Luis Enrique Franco Marín

410921353

Week 9: Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. The technique calculates the shortest path between a specified source node in the graph and every other node. It can also be used to find the shortest paths from a single node to a single destination node, with the method halting once the shortest path to the destination node has been identified.

Problem:

The first line of input gives the number of cases, N. N test cases follow. Each one starts with a line containing $n(2 \leq n \leq 20000)$,$m(0 \leq m \leq 50000)$,$S(0 \leq S < n)$ and $T(0 \leq T < n)$. $S \neq T$. The next m lines will each contain 3 integers: 2 different servers (in the range [0,n−1]) that are connected by a biderectional cable and the latency, w, along with cable ($0 \leq w \leq 10000$).

For each test case, output the line 'Case #x:' followed by the number of milliseconds required to send a message from S to T. Print 'unreachable' if there is no route from S to T

Code:

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> AdjList;
priority_queue<ii, vector<ii>, greater<ii> > pq;

int main() {
    int N, n, m, S, T, a, b, w;
    vi dist;
    cin>>N;
    for(int i=0; i<N; i++){
        cin >> n >> m >> S >> T;
        AdjList.assign(n, vii());
        while (m--) {
            cin >> a >> b >> w;
            AdjList[a].push_back(ii(b, w));
            AdjList[b].push_back(ii(a, w));
        }

        dist.assign(n, INT_MAX);
        dist[S] = 0;
        pq.push(ii(0, S));
        while (!pq.empty()) {
            ii front = pq.top(); pq.pop();
            int d = front.first, u = front.second;
            if (d == dist[u]) {
                for (int j= 0 ; j< AdjList[u].size() ; j++){
                    ii v = AdjList[u][j];
                    if (dist[u] + v.second < dist[v.first]) {
                        dist[v.first] = dist[u] + v.second;
                        pq.push(ii(dist[v.first], v.first));
                    }
                }
            }
        }
        cout <<"Case #"<<i+1<<": ";
        if (dist[T] != INT_MAX)
            cout << dist[T] << endl;
        else
            cout << "unreachable" << endl;

    }
    return 0;
}
```

Discussion:

The problem gives a n quantity of servers, these servers are interconnected. The servers may be interpreted as the nodes of a graph with bidirectional edges. The quantity of cables is given by m, each cable is assigned two computers to connect but also, they are given a latency which can be interpreted as the weight of each edge in the graph. This information given is enough to construct the graph, two computers S and T are also given as start and finish for an email to go. Then Dijkstra algorithm is used to find the minimum latency possible to connect these two computers or two see if they are connected at all.

This is done by creating an adjacency list for the graph, since the cables are bidirectional the adjacency list is symmetric. A priority list is created so the edges can be taken with the ones with least value are processed first. The space complexity is then the n*n from the adjacency list plus that can be n, so O(n^2). The time complexity is O(n*n) from traversing the adjacency list but this is done at worse case m cases from the priority queue.

Time: O(n^2*m)

Space: O(n^2)

Result: