Algoritmo de Dijkstra: Camino más corto desde el origen a todos los vértices

Introducción

El algoritmo de Dijkstra es un algoritmo codicioso que se utiliza para encontrar el camino más corto desde un nodo de origen a todos los demás nodos en un gráfico ponderado con aristas no negativas. Este algoritmo es ampliamente utilizado en redes de computadoras, sistemas de transporte y otras áreas donde se requiere encontrar la ruta más eficiente entre dos puntos.

Descripción del Algoritmo

El algoritmo sigue estos pasos:

1. Inicializar la distancia desde el nodo de origen a todos los demás nodos como infinito, excepto el nodo de origen, que se inicializa en 0.
2. Marcar todos los nodos como no visitados. Establecer el nodo de origen como el nodo actual.
3. Para el nodo actual, considerar todos sus vecinos no visitados y calcular sus distancias desde el nodo de origen. Actualizar la distancia si la nueva calculada es menor.
4. Una vez considerados todos los vecinos del nodo actual, marcar el nodo como visitado. Un nodo visitado no se verificará de nuevo.
5. Seleccionar el nodo no visitado con la distancia más pequeña como el nuevo nodo "actual" y repetir el proceso.
6. El algoritmo continúa hasta que todos los nodos hayan sido visitados.

Implementación en C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();

void dijkstra(int src, const vector<vector<pair<int, int>>>& graph, vector<int>& dist)
{
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq;
    pq.push({0, src});
    dist[src] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
```

```
        for (const auto& edge : graph[u]) {
            int v = edge.first;
            int weight = edge.second;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }
}

int main() {
    int V = 5;  // Número de vértices
    vector<vector<pair<int, int>>> graph(V);

    // Agregando aristas al grafo
    graph[0].push_back({1, 10});
    graph[0].push_back({3, 5});
    graph[1].push_back({2, 1});
    graph[1].push_back({3, 2});
    graph[2].push_back({4, 4});
    graph[3].push_back({1, 3});
    graph[3].push_back({2, 9});
    graph[3].push_back({4, 2});
    graph[4].push_back({0, 7});
    graph[4].push_back({2, 6});

    vector<int> dist(V, INF);
    dijkstra(0, graph, dist);

    cout << "Distancia desde el nodo 0 a todos los demás nodos:" << endl;
    for (int i = 0; i < V; ++i) {
        cout << "0 -> " << i << " = " << dist[i] << endl;
    }

    return 0;
}
```

Ejemplo de Salida

```
Distancia desde el nodo 0 a todos los demás nodos:
0 -> 0 = 0
0 -> 1 = 8
0 -> 2 = 9
0 -> 3 = 5
0 -> 4 = 7
```

Ejemplos de Aplicaciones del Algoritmo de Dijkstra

1. **Redes de Computadoras**: Se utiliza para encontrar la ruta más corta en la transmisión de datos entre routers en una red.

2. **Sistemas de Navegación GPS**: En los dispositivos GPS, Dijkstra se utiliza para calcular la ruta más corta entre dos ubicaciones en un mapa.
3. **Gestión del Tráfico**: En ciudades inteligentes, el algoritmo se puede utilizar para optimizar las rutas de los vehículos para minimizar la congestión en las carreteras.

Conclusión

El algoritmo de Dijkstra es una herramienta poderosa para resolver problemas de caminos más cortos en grafos ponderados. Su implementación es relativamente sencilla y su aplicabilidad en el mundo real es extensa, abarcando desde redes de comunicación hasta sistemas de transporte.