|  |  |
| --- | --- |
|  | |
| Assignment No: 08     1. A program of Minimum Spanning Tree (Kruskal’s algorithm). 2. A program of Minimum Spanning Tree (Prims algorithm). | |
| Date of Performance: 08/09/2019  Date of Submission: 16/09/2019 | Student ID: 17-02-04-058  Group: B1 |

**No.1:**

#include<bits/stdc++.h>

using namespace std;

const int MAX = 1e4 + 5;

int id[MAX], nodes, edges;

pair <long long, pair<int, int> > p[MAX];

void initialize()

{

for(int i = 0;i < MAX;++i)

id[i] = i;

}

int root(int x)

{

while(id[x] != x)

{

id[x] = id[id[x]];

x = id[x];

}

return x;

}

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

for(int i = 0;i < edges;++i)

{

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

if(root(x) != root(y))

{

minimumCost += cost;

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

initialize();

cin >> nodes >> edges;

for(int i = 0;i < edges;++i)

{

cin >> x >> y >> weight;

p[i] = make\_pair(weight, make\_pair(x, y));

}

sort(p, p + edges);

minimumCost = kruskal(p);

cout << minimumCost << endl;

return 0;

}

**No.2:**

#include<bits/stdc++.h>

using namespace std;

const int MAX = 1e4 + 5;

typedef pair<long long, int> PII;

bool marked[MAX];

vector <PII> adj[MAX];

long long prim(int x)

{

priority\_queue<PII, vector<PII>, greater<PII> > Q;

int y;

long long minimumCost = 0;

PII p;

Q.push(make\_pair(0, x));

while(!Q.empty())

{

p = Q.top();

Q.pop();

x = p.second;

if(marked[x] == true)

continue;

minimumCost += p.first;

marked[x] = true;

for(int i = 0;i < adj[x].size();++i)

{

y = adj[x][i].second;

if(marked[y] == false)

Q.push(adj[x][i]);

}

}

return minimumCost;

}

int main()

{

int nodes, edges, x, y;

long long weight, minimumCost;

cin >> nodes >> edges;

for(int i = 0;i < edges;++i)

{

cin >> x >> y >> weight;

adj[x].push\_back(make\_pair(weight, y));

adj[y].push\_back(make\_pair(weight, x));

}

minimumCost = prim(1);

cout << minimumCost << endl;

return 0;

}