Shortest Path Assignment ACA (230679)

1. Implementation of Dijkstra's Algorithm

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

#define INF 1000000000000000
#define INF 0x3f3f3f3f

//typedef pair<int,int>iPair;

void addEdge(vector<pair<int,int>>adj[],int u,int v,int wt){
    adj[u].push_back(make_pair(v,wt));
    adj[v].push_back(make_pair(u,wt));
}

void dijkstra(vector<pair<int,int>>adj[],int V,int src){

    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>
pq;
    vector<int>dist(V,INF);
    pq.push({0,src});
    dist[src]=0;
    while(!pq.empty()){
        int u=pq.top().second;
        pq.pop();

        for(auto x:adj[u]){
            int v=x.first;
            int weight=x.second;

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


        }

    }
    cout<<"Vertex Distance from Source\n";
    for(int i=0;i<V;i++){
        cout<<i<<"\t"<<"\t"<<dist[i];
        cout<<"\n";
    }

}
```

```
int main(){
    int V=9;
    vector<pair<int,int>>adj[V];
    addEdge(adj,0,1,4);
    addEdge(adj,0,7,8);
    addEdge(adj,1,2,8);
    addEdge(adj,1,7,11);
    addEdge(adj,2,3,7);
    addEdge(adj,2,8,2);
    addEdge(adj,2,5,4);
    addEdge(adj,3,4,9);
    addEdge(adj,3,5,14);
    addEdge(adj,4,5,10);
    addEdge(adj,5,6,2);
    addEdge(adj,6,7,1);
    addEdge(adj,6,8,6);
    addEdge(adj,7,8,7);
    dijkstra(adj,V,0);
    return 0;

}
```

We maintain two sets, first set of vertices for which distances are yet to be finalised and the other for which the distance is already finalised. Initially, we do not finalise the distance for any vertex, we pick up vertices one by one from the other set and then we finalise the distance. That vertex is picked which has minimum distance. After picking the vertex, we go through all adjacent of it, and perform the operation called **RELAX.**

Relax(u,v), where **u** is the vertex which is just finalised, and **v** is the adjacent vertex. In **RELAX**, since the distance of **u** is finalised and we go to every adjacent of **u**, we see that if we get a shorter path through **u** to **v.** In simple words, we compare current distance of **v** with the distance that we are getting through **u**. If that distance of v is greater, then we reinitialise the distance of **v**, i.e.

```
if ( dist[v]>dist[u]+weight){
  dist[v]=dist[u]+weight ;
}
```

We begin by making vector of pair of integers, imputs being two nodes and weights (or the distance between them). Then we make a priority queue (specifically **min heap priority queue**). Here, **greater<int>** is a comparison function which sets up a min heap such that smallest element will be at the top (since we want smallest distance).The first element of each pair is the weight as the first item is by default used to compare. Then we make a

vector ‘dist’ of size V (no of nodes) to keep track of the distances of all vertices from source vertex. Initially, we initialise all the distances as infinity and then update the distance. Then, we push our first pair, of source itself and its distance from the source that is 0 into the priority queue. Then we initialise the distance of source as 0 (rest all distances are still infinity). Then we run a while loop for element in priority queue. We traverse through all the adjacent of the picked element and use **RELAX** operation to finalise and update the distances. At last, we print the ‘dist’ array to get distances of all the vertices from source.

2.

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

#define INF 1000000000000000

void dijkstra(long long int N,vector<pair<long long int,long long int
>>adj[]){
    vector<long long int>distance(N,INF);
    distance[0]=0;
    priority_queue<pair<long long int,long long int>,vector<pair<long long
int,long long int>>,greater<pair<long long int,long long int>>>pq;
    pq.push({0,0});
    while(!pq.empty()){
        long long int node=pq.top().first;
        long long int dist=pq.top().second;
        pq.pop();
        if(dist>distance[node])continue;
        for(auto u:adj[node]){
           long long int v=u.first;
            long long int weight=u.second;
            if(distance[v]>dist+weight){
                distance[v]=dist+weight;
                pq.push({v,distance[v]});
            }

        }

    }
    for(long long int i=0;i<N;i++){
        cout<<distance[i]<<" ";
    }

}
```

```cpp
int32_t main(){
    long long int n,m;
    cin>>n>>m;
    vector<pair<long long int,long long int>>adj[n];
    for(long long int i=0;i<m;i++){
        long long int a,b,wt;
        cin>>a>>b>>wt;
        a--,b--;
        adj[a].push_back({b,wt});
    }

    dijkstra(n,adj);
    return 0;

}
```

The implementation of above problem is same as that in qs1, the only difference is that in qs 1, we have 0 as one of our vertex, but in qs 2, numbering of cities start from 1, thus, accordingly we have modified our adjacency list (in main function, for loop).

Sender:	Richard_Hazer
Submission time:	2024-06-17 17:31:07 +0300
Language:	C++20
Status:	READY
Result:	RUNTIME ERROR

Test results

test	verdict	time	
#1	ACCEPTED	0.00 s	»
#2	ACCEPTED	0.00 s	»
#3	ACCEPTED	0.00 s	»
#4	ACCEPTED	0.00 s	»
#5	ACCEPTED	0.00 s	»
#6	ACCEPTED	0.28 s	»
#7	ACCEPTED	0.28 s	»
#8	ACCEPTED	0.28 s	»
#9	ACCEPTED	0.28 s	»
#10	ACCEPTED	0.28 s	»
#11	ACCEPTED	0.16 s	»
#12	ACCEPTED	0.15 s	»
#13	ACCEPTED	0.00 s	»
#14	ACCEPTED	0.15 s	»
#15	ACCEPTED	0.16 s	»
#16	ACCEPTED	0.14 s	»
#17	ACCEPTED	0.13 s	»
#18	ACCEPTED	0.18 s	»
#19	ACCEPTED	0.27 s	»
#20	ACCEPTED	0.27 s	»
#21	ACCEPTED	0.21 s	»
#22	RUNTIME ERROR	0.97 s	»
#23	ACCEPTED	0.15 s	»

Round Trip -
Monsters -
Shortest Routes I ×
Shortest Routes II -
High Score -
Flight Discount -
Cycle Finding -
...

Your submissions

2024-06-17 17:31:07 ×
2024-06-14 07:16:01 ×
2024-06-14 07:14:46 ×
2024-06-14 07:10:58 ×
2024-06-14 07:07:50 ×
2024-06-14 07:03:42 ×
2024-06-14 07:02:22 ×
2024-06-14 06:51:08 ×
2024-06-14 06:49:41 ×
2024-06-14 06:45:55 ×
2024-06-14 06:43:19 ×
2024-06-14 06:40:11 ×
2024-06-14 06:37:15 !
2024-06-13 07:41:06 ×
2024-06-13 07:39:37 ×
2024-06-13 07:38:17 ×
2024-06-13 07:29:50 ×
2024-06-13 07:20:48 ×
2024-06-13 07:20:48 ×
2024-06-13 07:17:44 !