Ayudantía 10

Hola <3

Temario

1. Repaso de pauta T2
2. BFS
3. Dijkstra
4. Si sobra tiempo podemos pasar al Discord a ayudar con la tarea

Repaso de Pauta T2

BFS

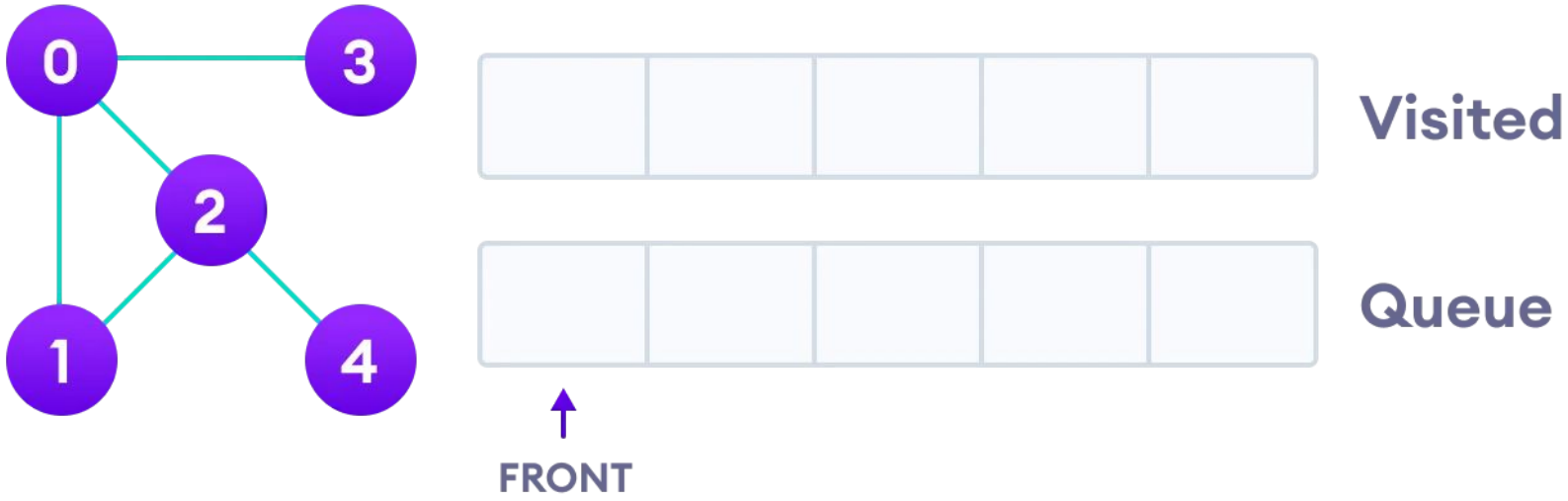
Primero, veamos que es un queue

- Sigue la regla: **FIRST IN, FIRST OUT**
- Es decir, lo primero que tenemos en un arreglo es lo primero que sacamos.
- Pensemos en una cola.

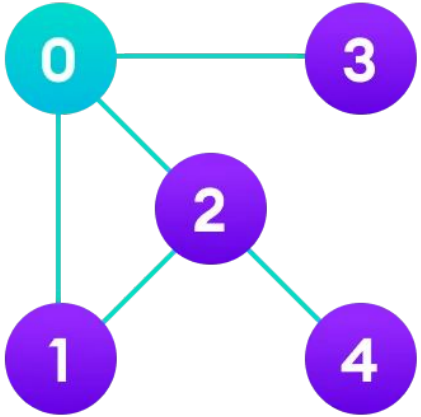
Algoritmo

1. Empezamos eligiendo cualquiera de los nodos del grafo y lo ponemos en la queue.
2. Sacamos el **primer** elemento del queue y lo agregamos a la lista de nodos visitados.
3. Creamos una lista de nodos adyacentes y agregamos los nodos no visitados en la parte posterior del queue.
4. Seguimos repitiendo los pasos 2 y 3 hasta que el queue quede vacío.

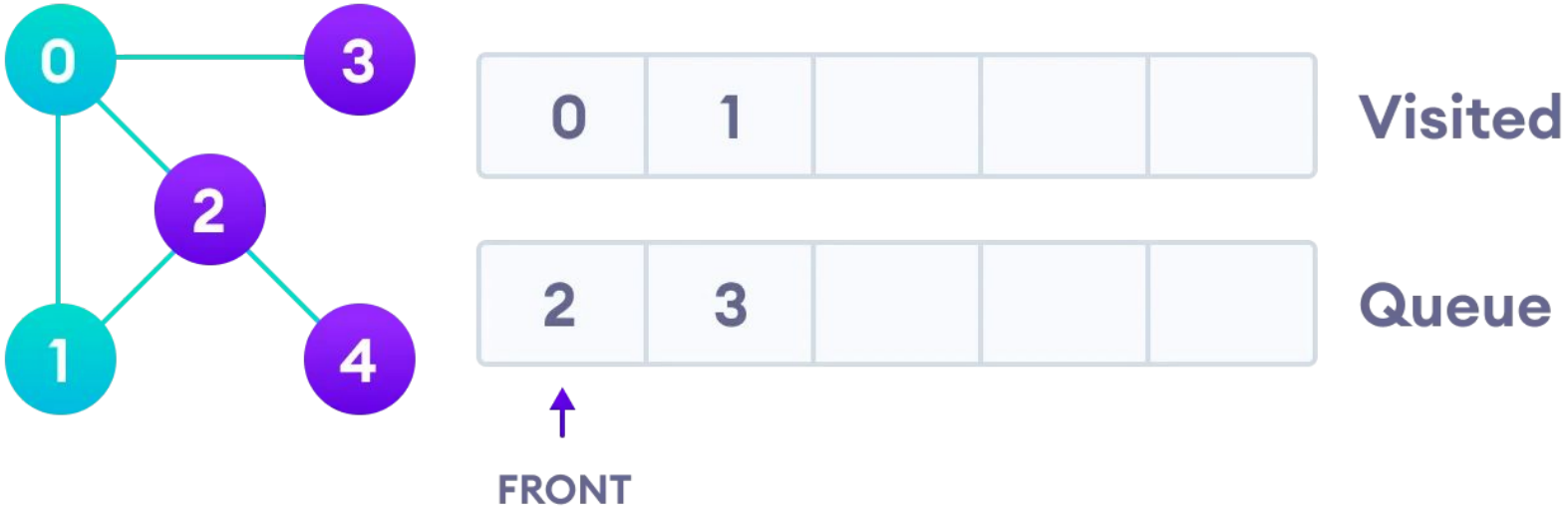
Ejemplo



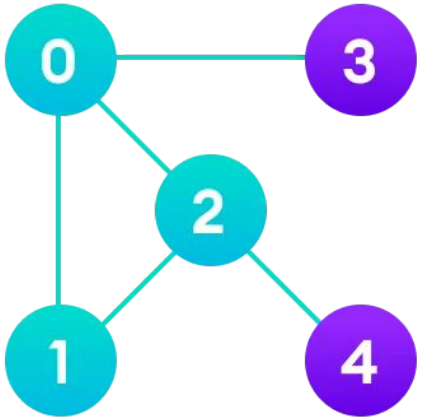
Ejemplo

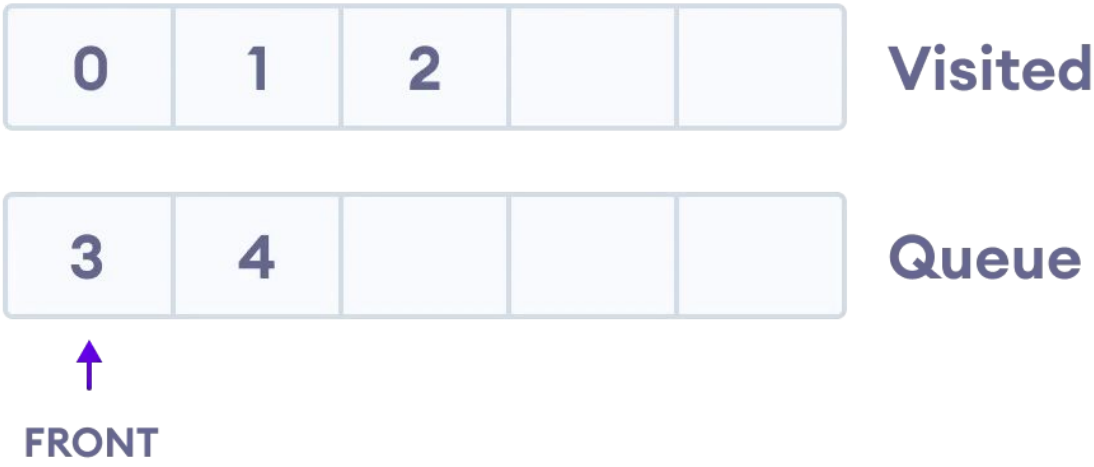


Ejemplo



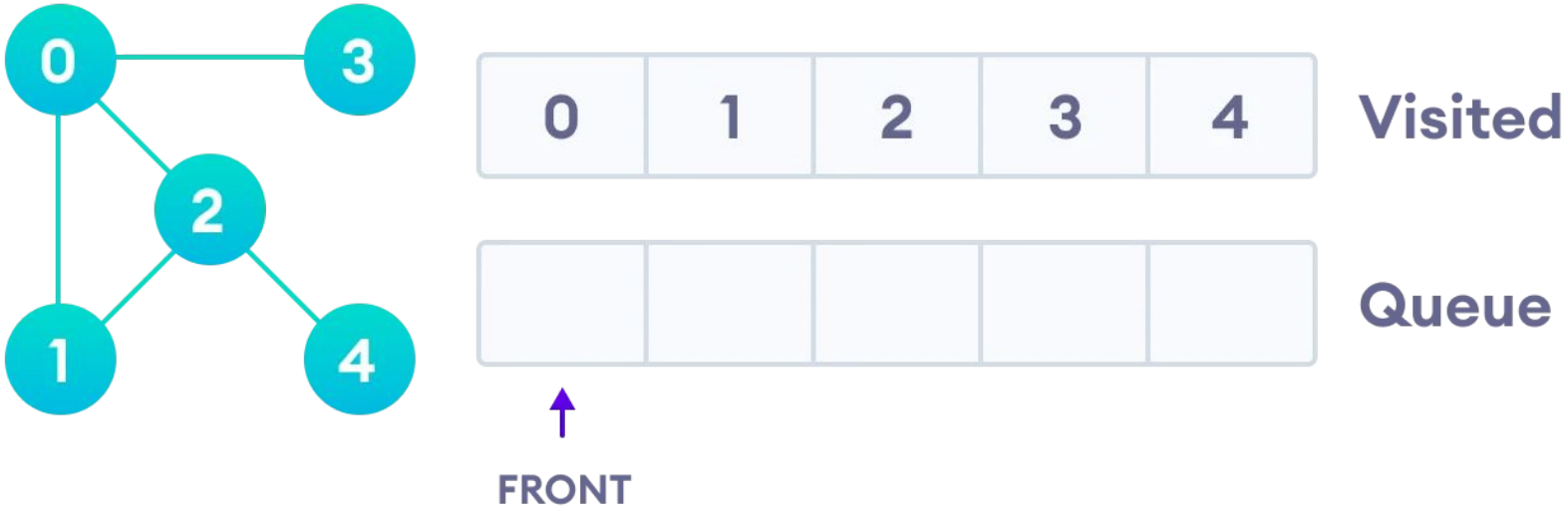
Ejemplo





Ejemplo

Ejemplo



Dijkstra

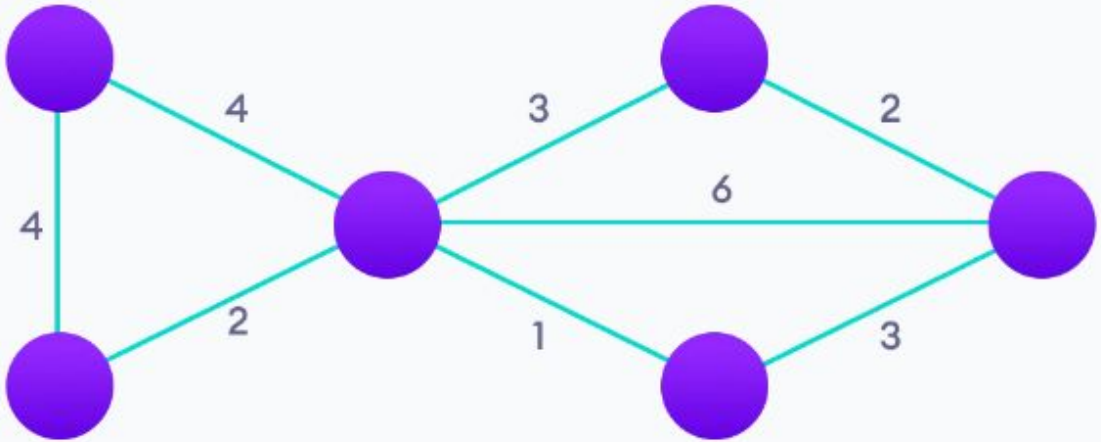
Algoritmo

```
function Dijkstra(Graph, source):
    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0
```

Algoritmo

```
while Q is not empty:
    u ← vertex in Q with min dist[u]
    remove u from Q
    for each neighbor v of u still in Q:
        alt ← dist[u] + Graph.Edges(u, v)
        if alt < dist[v]:
            dist[v] ← alt
            prev[v] ← u
return dist, prev
```

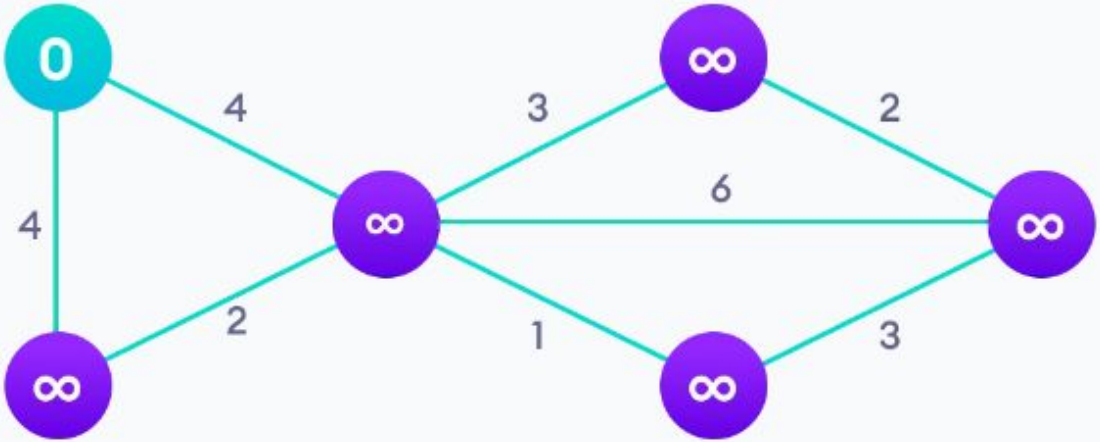
Ejemplo



Step: 1

Partimos con un grafo ponderado

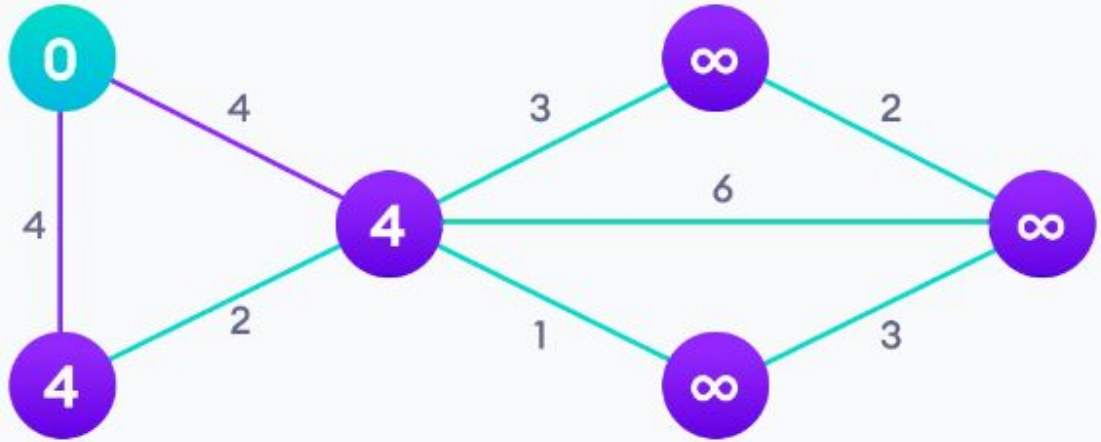
Ejemplo



Step: 2

Elige un nodo inicial y asigna valores de ruta a infinito en todos los vértices

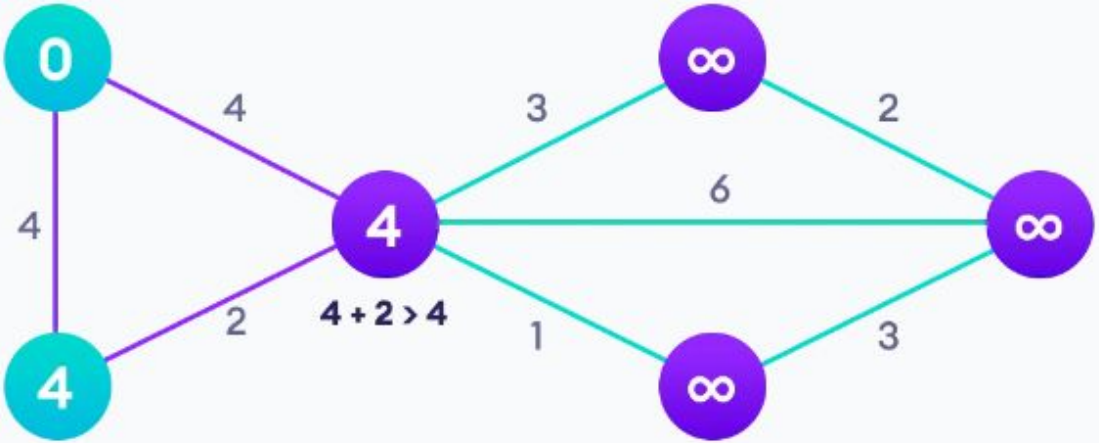
Ejemplo



Step: 3

En cada vértice actualizamos el costo de la ruta

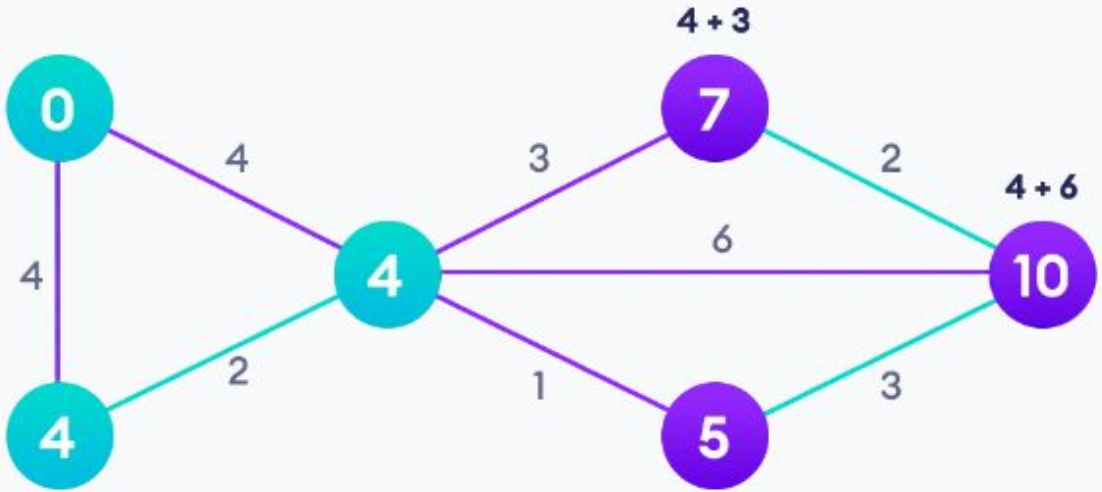
Ejemplo



Step: 4

Si el costo de la ruta al vértice adyacente es menor que la ruta nueva, no la actualices el costo.

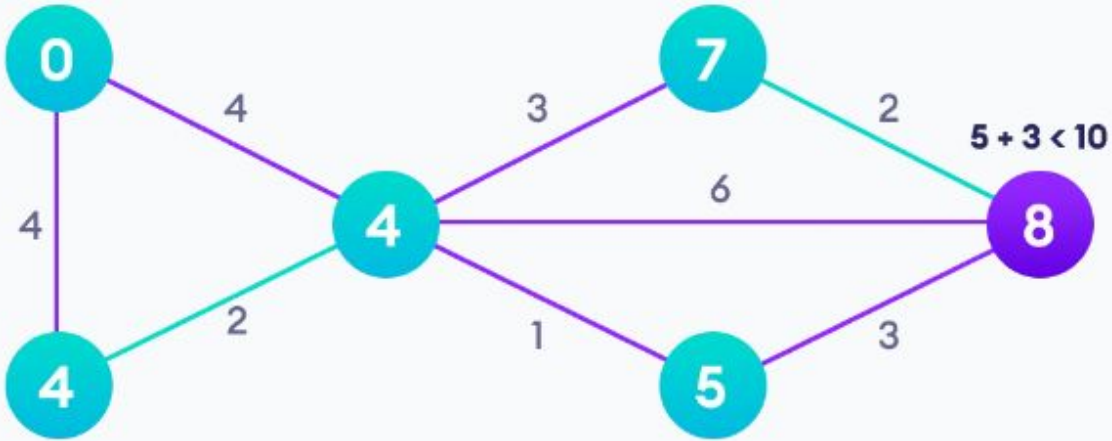
Ejemplo



Step: 5

Evitamos actualizar los costos de las ruta de los vértices ya visitados

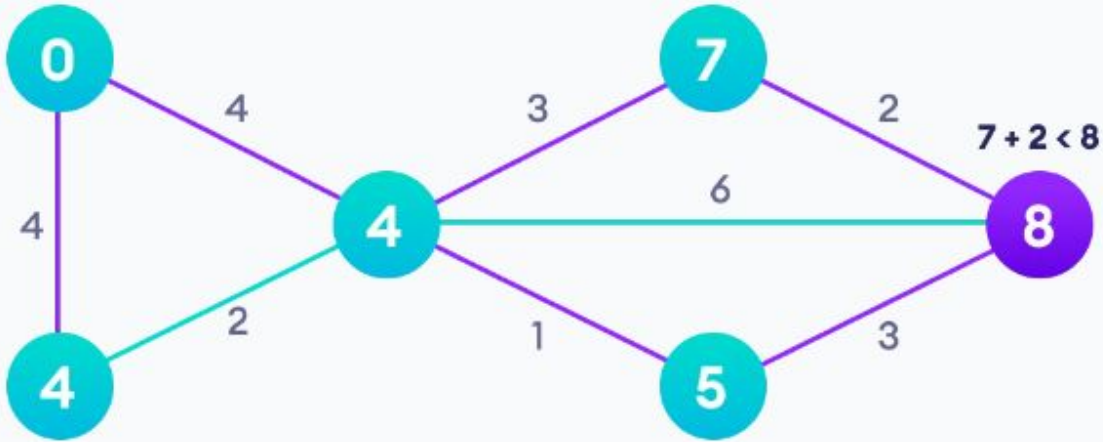
Ejemplo



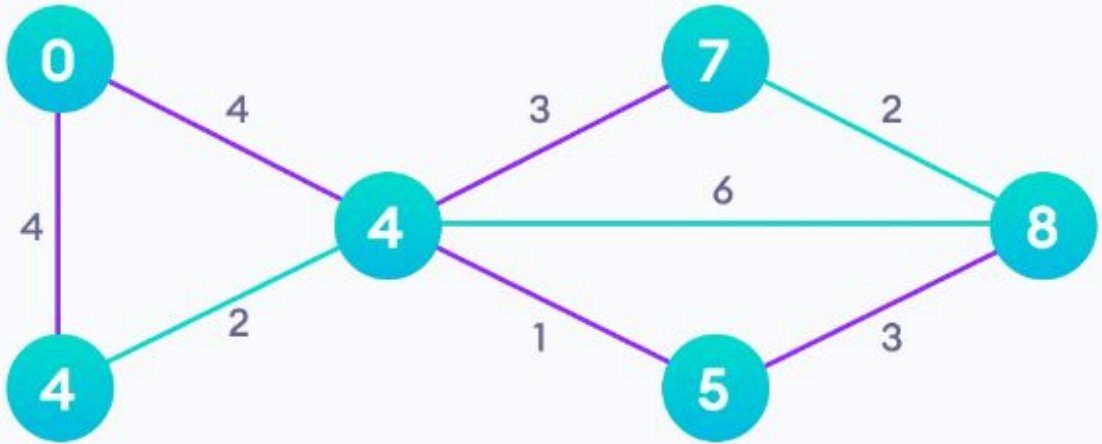
Step: 6

Después de cada interacción, elegimos el vértice no visitado con el menor costo. Por ende elegimos el 5 antes que el 7

Ejemplo



Ejemplo

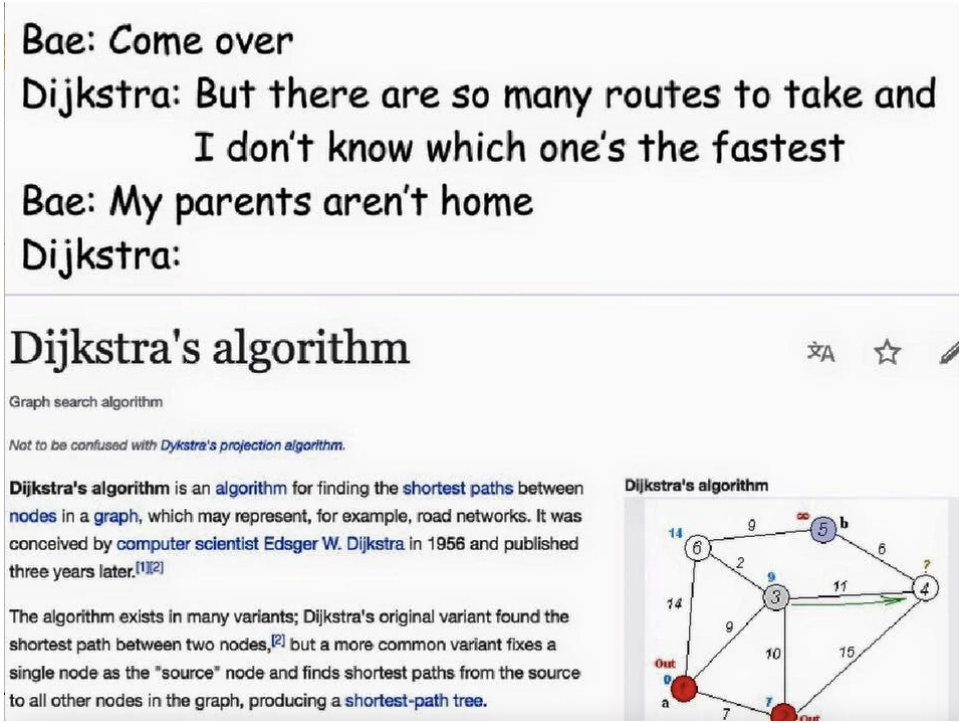


Step: 8

Repetimos los pasos hasta que visitemos todos los nodos

¿Para qué sirve?

1. Dijkstra encuentra el **primer** camino de menor costo entre nodos de un grafo.
2. BFS, pero más BKN.
3. Algoritmo codicioso, reduce costo local para reducir costo global.
4. ¿Complejidad? O(V log(V))



Ejemplo - I3 2022-1

1. Las alternativas para desplazarse en la ciudad son múltiples: metro, buses, bicicletas, servicios de plataformas, caminando, etc. La elección de los medios de transporte más adecuados para ir de A a B depende del tiempo necesario para realizar el viaje en el medio elegido y el costo incurrido en el viaje.
Un mismo viaje puede involucrar más de un medio de transporte en diferentes tramos (A-metro-C-bus-D-caminar-B)).

El tiempo de desplazamiento y el costo involucrado para moverse dentro de la ciudad en cada alternativa de transporte está disponible en archivos como el siguiente para cada medio de transporte. Cada celda contiene el tempo necesario y el costo para ir del origen al destino en el medio de transporte correspondiente (las celdas vacías indican que no es posible ir en ese medio desde ese origen al destino)

Ejemplo - I3 2022-1

Metro					
Origen/Destino	**A**	**B**	**C**	**D**	**E**
A		10 | 22			25 | 42
B	15 | 22		56 | 44	60 | 100	
C					70 | 150
D	34 | 55	60 | 100			
E		56 | 78			

Ejemplo - I3 2022-1

a) Escriba en pseudocódigo un algoritmo que permita determinar la mejor ruta para ir de un lugar a otro de la ciudad, registrando los múltiples medios de transporte utilizados, el tiempo y costo total del viaje. [HINT: considere que el tiempo es oro].

Ejemplo - I3 2022-1

Algorithm 1 Dijkstra(s, V, α)

```
d ← Arreglo vacío del largo de V          // Tiempo de llegada a cada nodo
π ← Arreglo vacío del largo de V          // Arista por la cual se llegó al nodo
for u ∈ V do
    u.color ← white
    d[u] ← ∞
    π[u] ← null
end for
Q ← cola de prioridades
s.color ← gray
d[s] ← 0
Q.enqueue(s)
while !Q.empty() do
    u ← Q.dequeue()
    for e ∈ α[u] do
        v ← e.end
        if v.color == white or v.color == grey then
            if d[v] > d[u] + e.time then
                d[v] ← d[u] + e.time
                π[v] ← e
            end if
            if v.color == white then
                v.color ← grey
                Q.enqueue(v)
            end if
        end if
    end for
    u.color ← black
end while
return d, π
```

Muchas gracias <3