Name of Experiment:

Introduction to Minimum Spanning Tree (MST), Prim's and Kruskal's Algorithm

Objectives:

1. To understand the concept of minimum spanning tree.
2. To understand the implementation of the Algorithm to find the minimum spanning tree in weighted graph.
3. To learn how Prim's Algorithm works.
4. To learn how Krushkal's Algorithm works.
5. To implement the algorithm for solving problem in real life.
6. To analyze the time and space complexity for Prims and Krushkals Algorithm.

Introduction:

Minimum spanning tree is a subset of a connected undirected graph with weight, that connects all the vertices together without any cycle. And with the minimum total cost. There are two algorithms for finding minimum spanning tree ,

1. Prims Algorithm :

It starts from any vertex and keep adding the smallest edge that connect a vertex inside the MST with a vertex outside of it. It grows one vertex at a time until all are included.

2. Kruskals Algorithm :

It takes all the edges and sort them by their weight. Then it starts taking edge one by one from smallest to biggest, and add it if it doesn't make any cycle. It uses disjoint set (union-find) to check cycle.

Both algorithm gives the same MST but in different time .

Algorithm (Prim's Algorithm):

Step 1: Pick any vertex as starting point.
Step 2: Mark all vertex as not visited and set key of all as infinity except the start one (key=0).
Step 3: Choose vertex u with smallest key value which is not visited.
Step 4: Include u in MST and update the key of its all neighbor vertices if their edge weight is smaller.
Step 5: Repeat until all vertices are visited.

Algorithm (Kruskal's Algorithm):

Step 1: Sort all edges according to their weight.

Step 2: Start adding edges from smallest weight one.

Step 3: If adding this edge doesn't form a cycle (checked by union-find), include it in MST.

Step 4: Stop when you have V–1 edges in MST.

Code:

Prim's Algorithm:

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

int primsMST(int V, vector<pair<int, int>> adj[]) {
   priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
   vector<bool> visited(V, false);
   int cost = 0;
   pq.push({0, 0});

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

      if (visited[u]) continue;

      visited[u] = true;
      cost += w;

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

         if (!visited[v]) {
            pq.push({weight, v});
         }
      }
   }

   return cost;
}

int main() {
```

```
    int V, E;
    cin >> V >> E;
    vector<pair<int, int>> adj[V];
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }

    int mstCost = primsMST(V, adj);
    cout << "Cost: " << mstCost << endl;

    return 0;
}
```

Krushkal's Algorithm:

```
#include <bits/stdc++.h>
using namespace std;
class DSU {
    vector<int> parent, rank;

public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n);
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 1;
        }
    }

    int find(int i) {
        return (parent[i] == i) ? i : (parent[i] = find(parent[i]));
    }

    void unite(int x, int y) {
        int s1 = find(x), s2 = find(y);
        if (s1 != s2) {
            if (rank[s1] < rank[s2]) parent[s1] = s2;
            else if (rank[s1] > rank[s2]) parent[s2] = s1;
            else parent[s2] = s1, rank[s1]++;
        }
```

```
    }
};
bool comparator(vector<int> &a,vector<int> &b){
  return a[2] < b[2];
}
int kruskalsMST(int V, vector<vector<int>> &edges) {

    sort(edges.begin(), edges.end(),comparator);

    DSU dsu(V);
    int cost = 0, count = 0;

    for (auto &e : edges) {
        int x = e[0], y = e[1], w = e[2];
        if (dsu.find(x) != dsu.find(y)) {
            dsu.unite(x, y);
            cost += w;
            if (++count == V - 1) break;
        }
    }
    return cost;
}

int main() {
    int V, E;
    cin >> V >> E;
    vector<vector<int>> edges;

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

    int mstCost = kruskalsMST(V, edges);
    cout << "Cost: " << mstCost << endl;

    return 0;
}
```

Time Complexity:

Prim's Algorithm:
Using adjacency matrix : $O(V^2)$
Using min-heap and adjacency list : O(E log V)

Kruskal's Algorithm:
O(E log E) for sorting edges

Space Complexity:

Prim's : O(V + E)

Kruskal's: O(V + E)

Conclusion:

In this lab , we have implemented the Prim's Algorithm and Kruskhal's Algorithm using CPP language , to find the minimum spanning tree in a undirected , weighted graph . Both the algorithm can find MST but in different way, Prims algorithm goes for one by one vertex where Krushkals works with edge, sorting edge. The result using both algorithm is same .But prims algorithm is better for dense graph where krushkals algorithm is better for sparse graph .These algorithm is usefull in solving real life problem like road mapping, networks or internet cable etc.

Reference:

1. Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms*, MIT Press.
2. GeeksforGeeks: https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
3. GeeksforGeeks: https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
4. Class Lecture