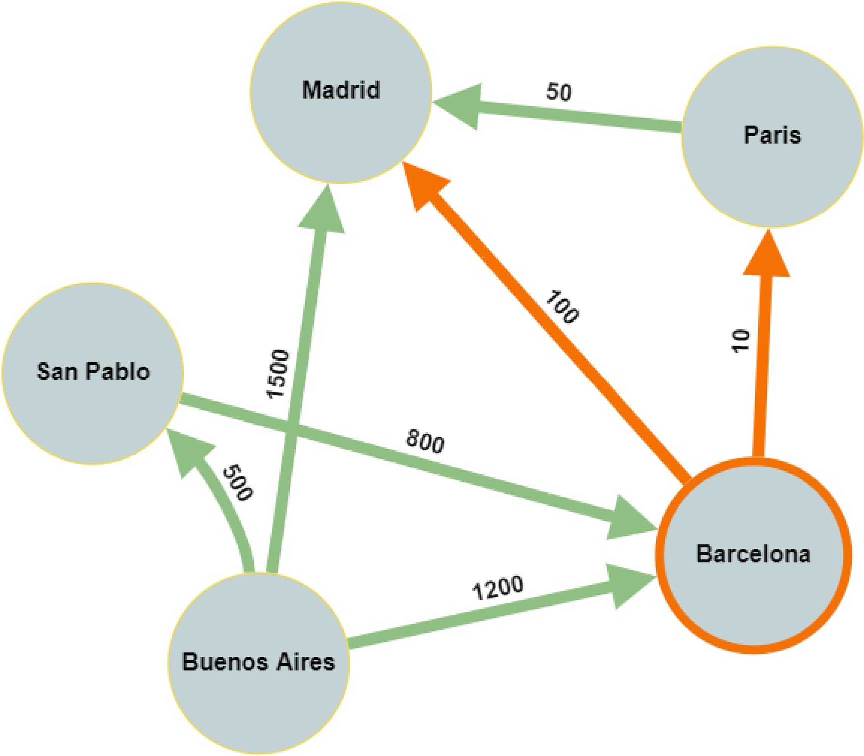


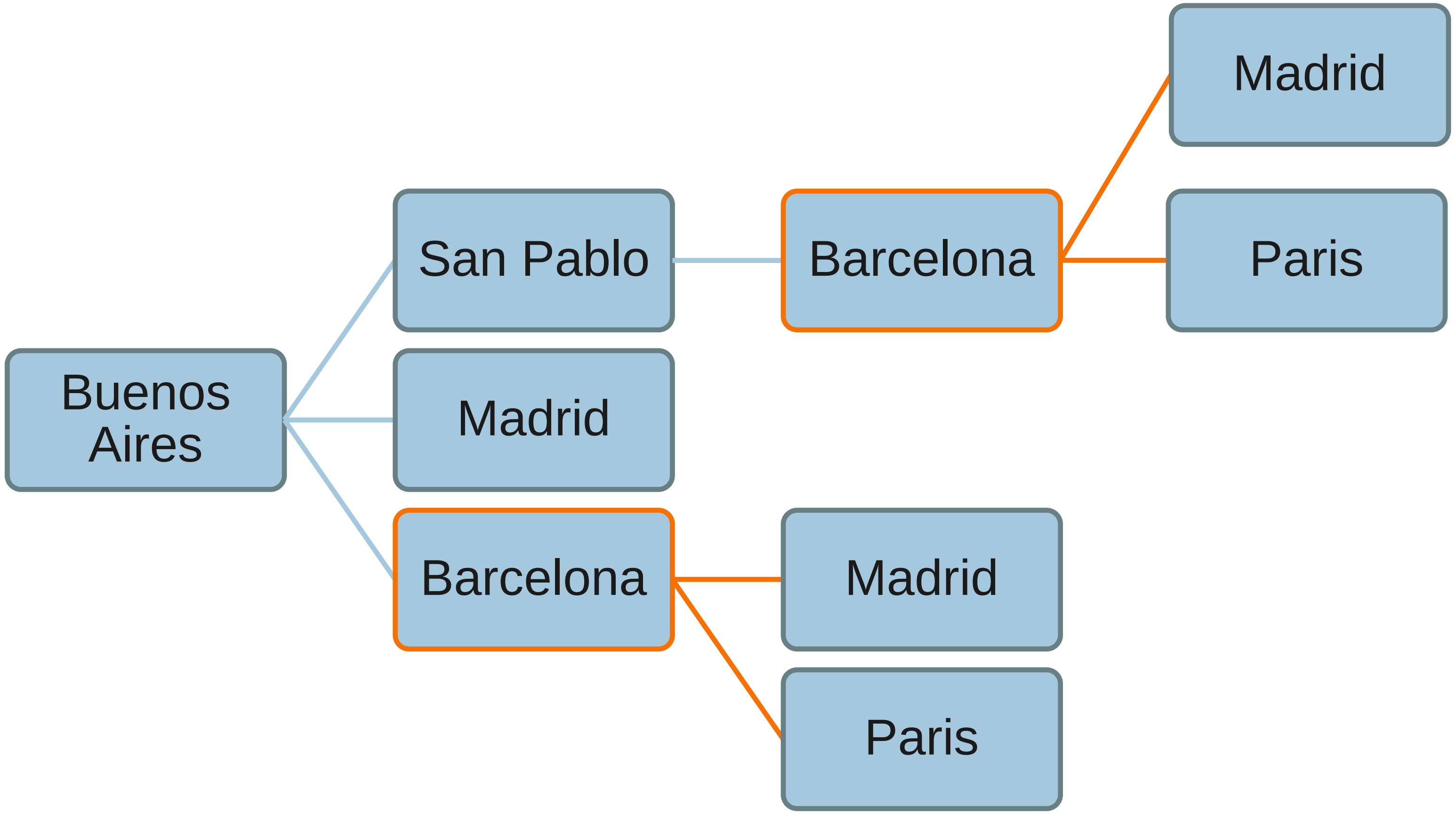
Una solución posible



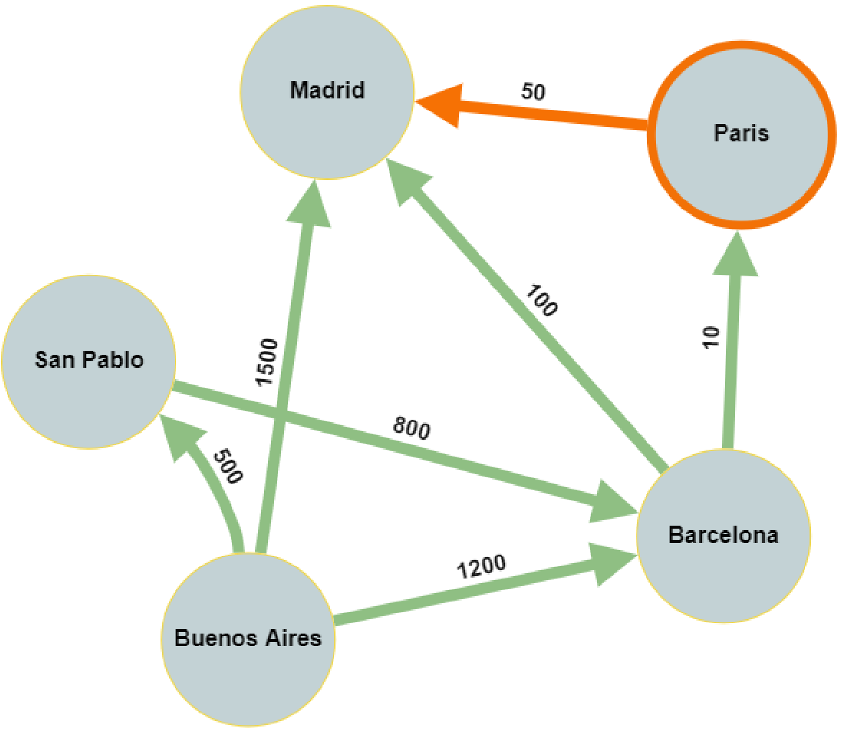


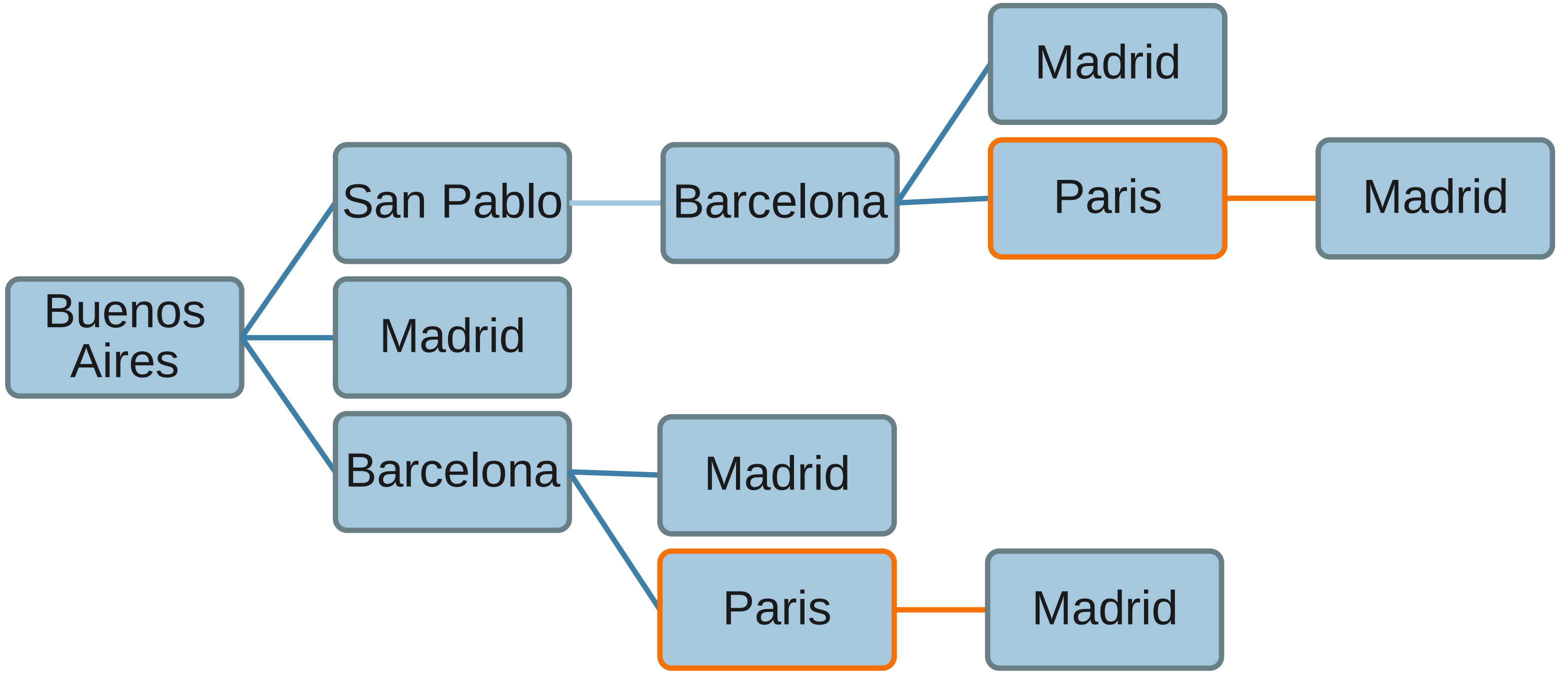
Una solución posible



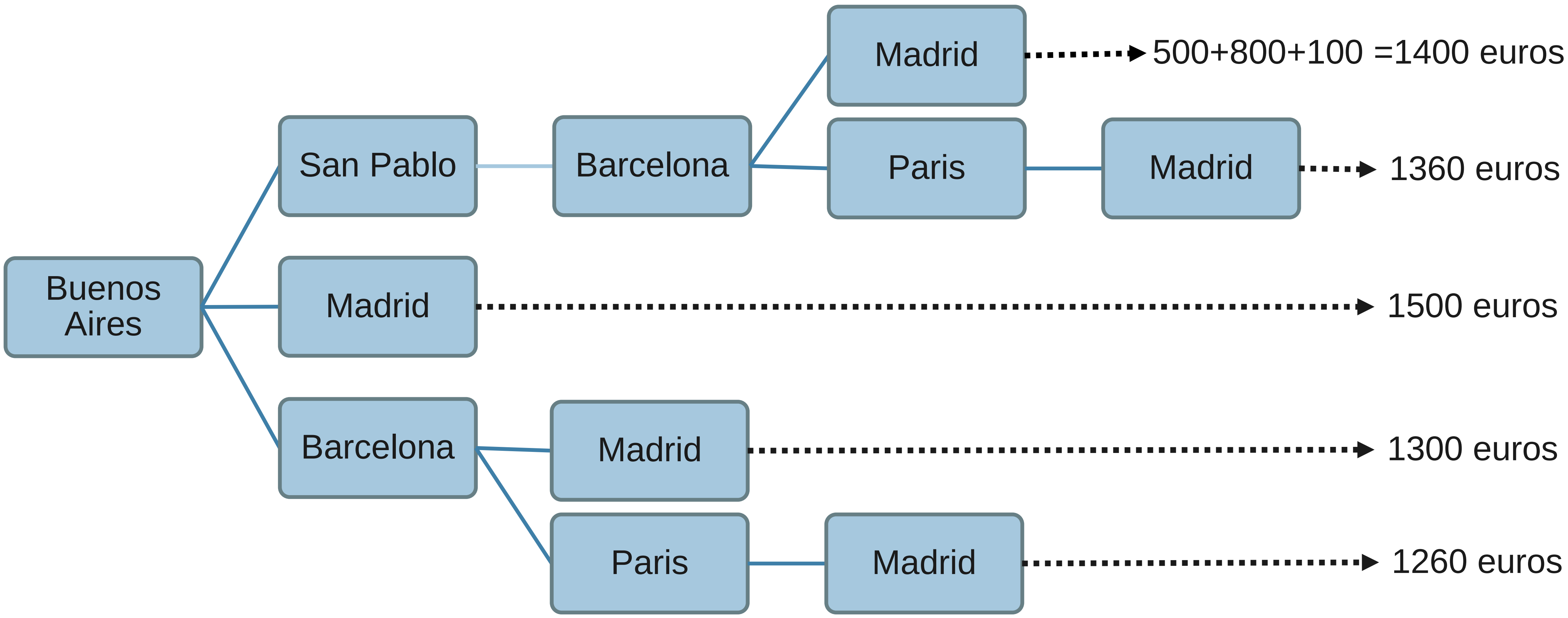


Una solución posible





Calculo de los costos de cada alternativa



Algoritmo de Dijkstra

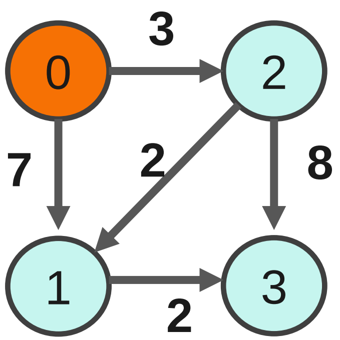
Este algoritmo fue creado por uno de los padres de la computación, Edger W. Dijkstra, en 1956.

El algoritmo de Dijkstra halla los caminos más cortos desde un nodo origen a todos los demás

- Sirve para grafo ponderados, dirigido o no
- Sus pesos son siempre positivos
- Es de una complejidad computacional más baja en comparación con el algoritmo anterior
- Es un algoritmo ávido

Algoritmo de Dijkstra - Ejemplo

Ejemplo:¿Cuál es el camino más corto de 0 a los demás vértices?



INPUT:

- Matriz de adyacencias
- Lista de nodos
- Nodo Inicial

OUTPUT:

- Vector de costos mínimos
- Vector de predecesores

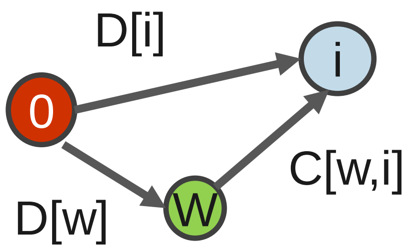
	0	1	2	3
D				

	0	1	2	3
P				

	0	**1**	**2**	**3**
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞

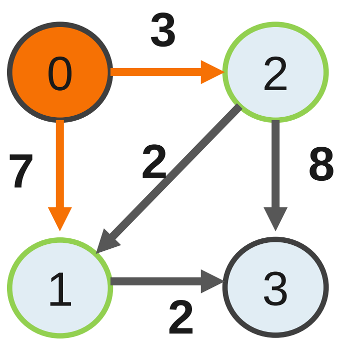
Sub	Rótulos
0	Buenos Aires
1	San Pablo
2	Barcelona
3	Madrid

Idea del algoritmo



Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



Paso Inicial

Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}	-	7/0	3/0	∞ /0

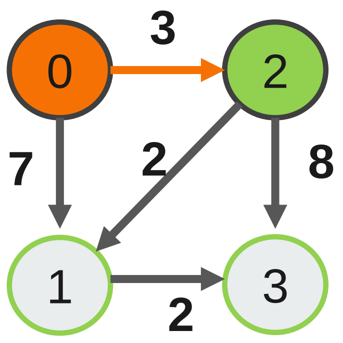
P[1]	P[2]	P[3]
0	0	0

V={0,1,2,3,4}

S conjunto de nodos cuya distancia más corta desde el origen es conocida

Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



Iteración 1-Paso 1

Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}	-	7/0	3/0	∞ /0
1	{0,2}	{1,3}	2		3/0	

Mejor camino de 0 al nodo 2

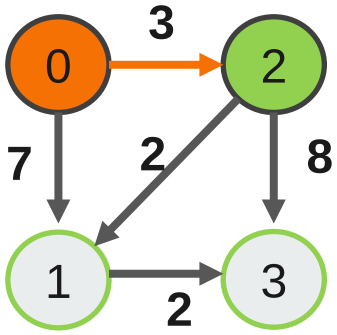
P[1]	P[2]	P[3]
0	0	0

El camino 0-2 no se puede mejorar pasando por 1

7 + ~~algo~~ < **3** => que D[2] =3 es mínimo (camino especial)

Algoritmo de Dijkstra - Ejemplo

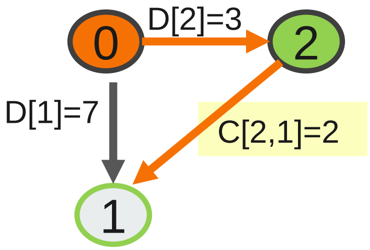
	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



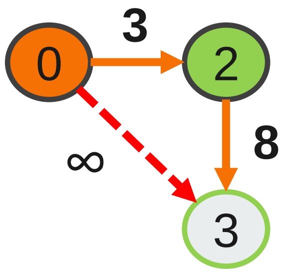
Iteración 1-Paso 2

Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}	-	7/0	3/0	∞ /0
1	{0,2}	{1,3}	2	?	3	?

W=2 Encontramos caminos mas cortos pasando por w



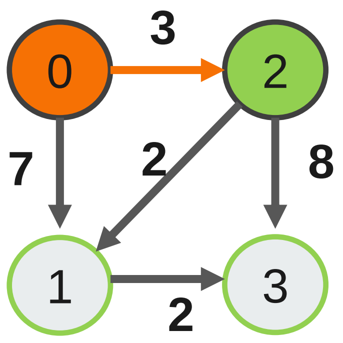
$$D[1]= \min \begin{cases} D[1]= 7 \\ D[2] + C[2,1]= 3+2=5 \end{cases}$$



$$D[3]= \min \begin{cases} D[3]= \infty \\ D[2] + C[2,3]= 3+8=11 \end{cases}$$

Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞

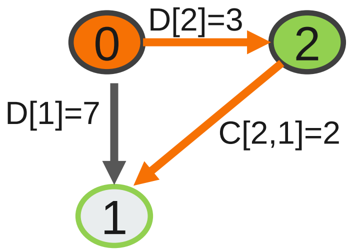


Iteración 1-Paso 2

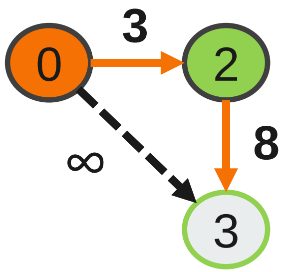
Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}	-	7/0	3/0	∞ /0
1	{0,2}	{1,3}	2	**5**/2	3	**11**/2

P[1]	P[2]	P[3]
2	0	**2**

W=2 Encontramos caminos mas cortos pasando por w



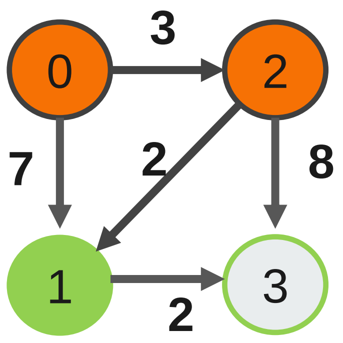
$$D[1] = \min \begin{cases} D[1] = 7 \\ D[2] + C[2,1] = 3+2=\mathbf{5} \end{cases}$$



$$D[3] = \min \begin{cases} D[3] = \infty \\ D[2] + C[2,3] = 3+8=\mathbf{11} \end{cases}$$

Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



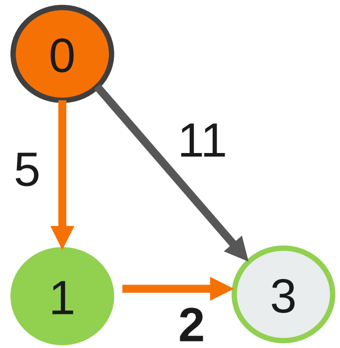
Iteración 2-Paso 1

Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}		7/0	3/0	∞ /0
1	{0,2}	{1,3}	2	5/2	3/0	11/2
2	{0,2,1}	{3}	1	5/2	3/0	**7**/1

P[1]	P[2]	P[3]
2	0	1

Iteración 2-Paso 2

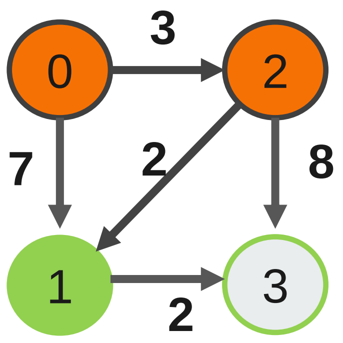
W=1 Encontramos caminos mas cortos pasando por w



$$D[3]= \min \begin{cases} D[3]= 11 \\ D[1] + C[1,3]= 5+2=\mathbf{7} \end{cases}$$

Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



Iteración 3-Paso 1

Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}		7/0	3/0	∞ /0
1	{0,2}	{1,3}	2	5/2	3/0	11/2
2	{0,2,1}	{3}	1	5/2	3/0	7/1
3	{0,2,1,3}	Ø	3	5/2	3/0	7/1

P[1]	P[2]	P[3]
2	0	1

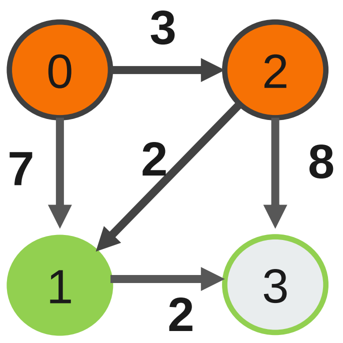
Iteración 3-Paso 2

W=3 Encontramos caminos mas cortos pasando por w

No hay nodos en V-S por lo tanto terminamos

Algoritmo de Dijkstra - Ejemplo

	0	1	2	3
0	∞	7	3	∞
1	∞	∞	∞	2
2	∞	2	∞	8
3	∞	∞	∞	∞



Iteración	S	V-S	W	D[1]	D[2]	D[3]
Inicial	{0}	{1,2,3}		7/0	3/0	∞ /0
1	{0,2}	{1,3}	2	5/2	3/0	11/2
2	{0,2,1}	{3}	1	5/2	3/0	7/1
3	{0,2,1,3}	Ø	3	5/2	3/0	7/1

P[1]	P[2]	P[3]
2	0	1

Solución

Caminos más cortos de 0 a…

– 1 (pasando por 2), costo 5
– 2 (directo desde 0) costo 3
– 3 (pasando por 2 y 1) costo 7

Camino más corto de 0 a 3
3-1-2-0

Algoritmo de Dijkstra – Complejidad Computacional

Paso Inicial:
S={0}   V-S={1,2,3}
Consideramos loa caminos directos desde el nodo inicial

O(n)

Mientras V-S ≠Ø

Paso 1:
Elegimos un w perteneciente a V-S tal que D[w] sea mínimo

O(n)

+

Paso 2:
Para cada nodo que pertenezca al conjunto V-S
calculamos si hay un mejor camino pasando por w
D[i]= min (D[i], D[w] + C(w,i)

O(n)

O(n)

O(n^2)

O(n^2)

Pseudocódigo de Dijkstra sin cola de prioridad

```
Dijkstra (Grafo G, nodo inicial s)
  visitado[n] = {false, ..., false} // guarda si un nodo ya fue visitado
  distancia[n] = {Infinito, ..., Infinito} // guarda las distancias del nodo
        salida al resto

  para cada w en V[G] hacer
     si existe arista entre s y w entonces
         distancia[w] = peso (s, w)

  distancia[s] = 0
  visitado[s] = true

  mientras que no esten visitados todos hacer
     v = nodo de menor distancia a s que no fue visitado aun
     visitado[v] = true
     para cada w en sucesores (G, v) hacer
         si distancia[w] > distancia[v] + peso (v, w) entonces
            distancia[w] = distancia[v] + peso (v, w)
            padre[w] = v
```

Pseudocódigo de Dijkstra con cola de prioridad

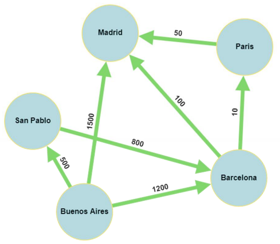
```
Dijkstra (Grafo G, nodo_fuente s)
   para todo u en V[G] hacer
       distancia[u] = INFINITO
       padre[u] = NULL
       visitado[u] = false

   distancia[s] = 0
   adicionar (cola, (s, distancia[s]))

   mientras que cola no sea vacia hacer
       u = extraer_minimo(cola)
       visitado[u] = true
       para todo v en adyacencia[u] hacer
           si no visitado[v] y distancia[v] > distancia[u] + peso (u, v)
               hacer
              distancia[v] = distancia[u] + peso (u, v)
              padre[v] = u
              adicionar(cola,(v, distancia[v]))
```

¿Qué implementación uso?

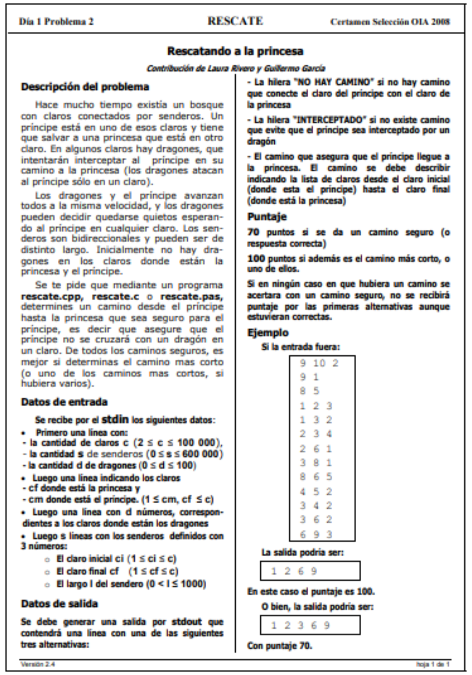
- Si el grafo es **ralo**, o sea, tiene pocas aristas, conviene utilizarla implementación con cola de prioridad ($O(m \log n)$)

- Si el grafo es **denso**, o sea, tiene muchas aristas, convieneutilizar la implementación básica ($O(n^2)$)

Ejercicios tipo de la práctica



- Resolver el ejercicio del viaje más económico a Madrid por el algoritmo de Dijkstra
- Resolver el ejercicio de la OIA “Rescatando a la Princesa”
- Investigar como se puede aplicar programación dinámica a este problema



Día 1 Problema 2 — RESCATE — Certamen Selección OIA 2008

Rescatando a la princesa

Contribución de Laura Rivero y Guillermo García

Descripción del problema

Hace mucho tiempo existía un bosque con claros conectados por senderos. Un príncipe está en uno de esos claros y tiene que salvar a una princesa que está en otro claro. En algunos claros hay dragones, que intentarán interceptar al príncipe en su camino a la princesa (los dragones atacan al príncipe sólo en un claro).

Los dragones y el príncipe avanzan todos a la misma velocidad, y los dragones pueden decidir quedarse quietos esperando al príncipe en cualquier claro. Los senderos son bidireccionales y pueden ser de distinto largo. Inicialmente no hay dragones en los claros donde están la princesa y el príncipe.

Se te pide que mediante un programa **rescate.cpp, rescate.c** o **rescate.pas,** determines un camino desde el príncipe hasta la princesa que sea seguro para el príncipe, es decir que asegure que el príncipe no se cruzará con un dragón en un claro. De todos los caminos seguros, es mejor si determinas el camino mas corto (o uno de los caminos mas cortos, si hubiera varios).

Datos de entrada

Se recibe por el **stdin** los siguientes datos:

- Primero una línea con:
  - la cantidad de claros c ($2 \le c \le 100\,000$),
  - la cantidad s de senderos ($0 \le s \le 600\,000$)
  - la cantidad d de dragones ($0 \le d \le 100$)
- Luego una línea indicando los claros
  - cf donde está la princesa y
  - cm donde está el príncipe. ($1 \le cm, cf \le c$)
- Luego una línea con d números, correspondientes a los claros donde están los dragones
- Luego s líneas con los senderos definidos con 3 números:
  - El claro inicial ci ($1 \le ci \le c$)
  - El claro final cf ($1 \le cf \le c$)
  - El largo l del sendero ($0 < l \le 1000$)

Datos de salida

Se debe generar una salida por **stdout** que contendrá una línea con una de las siguientes tres alternativas:

- La hilera "NO HAY CAMINO" si no hay camino que conecte el claro del príncipe con el claro de la princesa
- La hilera "INTERCEPTADO" si no existe camino que evite que el príncipe sea interceptado por un dragón
- El camino que asegura que el príncipe llegue a la princesa. El camino se debe describir indicando la lista de claros desde el claro inicial (donde esta el príncipe) hasta el claro final (donde está la princesa)

Puntaje

70 puntos si se da un camino seguro (o respuesta correcta)

100 puntos si además es el camino más corto, o uno de ellos.

Si en ningún caso en que hubiera un camino se acertara con un camino seguro, no se recibirá puntaje por las primeras alternativas aunque estuvieran correctas.

Ejemplo

Si la entrada fuera:

```
9 10 2
9 1
8 5
1 2 3
1 3 2
2 3 4
2 6 1
3 8 1
8 6 5
4 5 2
3 4 2
3 6 2
6 9 3
```

La salida podría ser:

```
1 2 6 9
```

En este caso el puntaje es 100.

O bien, la salida podría ser:

```
1 2 3 6 9
```

Con puntaje 70.

Versión 2.4 — hoja 1 de 1

¡Gracias!