"Heaven's Light is Our Guide"

Department of Computer Science & Engineering

RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOG

Lab Report-03

Submitted By:
Name: Khandoker Sefayet Alam
Roll:2003121
Department: Computer Science & Engineering
Section-C

Course code: CSE 2202
Course name: Algorithm Analysis and Design Sessional

Submitted To:

A. F. M. Minhazur Rahman
Lecturer, Department of CSE,RUET

Problem 1: Implementation of Disjoint Set Union data structure

Statement:

Implement disjoint sets union data structure. You have to perform queries of two types:

- `union` u v — unite two sets that contain u and v, respectively;
- `get` u v — check that two elements u and v belong to the same set.

Algorithm:

To solve this kind of problems, we make individual nodes by making a node it's own parent. This is done by make() operation. When two nodes are asked to be made of the same set, we at first check if they belong to the same set already or not. To do this, we maintain a parent array and using find() operation we traverse the parent nodes and check if we end up in the same parent. To make them of the same set, we use union operation. It checks current nodes' parent nodes' sizes and compares them. Then we make the smaller set as the subset of the larger set.

Code:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long




///PRINTING

#define deb(x) cout << #x << "=" << x << endl
#define deb2(x, y) cout << #x << "=" << x << "," << #y << "=" << y << endl
#define nn '\n'

#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()
```

```cpp
//CONSTANTS
#define md                  10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;



/// Data structures
typedef unsigned long long ull;
typedef pair<ll, ll>    pll;
typedef vector<ll>      vl;
typedef vector<pll>     vpll;
typedef vector<vl>      vvl;
template <typename T> using PQ = priority_queue<T>;
template <typename T> using QP = priority_queue<T,vector<T>,greater<T>>;

template <typename T> using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
template <typename T,typename R> using ordered_map = tree<T, R , less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
;


ll par[N];
ll sz[N];
// multiset<int> sizes;

void make(ll v){
    par[v]=v;
    sz[v]=1;
    // sizes.insert(1);
}

int find(ll v){
    if(v==par[v]) return v;
    return par[v]=find(par[v]);
}
```

```
void Union(ll a,ll b){
    a=find(a);
    b=find(b);
    if(a!=b){
        if(sz[a]<sz[b]) swap(a,b);
        par[b]=a;
        // merge(a,b);
        sz[a]+=sz[b];
    }
}
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
     t=1;
    //cin>>t;

    while(t--){
        ll n,m;
        cin>>n>>m;
        string s;
        for(ll i=1;i<=n;i++){
            make(i);
        }
        ll u,v;
        while(m--){
            cin>>s>>u>>v;
            if(s=="union"){
                Union(u,v);
            }
            else{

                if(find(u)==find(v)){
                    cout<<"YES"<<endl;
                }
                else cout<<"NO"<<endl;
            }
        }

    }
    return 0;
}
```

Inputs And Outputs:

Input-01:

4 4

union 1 2

union 1 3

get 1 4

get 2 3

Output-01:

NO

YES

Problem 2: Implementation of Kruskal's algorithm for MST using DSU. Solve the **HackerRank** problem "**Kruskal (MST): Really Special Subtree**"

Statement:

Given an undirected weighted connected graph, find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and:

- There is only one exclusive path from a node to every other node.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- No cycles are formed

To create the Really Special SubTree, always pick the edge with smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

- Choose the edge that minimizes the sum u+v+wt where u and are v vertices and wt is the edge weight.
- If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules.

For example, given the following edges:

u	v	wt
1	2	2
2	3	3
3	1	5

First choose 1-> 2 at weight 2. Next choose 2->3 at weight 3. All nodes are connected without cycles for a total weight of 3+2=5.

Algorithm:

This problem can be solved using Disjoint set union(dsu) and krushkal's algorithm. To implement this, at first we store edges and sort them according to their edges. Now, for each edge, we check the nodes connected by it are of the same set or not using dsu. If they are not, we consider the edge to be the part of the minimum spanning tree and union those nodes and add the weight of the edge to the answer.

Code:

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long


#define nn '\n'

#define all(x) x.begin(), x.end()
#define rev(x) reverse(all(x))

//CONSTANTS
#define md                  10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;
```

```cpp
int par[N];
int sz[N];
// multiset<int> sizes;

void make(int v){
    par[v]=v;
    sz[v]=1;
    // sizes.insert(1);
}

int find(int v){
    if(v==par[v]) return v;
    return par[v]=find(par[v]);
}

// void merge(int a,int b){
//     sizes.erase(sizes.find(sz[a]));
//     sizes.erase(sizes.find(sz[b]));
//     sizes.insert(sz[a]+sz[b]);
// }

void Union(int a,int b){
    a=find(a);
    b=find(b);
    if(a!=b){
        if(sz[a]<sz[b]) swap(a,b);
        par[b]=a;
        // merge(a,b);
        sz[a]+=sz[b];
    }
}
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
     t=1;
    //cin>>t;

    while(t--){
```

```
    vector<pair<ll,pair<ll,ll>> >edges;
    ll n,m;
    cin>>n>>m;
    for(ll i=1;i<=n;i++){
      make(i);
    }
    ll u,v,w;
    while(m--){
      cin>>u>>v>>w;
      // Union(u,v);
      edges.push_back({w,{u,v}});
    }
    sort(all(edges));
    ll ans=0;
    for(auto it:edges){
      u=it.second.first;
      v=it.second.second;
      if(find(u)!=find(v)){
          Union(u,v);
          ans+=it.first;
      }
    }
    cout<<ans<<endl;
  }


  return 0;
}
```

Input and Outputs:

Input-01:

```
4 6
1 2 5
1 3 3
4 1 6
2 4 7
3 2 4
3 4 5
```

Output-01:

```
12
```

Input-02:

4 6

2 1 1000

3 4 299

2 4 200

2 4 100

3 2 300

3 2 6

Output-02:

1106

Problem 3: Implementation of Prim's algorithm for MST. Solve the **HackerRank** problem **"Prim's (MST) : Special Subtree"**

Statement:

Given a graph which consists of several edges connecting its nodes, find a subgraph of the given graph with the following properties:

- The subgraph contains all the nodes present in the original graph.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- It is also required that there is **exactly one, exclusive** path between any two nodes of the subgraph.

One specific node S is fixed as the starting point of finding the subgraph using Prim's Algorithm.

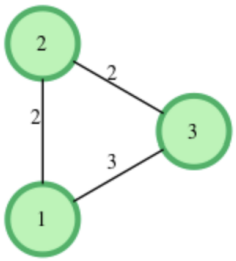
Find the total weight or the sum of all edges in the subgraph.

Example

$n = 3$

$edges = [[1, 2, 2], [2, 3, 2], [1, 3, 3]]$

$start = 1$



Starting from node 1, select the lower weight edge, i.e. $1 \leftrightarrow 2$, weight 2.

Choose between the remaining edges, $1 \leftrightarrow 3$, weight 3, and $2 \leftrightarrow 3$, weight 2.

The lower weight edge is $2 \leftrightarrow 3$ weight 2.

All nodes are connected at a cost of $2 + 2 = 4$. The edge $1 \leftrightarrow 3$ is not included in the subgraph.

Algorithm:

This problem can be solved using Prim's MST algorithm. This algorithm is similar to Dijkstra algorithm. In this algorithm, we maintain a minimum priority queue that keeps the edge with the lowest weight at the top. Initially, all edges of the source are pushed to the priority queue and then we take the top element each time and traverse the nodes connected by the edges. For each edge, we check if the second node is already part of the current MST using a visited array. If not, we traverse the child nodes of the current node and check if they are already visited. If not, they are pushed to the queue.This way, we get the minimum edges that turns the graph into a minimum spanning tree. Each time we find a edge with unvisited node, we add it's weight to the answer as the top element always gives the minimum current weight possible. Thus, we achieve the weight of the MST.

Code:

```cpp
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

//VVI
#define fast ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define pb                  push_back
#define ll                  long long


///PRINTING

#define deb(x) cout << #x << "=" << x << endl
#define deb2(x, y) cout << #x << "=" << x << "," << #y << "=" << y << endl
#define nn '\n'

#define mem(a,b) memset(a,b,sizeof(a))
#define all(x) x.begin(), x.end()
#define rev(x) reverse(all(x))

//CONSTANTS
#define md                  10000007
#define PI 3.14159265358979323846
const double EPS = 1e-9;
const ll N = 2e5+10;
const ll M = 1e9+7;


/// Data structures
typedef unsigned long long ull;
typedef pair<ll, ll>    pll;
typedef vector<ll>      vl;
typedef vector<pll>     vpll;
typedef vector<vl>      vvl;
template <typename T> using PQ = priority_queue<T>;
template <typename T> using QP = priority_queue<T,vector<T>,greater<T>>;

template <typename T> using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
template <typename T,typename R> using ordered_map = tree<T, R , less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
```

```
;


bool visit[N];
vector <pll > g[N];
ll n,m;
int prims_mst (ll src) {
   QP<pair<ll,pll>> pq;
   memset (visit, 0, sizeof (visit));
   visit[src] = 1;
   for (auto child:g[src]) {
      ll curr_v = child.first;
      ll curr_w = child.second;
      pq.push({curr_w,{src,curr_v}});
   }
   ll cnt = 0;
   while (!pq.empty()) {
      auto a = pq.top();
      pq.pop();
      ll from, to, w;
      from = a.second.first;
      to = a.second.second;
      w = a.first;
      //printf ("%d %d %d\n", u, v, w);
      if (!visit[to]) {
         visit[to] = 1;
        //  cout<<from<<" "<<to<<nn; //prints added edges
         cnt += w;
         from = to;
         for (auto child: g[from]) {
            to = child.first;
            w = child.second;
            if (!visit[to]) pq.push({w,{from,to}});
         }
      }
   }
   return cnt;
}
void reset(){
    for(ll i=0;i<=n;i++){
        g[i].clear();
        visit[i]=0;
    }
}
```

```
int main()
{
    fast;
     ll t;
    //setIO();
     //ll tno=1;;
     t=1;
    //cin>>t;

    while(t--){
      cin>>n>>m;
      ll x,y,w;
      for(ll i=0;i<m;i++){
        cin>>x>>y>>w;
        g[x].push_back({y,w});
        g[y].push_back({x,w});
      }

      ll src;
      cin>>src;
      ll ans=prims_mst(src);
      cout<<ans<<nn;
    }


    return 0;
}
```

Input and output:

Input-01:

5 6

1 2 3

1 3 4

4 2 6

5 2 2

2 3 5

3 5 7

1

Output-01:

15

Input-02:

4 6

2 1 1000

3 4 299

2 4 200

2 4 100

3 2 300

3 2 6

2

Output-02:

1106