# CSE 4304:Data Structure Lab-08
# Graph Basic

Namisa Najah Raisa 210042112

November 2023

## 1 Problem D:Travelling Cost

The problem is asking to find the minimum cost for travelling from a given source location(U) to a set of destination locations(V) in a city with roads and fixed travel costs.For each query I had to determine the minimum cost to travel from U to V using the available roads.If there is no path that connects U to V, the output would be 'NO PATH'.

**'const int MAX_LOCATIONS=501;'** defines the maximum number of locations in the city,and this value is used to define the size of various data structures in the program.

**'struct Edgeint to;int weight;;'** this line defines a 'struct' named 'Edge'.It represents and edge in the graph and has two integer members 'to',which refers to the target location of the edge, and 'weight', which refers to the cost of travelling that edge.

**'vector<vector<Edge» graph(MAX_LOCATIONS);'**this line creates a 2D vector named 'graph'.Each element in the 'graph' vector represents a location in the city,and the inner vector contains the edges from that location to other locations.The size of the graph vector is set to 'MAX_LOCATIONS' which allows for a maximum of 501 locations.

**'vector<int> dist(MAX_LOCATIONS, INT_MAX);'** this line creates a vector named 'dist' which is used to store the minimum distances from the source location to all other locations in the city.The vector is initialized with 'INT_MAX' indicating that the distances are initially unknown or infinite.

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
const int MAX_LOCATIONS = 501;
struct Edge
{
    int to;
    int weight;
};
vector<vector<Edge>> graph(MAX_LOCATIONS);
vector<int> dist(MAX_LOCATIONS, INT_MAX);
void dijkstra(int start)
{
    priority_queue<pair<int, int>, vector<pair<int,
       int>>, greater<pair<int, int>>> pq;
    dist[start] = 0;
    pq.push({0, start});

    while (!pq.empty())
    {
        int u = pq.top().second;
        int u_dist = pq.top().first;
        pq.pop();
        if (u_dist != dist[u]) continue;
        for (const Edge& edge : graph[u])
        {
            int v = edge.to;
            int weight = edge.weight;

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

int main()
{
    int N;
    cin >> N;
    for (int i = 0; i < N; i++)
    {
        int A, B, W;
        cin >> A >> B >> W;
        graph[A].push_back({B, W});
        graph[B].push_back({A, W});
    }
    int U;
    cin >> U;
    int Q;
    cin >> Q;
    for (int q = 0; q < Q; q++)
    {
        int V;
        cin >> V;
        dist.assign(MAX_LOCATIONS, INT_MAX);
        dijkstra(U);
        if (dist[V] != INT_MAX)
        {
            cout << dist[V] << endl;
        }
        else
        {
            cout << "NO PATH" << endl;
        }
    }
    return 0;
}
```

I used Dijkstra's algorithm to solve this problem.

**'void dijkstra(int start)'**→This function implements dijkstra's algorithm to find the minimum distances from a starting location('start') to all other locations in the city.

**'priority_queue<pair<int, int>, vector<pair<int, int», greater<pair<int, int»> pq;'**→This declares a priority queue 'pq' of pairs of integers.This is to keep track of locations to visit next during the algorithm.Each pair in the queue represents a location and its current distance from the source.The 'greater<pair<int ,int»'argument is used to ensure that the priority queue returns the smallest distance first.

**'dist[start]=0'**→This initializes the distance of the 'start' location to zero because the distance from the source to itself is zero.

**'pq.push(0, start)'**→This line pushes the starting location('start')and its distance(0) into the priority queue to begin the algorithm.

the WHILE loop that continues as long as there are locations in the priority queue to visit.

**'int u = pq.top().second'**→This retrieves the distance(represented by the second element of the pair)with the smallest distance from the priority queue and assigns it to 'u'.This will be the one currently being considered for updates.

**'int u_dist = pq.top().first'**→This retrieves the distance (represented by the first element of the pair) associated with the location 'u' and assigns it to the variable 'u_dist'. This is the current minimum distance from the source to 'u'.

**'pq.pop()'** this removes the location 'u' from the priority queue since it's about to be processed.

**'if (u_dist != dist[u]) continue'**→This checks if the distance I just retrieved (u_dist) is still the minimum distance for location 'u'. If it's not, it means that a shorter path to 'u' has already been found and processed, so it continues to the next location without further processing 'u'.

The FOR loop iterates through all the edges from location 'u' to other locations in the graph.

Inside the loop,I retrieve the target location of the current edge and assign it to the variable 'v'.

I also retrieve the weight(cost) associated with the edge and assign it to the variable 'weight'.

**'if (dist[u] + weight < dist[v])'**→This checks if the distance to location 'v' through the current edge is shorter than the previously known distance to 'v'.If it is,then the distance of location 'v' is updated to be the new shorter distance.

**'pq.push(dist[v], v)'**→If it is updated to location 'v', a pair representing the new distance and the location 'v' is pushed into the priority queue.This ensures that location 'v' and its updated distance will be later considered in the algorithm.

In the main function N is the variable to store the number of roads constructed in the city.

In the FOR loop that iterates N times.Inside the loop there are three integers.A,B and W.These are for storing the source location(A),destination location(B) and the cost of travelling that road(W).

**'graph[A].push_back(B, W)'**→This line updates the graph data structure.It adds and edge from location 'A' to location 'B' with a weight of 'W'.This is done by pushing a new 'Edge' object into the vector associated with location 'A'.I also added a reverse edge from 'B' to 'A' because the roads are bidirectional.

'int U' variable 'U' stores the source location from which Rohit wants to travel to other locations.

’int Q’ variable ’Q’ stores the number of queries Rohit wants to perform.Each query will involve finding the minimu cost to travel from location ’U’ to a specific destination.

In the FOR loop that iterates ’Q’ times ’V’ is the variable that stores the destination location of the current query.

**’dist.assign(MAX_LOCATIONS, INT_MAX)’**→This line re-initializes the ’dist’ vector with all elements set to ’INT_MAX’.This is done before each query to ensure that the algorithm starts with unknown distances.

Then dijkstra(U) function passes the source location ’U’.This calculates the minimum distances from ’U’ to all other locations in the city.

**’if (dist[V] != INT_MAX)’**→This line checks if there is a valid path from ’U’ to ’V’ by looking at the value stored in the ’dist’ vector for location ’V’.If it’s not equal to ’INT_MAX’,it means there is a path and you can print the minimum distance.If a valid path exists,print the minimum distance to the destination ’V’.IS there’s no valid path print "NO PATH."

________________X________________