Name:- Yash Sanjay Kale
Reg NO:- 2020BIT047
Subject:- DAA

Practical No. 09: Implement the following algorithm for minimum cost spanning tree

1) Prims algoritham using Binary Heap

```
#include <iostream>
#include <vector>
#include <queue>
#include <utility>

using namespace std;

const int MAXN = 1e5 + 5;
const int INF = 1e9;

vector<pair<int, int>> adj[MAXN];
bool visited[MAXN];
int dist[MAXN];

void prim(int start) {
    // initialize distances to infinity
    for (int i = 0; i < MAXN; i++) {
        dist[i] = INF;
    }

    // create priority queue using a binary heap
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

    // add start vertex to priority queue with distance 0
    pq.push(make_pair(0, start));
    dist[start] = 0;

    while (!pq.empty()) {
        // extract minimum distance vertex from priority queue
        int u = pq.top().second;
        pq.pop();

        if (visited[u]) {
            continue;
        }

        // mark vertex as visited
        visited[u] = true;

        // update distances of adjacent vertices
        for (auto v : adj[u]) {
            int weight = v.second;
            if (!visited[v.first] && weight < dist[v.first]) {
                dist[v.first] = weight;
                pq.push(make_pair(weight, v.first));
            }
        }
    }
}
```

```
int main() {
    int n, m;
    cin >> n >> m;

    // read in graph edges
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back(make_pair(v, w));
        adj[v].push_back(make_pair(u, w));
    }

    // run Prim's algorithm starting from vertex 1
    prim(1);

    // calculate minimum cost
    int minCost = 0;
    for (int i = 1; i <= n; i++) {
        minCost += dist[i];
    }

    cout << minCost << endl;

    return 0;
}
```

2) Kruskal's algorithm using Min Heap

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

const int MAXN = 1e5 + 5;

struct edge {
    int u, v, weight;
    bool operator<(const edge& other) const {
        return weight > other.weight;
    }
};

int parent[MAXN];
int size[MAXN];

void make_set(int v) {
    parent[v] = v;
    size[v] = 1;
}

int find_set(int v) {
    if (v == parent[v]) {
        return v;
```

```
    }
    return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (size[a] < size[b]) {
            swap(a, b);
        }
        parent[b] = a;
        size[a] += size[b];
    }
}

vector<edge> kruskal(vector<edge> edges, int n) {
    vector<edge> mst;

    // initialize parent and size arrays for disjoint sets
    for (int i = 1; i <= n; i++) {
        make_set(i);
    }

    // create priority queue using a min heap
    priority_queue<edge> pq;
    for (auto e : edges) {
        pq.push(e);
    }

    while (!pq.empty() && mst.size() < n - 1) {
        // extract minimum weight edge from priority queue
        edge e = pq.top();
        pq.pop();

        // check if endpoints are in different sets
        if (find_set(e.u) != find_set(e.v)) {
            mst.push_back(e);
            union_sets(e.u, e.v);
        }
    }

    return mst;
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<edge> edges;
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        edges.push_back({u, v, w});
    }
```

```
    vector<edge> mst = kruskal(edges, n);

    int minCost = 0;
    for (auto e : mst) {
        minCost += e.weight;
    }

    cout << minCost << endl;

    return 0;
}
```