Name: Kapil Arun Patil

Reg.no: 2020BIT153

Assignment no 9

Implement the following algorithm for minimum cost spanning tree

Code:

```
#include <iostream>

#include <vector>

#include <queue>

#include <climits>


using namespace std;


typedef pair<int, int> pii;


class Graph {

private:

   int V;

   vector<vector<pii>> adj;


public:

   Graph(int V) {

      this->V = V;

      adj.resize(V);

   }


   void add_edge(int u, int v, int weight) {

      adj[u].push_back({v, weight});

      adj[v].push_back({u, weight});

   }
```

```
vector<pii> prim_mst() {
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    vector<int> key(V, INT_MAX);
    vector<bool> visited(V, false);
    vector<pii> parent(V, {-1, -1});

    int src = 0;
    pq.push({0, src});
    key[src] = 0;

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        if (visited[u])
            continue;

        visited[u] = true;

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

            if (!visited[v] && weight < key[v]) {
                key[v] = weight;
                pq.push({key[v], v});
                parent[v] = {u, weight};
            }
        }
    }
```

```
        vector<pii> mst;
        for (int i = 1; i < V; i++)
            mst.push_back({parent[i].first, i});

        return mst;
    }
};

int main() {
    int V = 5;
    Graph g(V);
    g.add_edge(0, 1, 2);
    g.add_edge(0, 3, 6);
    g.add_edge(1, 2, 3);
    g.add_edge(1, 3, 8);
    g.add_edge(1, 4, 5);
    g.add_edge(2, 4, 7);
    g.add_edge(3, 4, 9);

    vector<pii> mst = g.prim_mst();

    for (auto& edge : mst)
        cout << edge.first << " - " << edge.second << endl;

    return 0;
}
```

2) Kruskal's algorithm using Min Hea

Code-

```
#include <iostream>
#include <vector>
```

```
#include <algorithm>
#include <queue>

using namespace std;

struct Edge {
    int u, v, w;
};

class UnionFind {
private:
    vector<int> parent, rank;

public:
    UnionFind(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }

    void unite(int x, int y) {
        int px = find(x), py = find(y);
        if (px != py) {
            if (rank[px] < rank[py]) {
                parent[px] = py;
            } else if (rank[px] > rank[py]) {
                parent[py] = px;
            } else {
                parent[px] = py;
                rank[py]++;
            }
        }
    }
};

class Kruskal {
private:
    vector<Edge> edges;
    int n, m;

public:
    Kruskal(int n, int m) : n(n), m(m) {}

    void addEdge(int u, int v, int w) {
        edges.push_back({u, v, w});
    }

    vector<Edge> findMST() {
        vector<Edge> MST;
        sort(edges.begin(), edges.end(), [](Edge a, Edge b) {
            return a.w < b.w;
        });
        UnionFind uf(n);
        for (auto e : edges) {
            if (uf.find(e.u) != uf.find(e.v)) {
                uf.unite(e.u, e.v);
                MST.push_back(e);
                if (MST.size() == n - 1) break;
            }
        }
        return MST;
    }
};

int main() {
    int n, m;
    cin >> n >> m;
    Kruskal kruskal(n, m);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        kruskal.addEdge(u - 1, v - 1, w);
    }
    vector<Edge> MST = kruskal.findMST();
    for (auto e : MST) {
        cout << e.u + 1 << " " << e.v + 1 << " " << e.w << endl;
    }
    return 0;
}
```