# Assignment 2

Q(2)

Dijkstra's algorithm is a well-known algorithm for finding the shortest paths from a single source node to all other nodes in a weighted graph. The primary goal is to implement this algorithm with a time complexity of O(nlogn) , where n represents the number of nodes in the graph.

**MY CODE IS :**

#include<bits/stdc++.h>

using namespace std;

const long long INF = numeric\_limits<long long>::max();

void dijkstra(int n, vector<vector<pair<int, int>>> &graph, vector<long long> &dist, int source) {

priority\_queue<pair<long long, int>, vector<pair<long long, int>>, greater<>> pq;

dist[source] = 0;

pq.push({0, source});

while (!pq.empty()) {

long long current\_dist = pq.top().first;

int u = pq.top().second;

pq.pop();

if (current\_dist > dist[u]) continue;

for (auto &[v, weight] : graph[u]) {

long long distance = current\_dist + weight;

if (distance < dist[v]) {

dist[v] = distance;

pq.push({distance, v});

}

}

}

}

int main() {

int n, m;

cin >> n >> m;

vector<vector<pair<int, int>>> graph(n + 1);

for (int i = 0; i < m; ++i) {

int a, b, c;

cin >> a >> b >> c;

graph[a].emplace\_back(b, c);

}

vector<long long> dist(n + 1, INF);

dijkstra(n, graph, dist, 1);

for (int i = 1; i <= n; ++i) {

cout << dist[i] << " ";

}

cout << endl;

return 0;

}

**Explanation:**

 #include<bits/stdc++.h>: This is a common C++ header that includes almost all standard C++ libraries.

 using namespace std;: This statement allows us to avoid prefixing standard library names with std::

 n is the number of nodes.

 m is the number of edges.

 graph is an adjacency list representing the graph, where each node i has a list of pairs (neighbor, weight).

 The nodes a, b, and the weight c, then add (b, c) to the adjacency list of node a.

 dist is initialized with INF for all nodes.

 The dijkstra function is called with the graph, distance vector, and starting node (node 1).

 const long long INF = INT\_MAX;: This sets INF to the maximum value of a 32-bit integer. This value is used to initialize distances to a large number representing "infinity".

 Then we declared dijkstra function which implements Dijkstra's algorithm.

 priority queue (pq) that stores pairs of (distance, node) in ascending order based on distance. The greater<> comparator ensures it behaves as a min-heap i.e smallest distance come first.

 The distance to the source node is set to 0, and the source node is added to the priority queue.

 The loop continues until the priority queue is empty.

 The node with the smallest distance (u) is extracted.

 If the extracted distance is greater than the currently known distance to u, it means a shorter path has already been found, so it skips processing.

 For each neighbor v of node u, it calculates the new distance distance to v through u.

 If this new distance is shorter than the currently known distance to v, it updates the distance and adds v to the priority queue with the new distance.

 At the end we printed the shortest distance from the source node to each node.

The priority queue ensures that nodes are processed in order of their current known shortest distance, allowing the algorithm to achieve O((V+E)logV) time complexity.  
O((V+E)logV) is same as O(nlogn) where n is number of node.  
It is because we check at n = infinity. Generally V=E=n so

O((V+E)logV) = O((n+n)logn) = O(2nlogn) so time complexity will be O(n logn)