

EDSGER DIJKSTRA

DIJKSTRA **ALGORITMO**

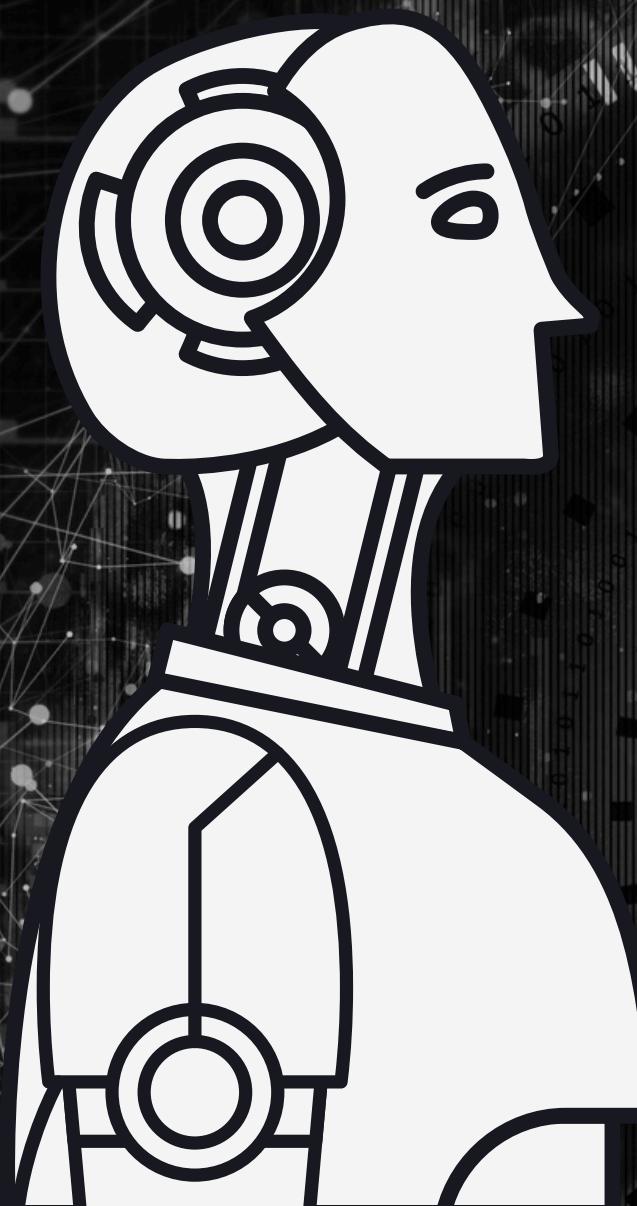
Camino más corto de Dijkstra

David Brizuela

¿QUE ES EL ALGORITMO DE DIJKSTRA?

Desarrollado por el científico informático holandés Edsger W. Dijkstra en 1956. El algoritmo de Dijkstra es un conjunto de instrucciones finitas, no ambiguas que se ejecutan de manera ordenada para poder solucionar problemas de ruta más corta. Básicamente encuentra la ruta más corta entre dos vértices en un grafo.

¿PARA QUE SIRVE ESTE ALGORITMO?



La aplicación más común es para los protocolos de enrutamiento para redes informáticas. Además se utiliza para los sistemas de mapas para encontrar el camino más corto entre el punto de partida y el destino (Google Maps).

El algoritmo Dijkstra puede funcionar tanto en grafos dirigidos como en grafos no dirigidos, la única condición que se debe cumplir para que este algoritmo funcione es la de tener pesos de aristas no negativos y estar conectados.

GLOSARIO

Grafos: Conjunto de objetos llamados nodos unido por enlaces llamados aristas. Permiten representar relaciones binarias entre elementos de un conjunto (fig. 1.1).

Grafo Dirigido: Cada arista tiene una dirección de viaje entre los nodos conectados por dicha arista (fig. 1.2).

Grafo no Dirigido: Las aristas no tienen una dirección, por lo tanto el algoritmo se puede desplazar atrás o adelante (fig. 1.3).

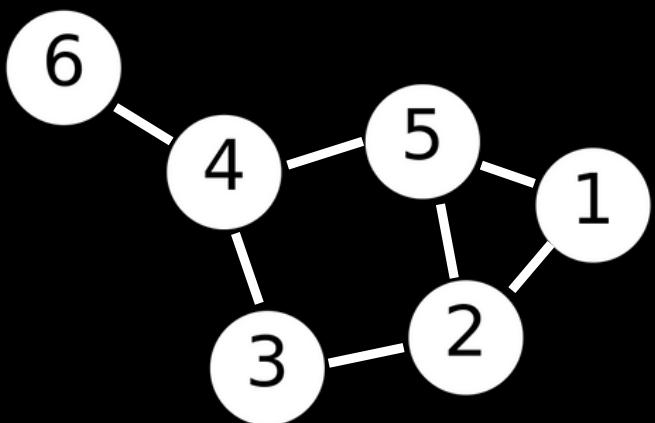


Fig. 1.1. Grafo etiquetado con 6 vértices y 7 aristas.

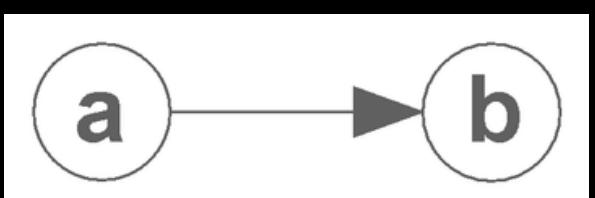


Fig. 1.2. Grafo dirigido.

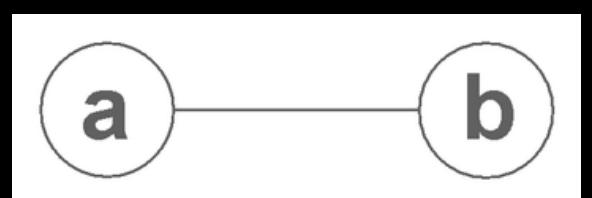


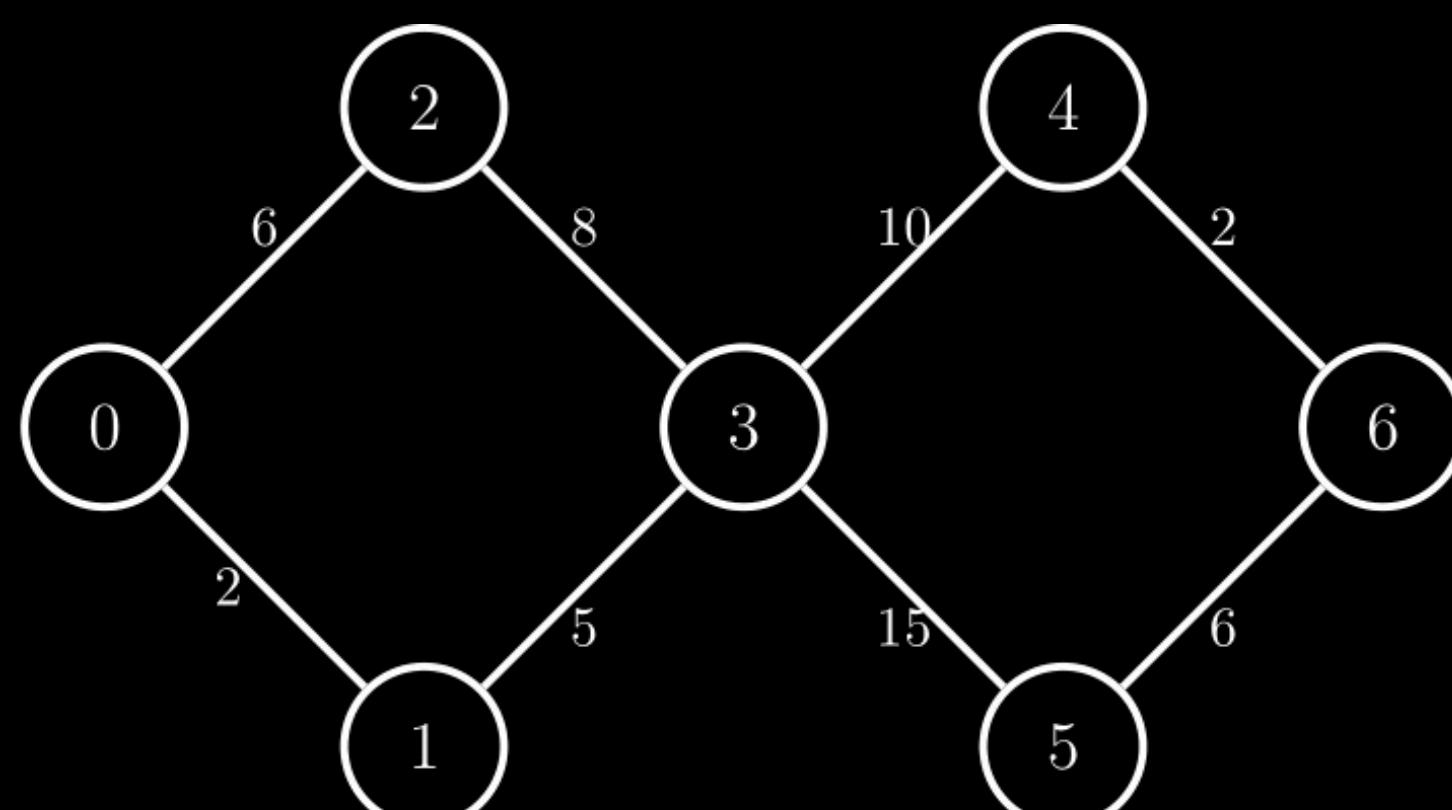
Fig. 1.3. Grafo no dirigido.

ALGORITMO DE DIJKSTRA

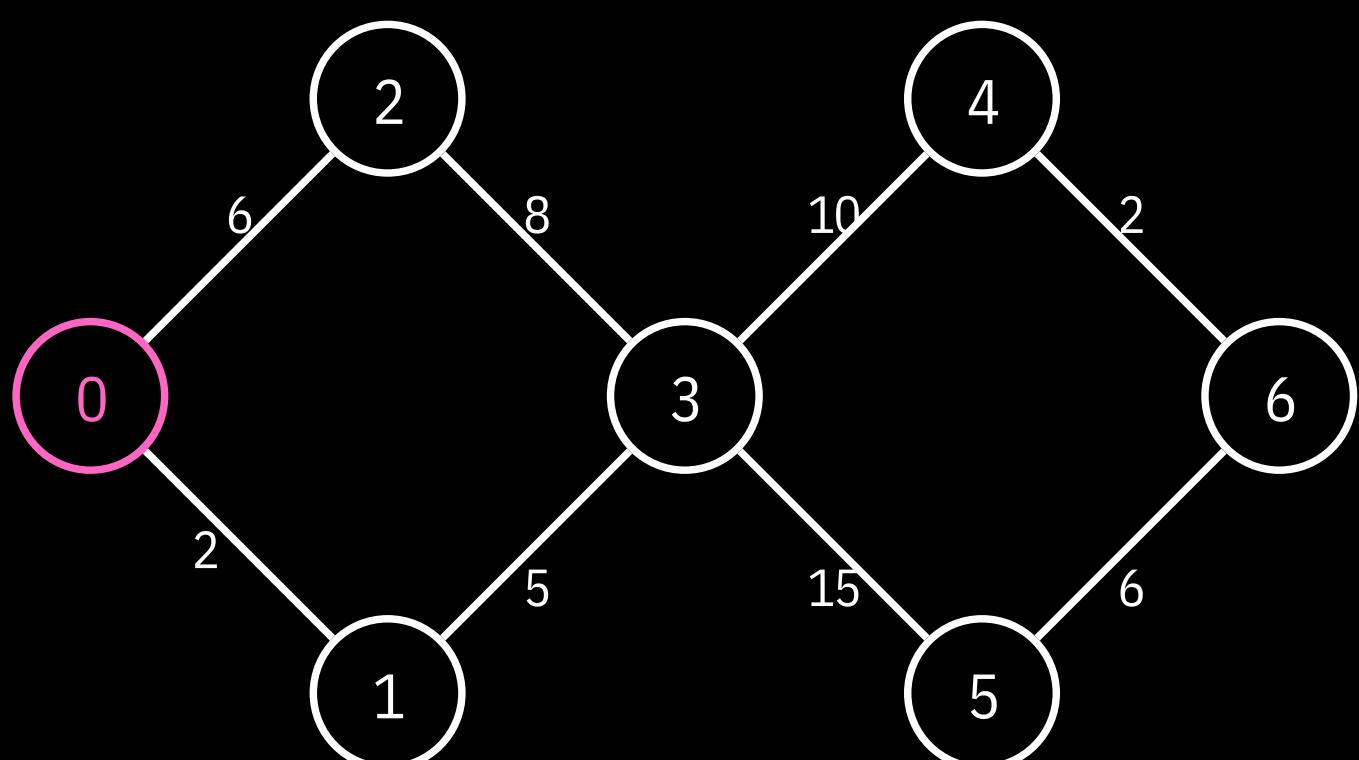
1. Marque el nodo de origen con una distancia **actual de 0** y el resto con **infinito**.
2. Encuentre el nodo no visitado con la distancia **más pequeña**.
3. Para cada vecino, sumará la distancia actual del nodo adyacente con el peso del borde que los conecta. Si es **menor que la distancia actual del nodo**, establezcalo como la **nueva distancia actual de N**.
4. Marque el nodo actual 1 como visitado.
5. Vaya al paso 2 si hay nodos sin visitar.

¿COMO FUNCIONA EL ALGORITMO DE DIJKSTRA?

Considere el siguiente grafo:



1. Comience desde el nodo 0 y marque el nodo como visitado, como puede verificar en la imagen, el nodo visitado está marcado con rosa:



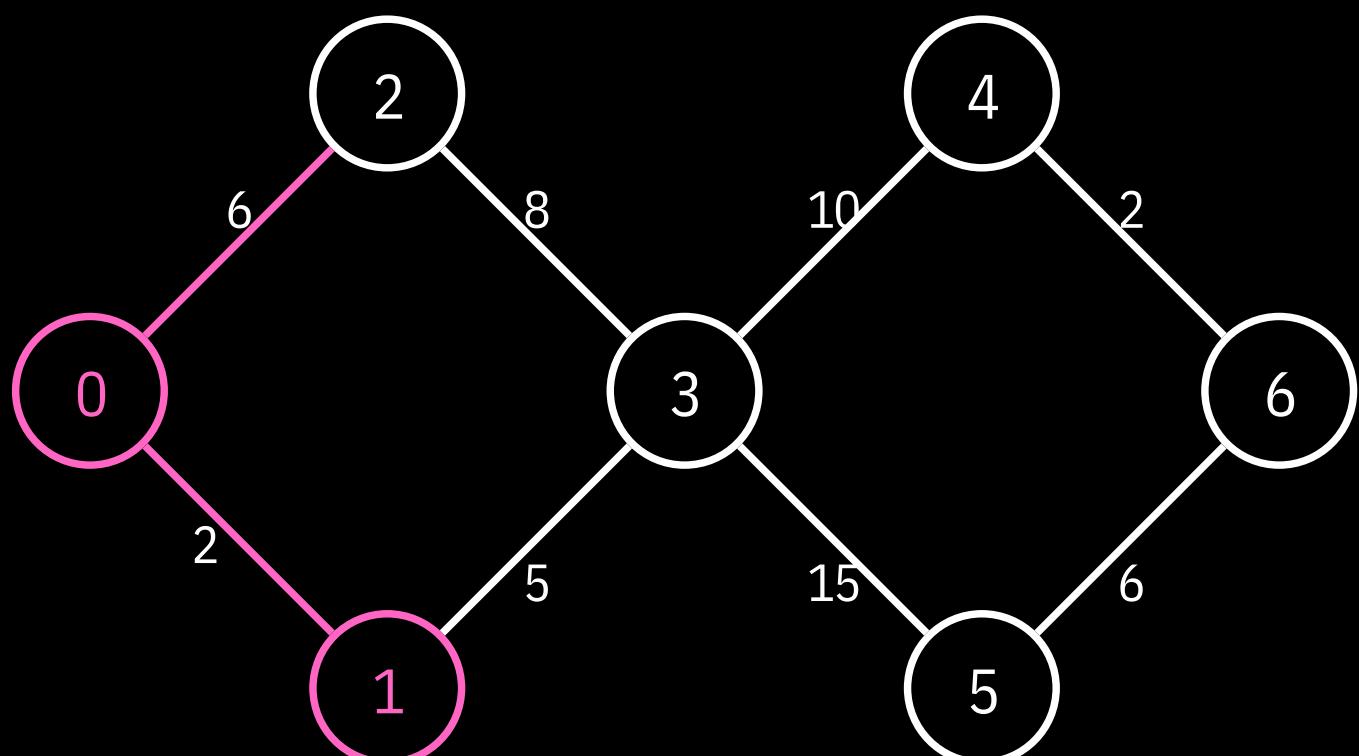
Nodos sin visitar
 $\{0, 1, 2, 3, 4, 5, 6\}$

Distancia:

0	0 ✓
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞

1. Distancia: Nodo 0 \rightarrow Nodo 0 = 0

2. Visitamos los nodos vecinos y elegimos el de menor distancia (Nodo 1). En este paso, el nodo 1 es la distancia mínima al nodo adyacente, así que márquelo como visitado y sume la distancia:



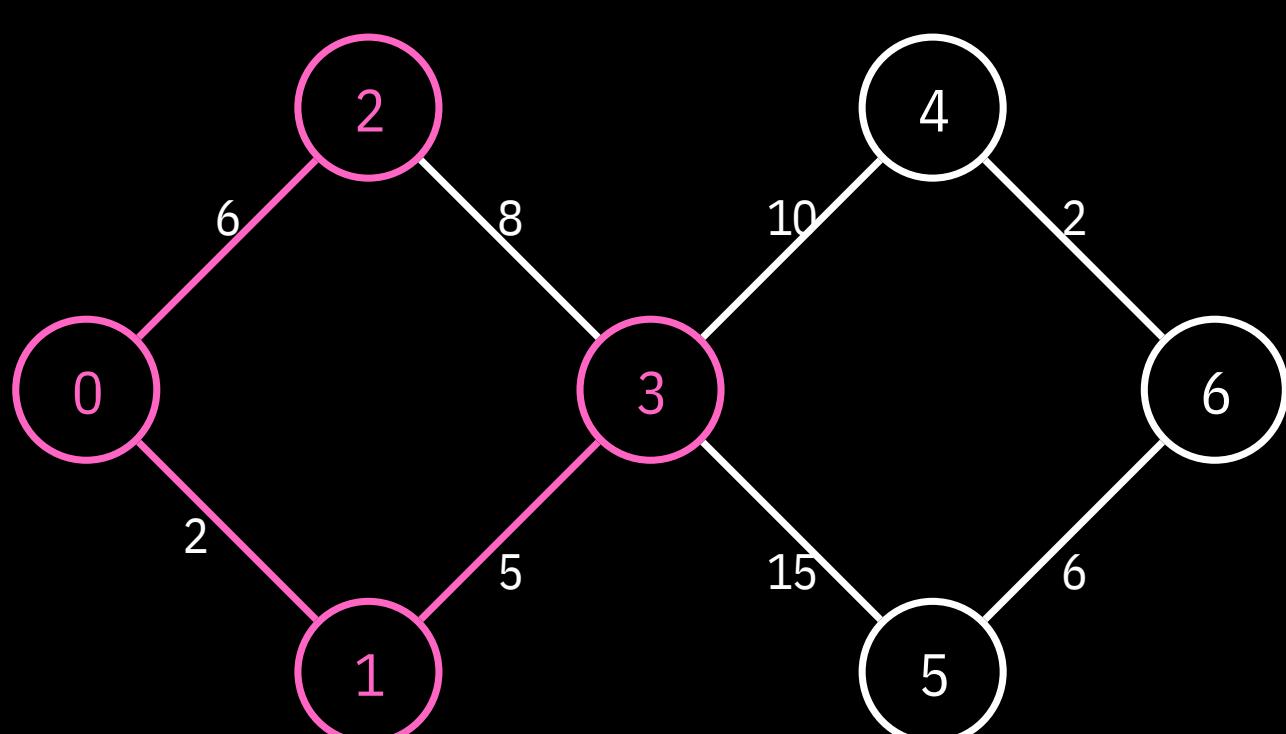
Nodos sin visitar
 $\{0, 1, 2, 3, 4, 5, 6\}$

Distancia:

0	0 ✓
1	2 ✓
2	∞
3	∞
4	∞
5	∞
6	∞

2. Distancia: Nodo 0 \rightarrow Nodo 1 = 2

3. Avance y verifique el nodo adyacente que es el nodo 3, así que márquelo como visitado y sume la distancia:



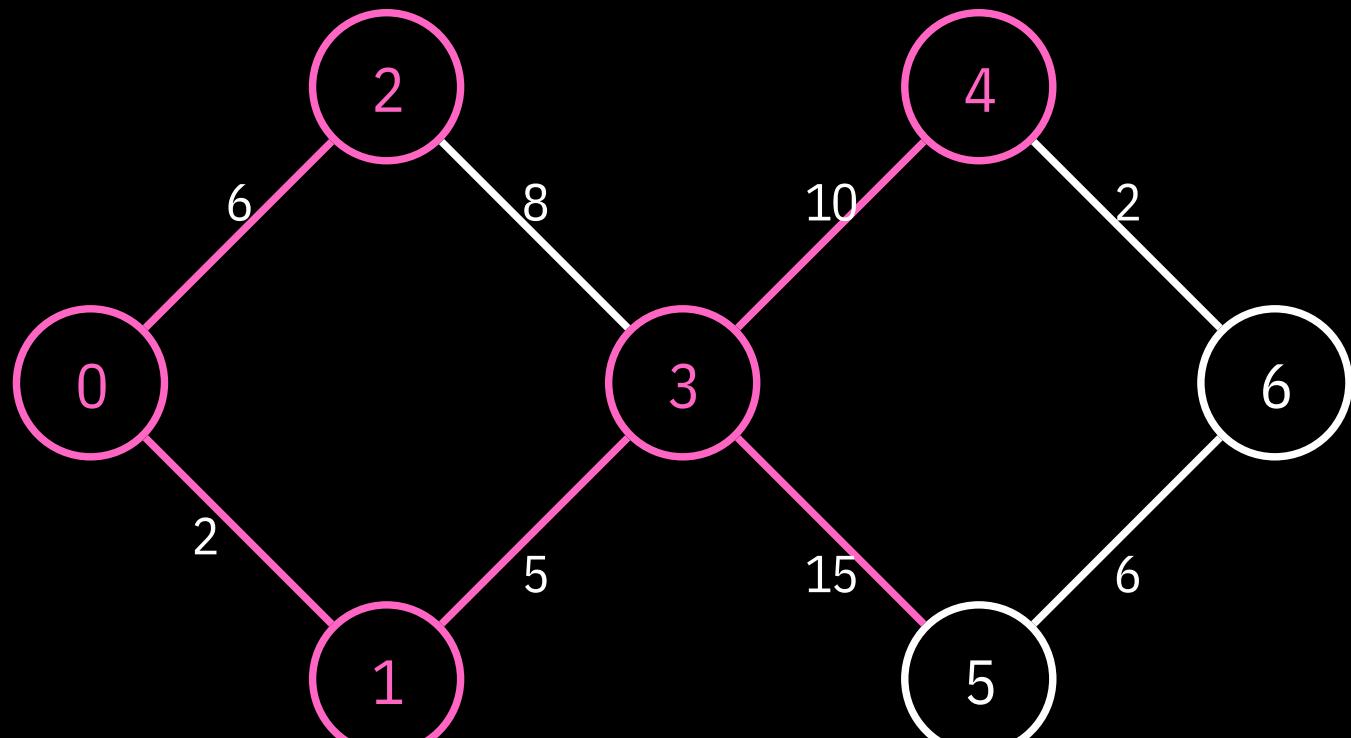
Nodos sin visitar
 $\{0, 1, 2, 3, 4, 5, 6\}$

Distancia:

0	0 ✓
1	2 ✓
2	6 ✓
3	7 ✓
4	∞
5	∞
6	∞

3. Distancia: Nodo 0 \rightarrow Nodo 1 \rightarrow Nodo 3 = 2 + 5 = 7

4. Ahora tenemos dos posibles caminos, el borde o arista con menor valor es el que conecta con el Nodo 4, por lo tanto, lo marcamos como visitado y sumamos la distancia:



Nodos sin visitar
 $\{0, 1, 2, 3, 4, 5, 6\}$

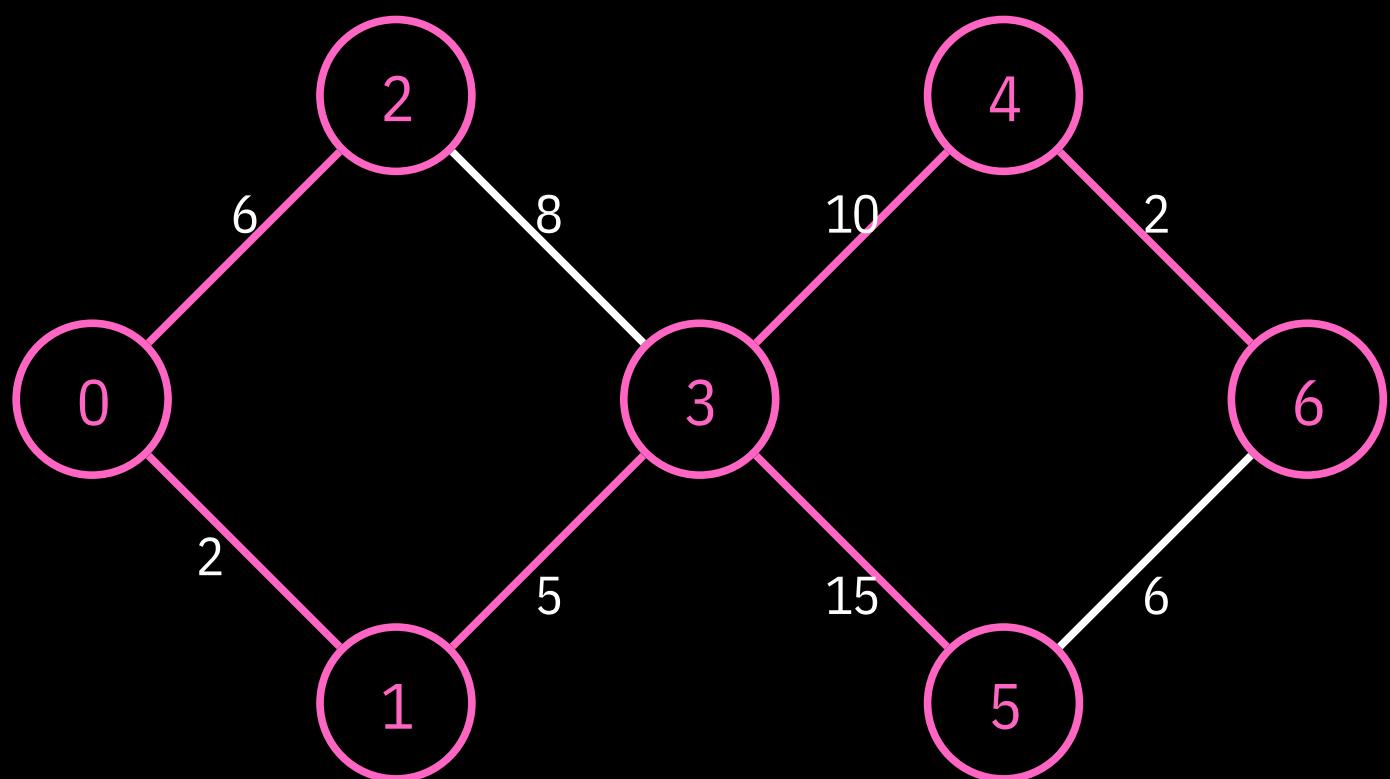
Distancia:

0	0 ✓
1	2 ✓
2	6 ✓
3	7 ✓
4	17 ✓
5	∞
6	∞

1

4. Distancia: Nodo 0 \rightarrow Nodo 1 \rightarrow Nodo 3 \rightarrow Nodo 4 = 2 + 5 + 10 = 17

5. Repita el ciclo y verifique el nodo adyacente que es el nodo 6, así que márquelo como visitado y sume la distancia.



Nodos sin visitar
 $\{0, 1, 2, 3, 4, 5, 6\}$

Distancia:

0	0 ✓
1	2 ✓
2	6 ✓
3	7 ✓
4	17 ✓
5	22 ✓
6	19 ✓

5. Distancia: Nodo 0 -> Nodo 1 -> Nodo 3 -> Nodo 4 -> Nodo 6 = $2 + 5 + 10 + 2 = 19$

PSEUDOCODIGO

```
Data: Grafo, origen
Result: Distancia mas corta del origen a otro nodo
// Inicializar las distancias de todos los nodos como
infinito, y la distancia al nodo de origen como 0;
distancias ← mapa(todos los nodos → infinito);
distancias ← 0;
// Inicializar un conjunto vacío de nodos visitados y
una cola de prioridad para realizar un seguimiento de
los nodos a visitar.;
visitado ← conjunto vacío;
cola ← nueva PriorityQueue(); cola.enqueue(origen, 0);
cola.enqueue(origen, 0);

while cola no está vacía do
    actual ← cola.dequeue();
    if actual esta en visitado then
        continue;
    end
    visitado.add(actual);
    foreach vecino en Grafo.vecino(actual) do
        distancia_tentativa ← distancias[actual] +
Grafo.distancia(actual, vecino);
        if distancia_tentativa < distancias[vecino] then
            distancias[vecino] ← distancia_tentativa;
            cola.enqueue(vecino, distancias[vecino]);
        end
    end
end
return distancias
```

1

Algorithm 1: Algoritmo de Dijkstra en pseudocódigo

IMPLEMENTACION DEL ALGORITMO DIJKSTRA EN PYTHON

```
# Implementacion de el Algoritmo de Dijkstra en Python

import heapq

class Nodo:
    # Constructor, inicializa los atributos que tendra un nodo
    def __init__(self, indice, distancia):
        self.indice = indice
        self.distancia = distancia

    # Metodo que permite comparar nodos por distancia
    def __lt__(self, other):
        return self.distancia < other.distancia

# Funcion Dijkstra
# nodos = numero total de nodos en el grafo
# adj = lista de adyacencia que representa el grafo
# nodo_origen = nodo de origen
def Dijkstra(nodos, adyacencia, nodo_origen):
    # Inicializacion de estructuras
    visitados = [False] * nodos # lista de nodos visitados
    map = {} # diccionario -> ¿de donde llego? y ¿cuanto recorrio desde el nodo de origen?
    q = [] # cola de prioridad

    # Inicializacion del nodo de origen
    # nodo_origen se establece con una distancia de 0
    map[nodo_origen] = Nodo(nodo_origen, 0)
    heapq.heappush(q, Nodo(nodo_origen, 0))

    # Procesamiento de la cola de prioridad
    while q:
        n = heapq.heappop(q)
        indice = n.indice
        distancia = n.distancia
        visitados[indice] = True
```

```

# Actualizacion de distancias para vecinos
adyList = adyacencia[indice]
for adyLink in adyList:
    vecino, peso = adyLink
    if not visitados[vecino]:
        if vecino not in map:
            map[vecino] = Nodo(indice, distancia +
peso)
        else:
            sn = map[vecino]
            if distancia + peso < sn.distancia:
                sn.indice = indice
                sn.distancia = distancia + peso
            heapq.heappush(q, Nodo(vecino, distancia +
peso))

# Preparacion del resultado final
resultado = [float('inf')] * nodos # Inicializa
con inf para nodos no alcanzables
for i in range(nodos):
    if i in map:
        resultado[i] = map[i].distancia

return resultado

# Funcion main
def main():
    adyacencia = [[] for _ in range(6)]

    nodos = 6
    aristas = 5
    u = [0, 0, 1, 2, 4]
    indice = [3, 5, 4, 5, 5]
    w = [9, 4, 4, 10, 3]

    # Creacion de la lista de adyacencia
    for i in range(aristas):
        edge = [indice[i], w[i]]
        adyacencia[u[i]].append(edge)

        edge2 = [u[i], w[i]]
        adyacencia[indice[i]].append(edge2)

```

```
# Ejecucion de algoritmo y visualizacion de resultados
nodo_origen = 1

resultado = Dijkstra(nodos, adyacencia,
nodo_origen)
print(resultado)

if __name__ == "__main__":
    main()
```

