Algoritmo de Dijkstra

Cosijopii

January 7, 2025

Introducción

El **algoritmo de Dijkstra** es un algoritmo para encontrar el camino más corto entre dos nodos en un grafo ponderado. Este algoritmo es útil cuando los pesos de las aristas son no negativos.

Pseudocódigo del Algoritmo de Dijkstra

El algoritmo de Dijkstra se puede describir en los siguientes pasos:

1. **Inicialización:**
   - Para cada nodo v en el grafo, asignar una distancia infinita $dist[v] = \infty$, excepto al nodo de inicio s que tiene $dist[s] = 0$.
   - Crear un conjunto de nodos no visitados (o una cola de prioridad).
   - Crear un diccionario o lista para almacenar el nodo previo de cada nodo en el camino más corto (esto es útil para reconstruir el camino).
2. **Algoritmo principal:**
   - Mientras haya nodos no visitados, seleccionar el nodo u no visitado con la distancia más pequeña $dist[u]$.
   - Marcar u como visitado.
   - Para cada vecino v de u, si v no ha sido visitado y la distancia desde u hasta v es menor que la distancia actual de v, actualizar $dist[v]$ y asignar u como el nodo previo de v.
3. **Reconstrucción del camino (opcional):**
   - Para reconstruir el camino desde el nodo destino t hasta el nodo origen s, seguir los nodos previos hasta llegar a s.

Pseudocódigo I

A continuación, el pseudocódigo del algoritmo de Dijkstra:

```
Dijkstra(Grafo G, Nodo origen s):
    1. Para cada nodo v en G:
         dist[v] = infinito
         prev[v] = NULL
    2. dist[s] = 0
    3. Crear una cola de prioridad Q, con todos
    los nodos de G ordenados por dist[v]

    4. Mientras Q no esté vacía:
         1. u = Nodo con la distancia más pequeña en Q
         2. Eliminar u de Q

         3. Para cada vecino v de u:
             1. Si v está en Q:
                 2. Si dist[u] + peso(u, v) < dist[v]:
```

Pseudocódigo II

```
                dist[v] = dist[u] + peso(u, v)
                prev[v] = u
                Actualizar la posición de v en Q

5. Retornar dist[] y prev[] (distancias
mínimas y el camino más corto
desde s a todos los nodos)
```

Explicación

- **Inicialización:**
  - Se asignan distancias infinitas a todos los nodos, excepto al nodo origen.
  - Se utiliza una cola de prioridad para seleccionar eficientemente el nodo con la distancia más pequeña.
- **Proceso de búsqueda:**
  - En cada iteración, se selecciona el nodo con la distancia más pequeña, se exploran sus vecinos y se actualizan las distancias.
- **Reconstrucción del camino (opcional):**
  - Se reconstruye el camino más corto desde el nodo destino siguiendo los nodos previos.

Complejidad

La complejidad temporal del algoritmo de Dijkstra utilizando una cola de prioridad implementada con un **heap binario** es:

$$O((V + E) \log V)$$

donde:

- V es el número de vértices.
- E es el número de aristas.

Esto hace que el algoritmo sea muy eficiente para grafos grandes.

Conclusión

El algoritmo de Dijkstra es ampliamente utilizado en aplicaciones como:

- Enrutamiento en redes.
- Navegación en mapas.
- Optimización de rutas.

Es una herramienta fundamental en el análisis de grafos y redes.

Ejemplo

Aplicamos el algoritmo de Dijkstra a un grafo con los siguientes nodos y aristas ponderadas:

- Nodo A está conectado con B (peso 1), C (peso 4).
- Nodo B está conectado con A (peso 1), C (peso 2), D (peso 5).
- Nodo C está conectado con A (peso 4), B (peso 2), D (peso 1).
- Nodo D está conectado con B (peso 5), C (peso 1).

Queremos encontrar el camino más corto desde el nodo A hasta el nodo D.

Inicialización

Inicializamos las distancias para cada nodo:

- $dist[A] = 0$, $prev[A] = \text{NULL}$
- $dist[B] = \infty$, $prev[B] = \text{NULL}$
- $dist[C] = \infty$, $prev[C] = \text{NULL}$
- $dist[D] = \infty$, $prev[D] = \text{NULL}$

También, agregamos los nodos a la cola de prioridad.

Paso 1: Nodo A

Empezamos con el nodo A (distancia 0) y actualizamos las distancias de sus vecinos:

- $dist[B] = 0 + 1 = 1$, $prev[B] = A$
- $dist[C] = 0 + 4 = 4$, $prev[C] = A$

Ahora, las distancias son:

- $dist[A] = 0$, $dist[B] = 1$, $dist[C] = 4$, $dist[D] = \infty$

Paso 2: Nodo B

Seleccionamos el nodo B (distancia 1). Ahora, actualizamos las distancias de sus vecinos:

- Para C, la nueva distancia sería $1 + 2 = 3$ que es mejor que 4, por lo actualizamos: $dist[C] = 3$, $prev[C] = B$
- Para D, la nueva distancia sería $1 + 5 = 6$, por lo actualizamos: $dist[D] = 6$, $prev[D] = B$

Ahora, las distancias son:

- $dist[A] = 0$, $dist[B] = 1$, $dist[C] = 3$, $dist[D] = 6$

Paso 3: Nodo C

Seleccionamos el nodo C (distancia 3). Actualizamos la distancia de D:

- Para D, la nueva distancia sería $3 + 1 = 4$, que es mejor que 6, por lo actualizamos: $dist[D] = 4$, $prev[D] = C$

Ahora, las distancias son:

- $dist[A] = 0$, $dist[B] = 1$, $dist[C] = 3$, $dist[D] = 4$

Paso 4: Nodo D

Finalmente, seleccionamos D (distancia 4). Como hemos llegado al nodo destino, podemos detener el algoritmo.

Reconstrucción del Camino

Para reconstruir el camino más corto de A a D, seguimos los nodos previos:

- El nodo previo de D es C.
- El nodo previo de C es B.
- El nodo previo de B es A.

Por lo tanto, el camino más corto es: $A \to B \to C \to D$.

Conclusión

El algoritmo de Dijkstra ha encontrado el camino más corto de A a D con una distancia total de 4. El camino es:

$$A \to B \to C \to D$$

Este algoritmo es útil para encontrar rutas óptimas en grafos ponderados y se aplica en áreas como el enrutamiento de redes y la navegación.