Informe Semana 9

Marlon Zurita

2024-01-25

Tabla de Contenidos

Algoritmo de Dijkstra

1. Objetivos

- Implementar el algoritmo de Dijkstra para encontrar la ruta más corta en un grafo ponderado.
- Demostrar la utilidad del algoritmo en la determinación de la ruta más económica en un grafo dado.

2. Introducción

El algoritmo de Dijkstra, desarrollado por Edsger W. Dijkstra, es un algoritmo de búsqueda de caminos en un grafo ponderado que encuentra la ruta más corta entre dos nodos. Se basa en la actualización iterativa de costos para cada nodo, partiendo desde un nodo inicial hasta

llegar al nodo objetivo. Su aplicación principal se encuentra en la optimización de redes de transporte y comunicación (Geeks 2024).

3. Ejercicios planteados

Aplique el algoritmo de Dijkstra y despliegue la ruta más barata desde S hasta F. Trabaje con el siguiente ejemplo, dé nombre a los nodos

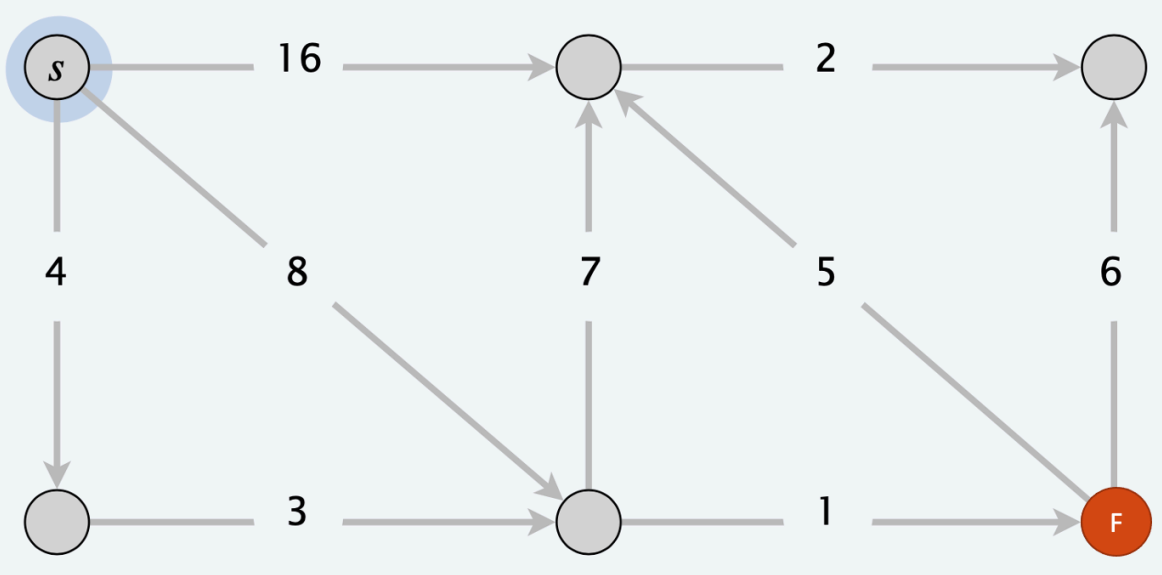

Figura 1: Ejemplo 1

```
graph = {}
graph['S'] = {'A': 4, 'B': 8, 'C': 16}
graph['A'] = {'B': 3}
graph['B'] = {'C': 7, 'F': 1}
graph['C'] = {'D': 2}
graph['D'] = {}
graph['F'] = {'C': 5, 'D': 6}

infinity = float('inf')

costs = {'A': 4, 'B': 8, 'C': 16, 'D': infinity, 'F': infinity}

parents = {'A': 'S', 'B': 'S', 'C': 'S', 'D': None, 'F': None, 'S': None}
```

```
processed = []

def find_lowest_cost_node(costs):
    lowest_cost = float('inf')
    lowest_cost_node = None
    for node in costs:
        cost = costs[node]
        if cost < lowest_cost and node not in processed:
            lowest_cost = cost
            lowest_cost_node = node
    return lowest_cost_node

current_node = find_lowest_cost_node(costs)

while current_node is not None:
    cost = costs[current_node]
    neighbors = graph[current_node]
    for neighbor in neighbors.keys():
        new_cost = cost + neighbors[neighbor]
        if costs[neighbor] > new_cost:
            costs[neighbor] = new_cost
            parents[neighbor] = current_node
    processed.append(current_node)
    current_node = find_lowest_cost_node(costs)

# Camino más barato de S a F
path = []
current_node = 'F'
total_cost = 0

while current_node is not None:
    path.insert(0, current_node)
    if parents[current_node] is not None:
        total_cost += graph[parents[current_node]][current_node]
    current_node = parents[current_node]

if path and path[0] == 'S':  # Check the new starting node
    print("Padres:", parents)
    print("Camino más barato de S a F:", ' -> '.join(path))
    print("Costo total del camino:", total_cost)
else:
    print("No existe un camino de S to F.")
```

```
Padres: {'A': 'S', 'B': 'A', 'C': 'F', 'D': 'F', 'F': 'B', 'S': None}
Camino más barato de S a F: S -> A -> B -> F
Costo total del camino: 8
```

Implemente el algoritmo de Dijkstra y despliegue la ruta más barata desde 0 hasta 6, trabaje con el siguiente ejemplo:

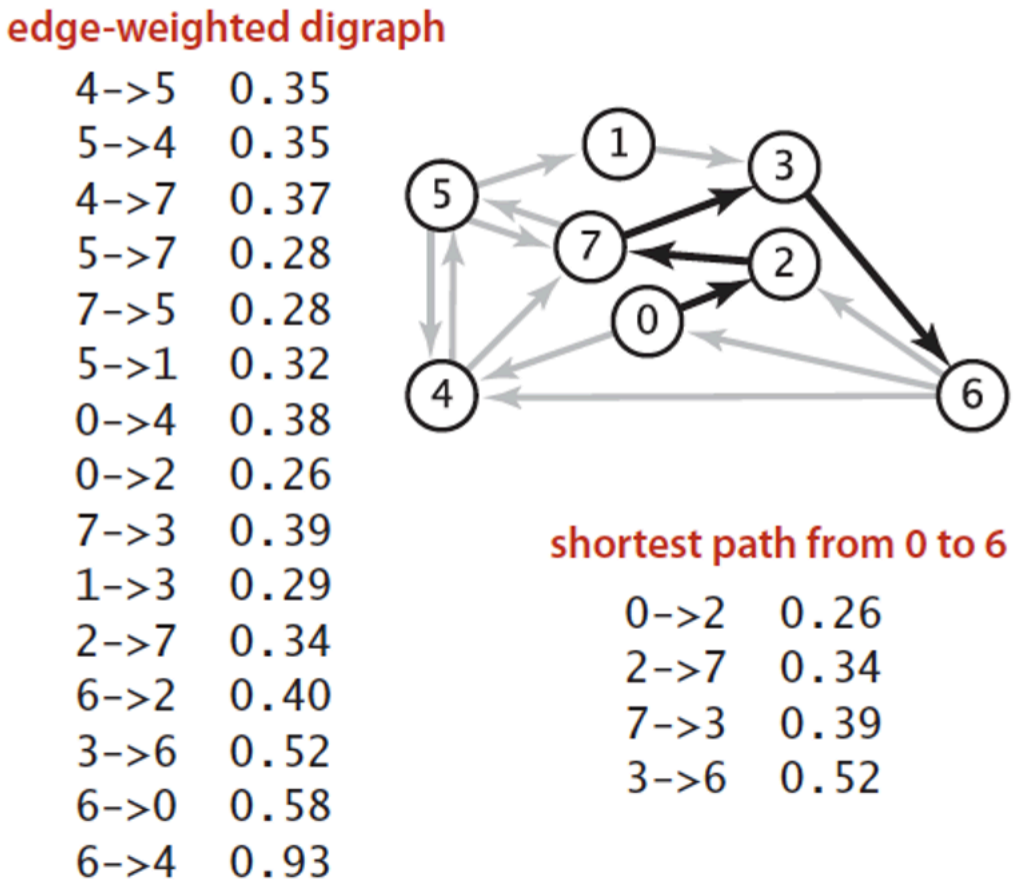

Figura 2: Ejemplo 2

```
graph = {}
graph['0'] = {'2': 0.26, '4': 0.38}
```

```
graph['1'] = {'3': 0.29}
graph['2'] = {'7': 0.34}
graph['3'] = {'6': 0.52}
graph['4'] = {'5': 0.35, '7': 0.37}
graph['5'] = {'1': 0.32, '4': 0.35, '7': 0.28}
graph['6'] = {'2': 0.40, '4': 0.93}
graph['7'] = {'3': 0.39, '5': 0.28}

infinity = float('inf')

costs = {}
costs['0'] = 0
costs['1'] = infinity
costs['2'] = 0.26
costs['3'] = infinity
costs['4'] = 0.38
costs['5'] = infinity
costs['6'] = infinity
costs['7'] = infinity

parents = {}
parents['0'] = None
parents['1'] = None
parents['2'] = '0'
parents['3'] = None
parents['4'] = '0'
parents['5'] = None
parents['6'] = None
parents['7'] = None

processed = []

def find_lowest_cost_node(costs):
    lowest_cost = float('inf')
    lowest_cost_node = None
    for node in costs:
        cost = costs[node]
        if cost < lowest_cost and node not in processed:
            lowest_cost = cost
            lowest_cost_node = node
    return lowest_cost_node
```

```
node = find_lowest_cost_node(costs)

while node is not None:
    cost = costs[node]
    neighbors = graph[node]
    for n in neighbors.keys():
        new_cost = cost + neighbors[n]
        if costs[n] > new_cost:
            costs[n] = new_cost
            parents[n] = node
    processed.append(node)
    node = find_lowest_cost_node(costs)

# Desplegar la ruta más barata desde S hasta F
path = []
current_node = '6'  # Cambiado a un nodo existente en el grafo
total_cost = 0

while current_node is not None:
    path.insert(0, current_node)
    if parents[current_node] is not None:
        total_cost += graph[parents[current_node]][current_node]
    current_node = parents[current_node]

if path and path[0] == '0':  # Verifica el nuevo nodo de inicio
    print("Padres:", parents)
    print("Ruta más barata desde 0 hasta 6:", ' -> '.join(path))
    print("Costo total de la ruta:", total_cost)
else:
    print("No hay una ruta desde S hasta F.")
```

```
Padres: {'0': None, '1': '5', '2': '0', '3': '7', '4': '0', '5': '4', '6': '3', '7': '2'}
Ruta más barata desde 0 hasta 6: 0 -> 2 -> 7 -> 3 -> 6
Costo total de la ruta: 1.51
```

4. Conclusiones

- El algoritmo de Dijkstra proporciona una solución efectiva para encontrar la ruta más corta en un grafo ponderado, destacando su aplicabilidad en diversos escenarios.

- La flexibilidad del código permite la aplicación del algoritmo a diferentes conjuntos de datos, lo que lo convierte en una herramienta versátil para la resolución de problemas de rutas en diversos contextos.

5. Referencias

Geeks, Geeks for. 2024. «Dijkstra's Algorithm». https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/.