CS 254 ASSIGNMENT - 10

NAME : HRITIKESH BOYAPATI ROLL NO. 190002014

PROBLEM STATEMENT:

John has to attend some conferences. There are N cities numbered from 1 to N, and conferences can be held in any city. John lives in city1, and he will attend the conference as per schedule.

Design and implement an algorithm with a minimum time complexity that will find the shortest path from John's location to any conference's location. Consider all cities are connected. The graph is a simple graph, with no parallel edges or self-loop. It is not mandatory that the graph should be complete.

EXAMPLE :

INPUT :	OUTPUT :
6 9	6 5
1 2 1	1 2 1
1 4 5	1 3 3
2 3 2	1 4 3
2 5 1	1 5 2
2 4 2	1 6 4
3 6 2	
3 5 3	
5 6 2	
4 5 10	

ALGORITHM :

- The above-stated problem is the simple implementation of Dijkstra's shortest path algorithm.
- We are given the city i.e vertices and road with the length i.e.the edges with weight.
- As city1 is the start city we will consider vertex 1 as the source vertex and find the shortest path to all other vertexes.
- We will iterate through all the vertex to get the solution.
- The simple logic for the algo islet we are at vertex u and vertex v is adjacent to it so we will check if v is not visited yet and if dist[v]>dist[u]+dist of v from u (i.e value of the weight of the edge from u to v).
- Then, we will update the distance of vertex v as dist[u]+weight of (u,v) and append v to the priority queue.
- Like this, we will check for all vertex adjacent to u and keep appending these vertices to a priority queue.
- The main motive of using the priority queue is that every time we select the vertex u, which has the least distance from the source vertex.
- At last, our distance array will have the shortest distance from the source vertex to all other vertexes.

CODE :

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
        int n,m,s=1;
        cin >> n >> m;
        vector<pair<int,int> > g[n+1];

        int a,b,wt;
        for(int i = 0; i<m ; i++){
                cin >> a >> b >> wt;
```

```
		g[a].push_back(make_pair(b,wt));
		g[b].push_back(make_pair(a,wt));
	}

	priority_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int> > > pq;
	vector<int> distTo(n+1,INT_MAX);

	distTo[s] = 0;
	pq.push(make_pair(0,s));
	while( !pq.empty() ){
		int dist = pq.top().first;
		int prev = pq.top().second;
		pq.pop();

		vector<pair<int,int> >::iterator it;
		for( it = g[prev].begin() ; it != g[prev].end() ; it++){
			int next = it->first;
			int nextDist = it->second;
			if( distTo[next] > distTo[prev] + nextDist){
				distTo[next] = distTo[prev] + nextDist;
				pq.push(make_pair(distTo[next], next));
			}
		}

	}

	cout << "The distances from s, " << s << ", are : \n";
	for(int i = 1 ; i<=n ; i++)	cout << distTo[i] << " ";
	cout << "\n";

	return 0;
}
```

TIME COMPLEXITY : O(ElogV)

The time complexity remains O(ElogV) as there will be at most O(E) vertices in the priority queue and O(log E) is the same as O(log V)

OUTPUT:

```
6 8

1 2 1

1 4 5

2 3 2

2 5 1

2 4 2

3 6 2

3 5 3

5 6 2

6 5
1 2 1
1 3 3
1 4 3
1 5 2
1 6 4
```

GRAPH :