****

**NAME: -** Darji Akshatkumar Hiteshbhai

**RollNo: -** 23MCD001

**Branch: -** M.tech-CSE**(Data Science)**

**Subject: -** Complexity Theory & Algorithms

**Practical-5**

**Aim:** Implement Kruskal’s algorithm to find MST using Greedy approach.

**Code for Kruskal’s Algorithm –**

#include <bits/stdc++.h>

using namespace std;

class DisjointSet

{

    vector<int> rank, parent, size;

public:

    DisjointSet(int n)

    {

        rank.resize(n + 1, 0);

        parent.resize(n + 1);

        size.resize(n + 1);

        for (int i = 0; i <= n; i++)

        {

            parent[i] = i;

            size[i] = 1;

        }

    }

    int findUPar(int node)

    {

        if (node == parent[node])

            return node;

        return parent[node] = findUPar(parent[node]);

    }

    void unionByRank(int u, int v)

    {

        int ulp\_u = findUPar(u);

        int ulp\_v = findUPar(v);

        if (ulp\_u == ulp\_v)

            return;

        if (rank[ulp\_u] < rank[ulp\_v])

        {

            parent[ulp\_u] = ulp\_v;

        }

        else if (rank[ulp\_v] < rank[ulp\_u])

        {

            parent[ulp\_v] = ulp\_u;

        }

        else

        {

            parent[ulp\_v] = ulp\_u;

            rank[ulp\_u]++;

        }

    }

    void unionBySize(int u, int v)

    {

        int ulp\_u = findUPar(u);

        int ulp\_v = findUPar(v);

        if (ulp\_u == ulp\_v)

            return;

        if (size[ulp\_u] < size[ulp\_v])

        {

            parent[ulp\_u] = ulp\_v;

            size[ulp\_v] += size[ulp\_u];

        }

        else

        {

            parent[ulp\_v] = ulp\_u;

            size[ulp\_u] += size[ulp\_v];

        }

    }

};

class Solution

{

public:

    // Function to find sum of weights of edges of the Minimum Spanning Tree.

    int spanningTree(int V, vector<vector<int>> adj[])

    {

        vector<pair<int, pair<int, int>>> edges;

        for (int i = 0; i < V; i++)

        {

            for (auto it : adj[i])

            {

                int adjNode = it[0];

                int wt = it[1];

                int node = i;

                edges.push\_back({wt, {node, adjNode}});

            }

        }

        DisjointSet ds(V);

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

        int mstWt = 0;

        // Create a vector to store the selected edges in the MST

        vector<tuple<int, int, int>> selectedEdges;

        for (auto it : edges)

        {

            int wt = it.first;

            int u = it.second.first;

            int v = it.second.second;

            if (ds.findUPar(u) != ds.findUPar(v))

            {

                mstWt += wt;

                ds.unionBySize(u, v);

                selectedEdges.push\_back(make\_tuple(u, v, wt));

            }

        }

        // Print the selected edges with their weights

        cout << "Selected edges for the MST:" << endl;

        for (auto edge : selectedEdges)

        {

            int u, v, wt;

            // tuple reference i have to create to extract

            tie(u, v, wt) = edge;

            cout << "{" << u << "-" << v << ", Weight: " << wt << "}" << endl;

        }

        return mstWt;

    }

};

int main()

{

    int V = 9;

    vector<vector<int>> edges = {{1, 4, 1}, {1, 2, 2}, {2, 3, 3}, {2, 4, 3}, {1, 5, 4}, {3, 4, 5}, {2, 6, 7}, {3, 6, 8}, {4, 5, 9}};

    vector<vector<int>> adj[V];

    for (auto it : edges)

    {

        vector<int> tmp(2);

        tmp[0] = it[1];

        tmp[1] = it[2];

        adj[it[0]].push\_back(tmp);

        tmp[0] = it[0];

        tmp[1] = it[2];

        adj[it[1]].push\_back(tmp);

    }

    Solution obj;

    cout << "MST for : " << "{{1,4,1}, {1,2,2}, {2,3,3}, {2,4,3}, {1,5,4}, {3,4,5}, {2,6,7}, {3,6,8}, {4,5,9}}" << endl;

    int mstWt = obj.spanningTree(V, adj);

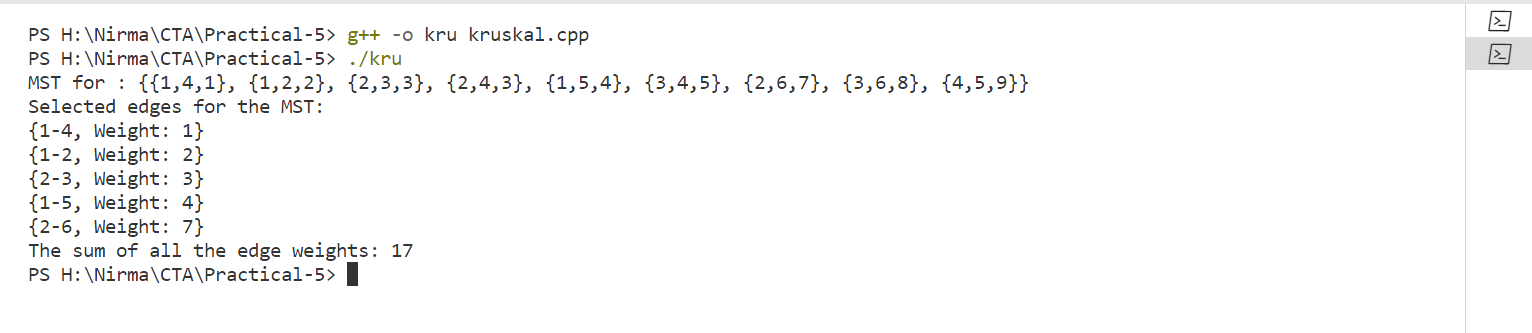
    cout << "The sum of all the edge weights: " << mstWt << endl;

    return 0;

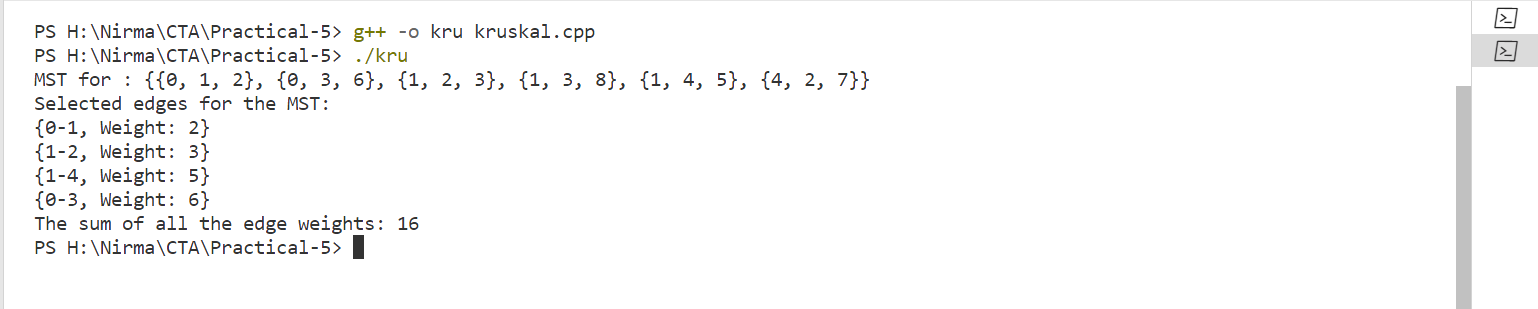
}

**Output –**

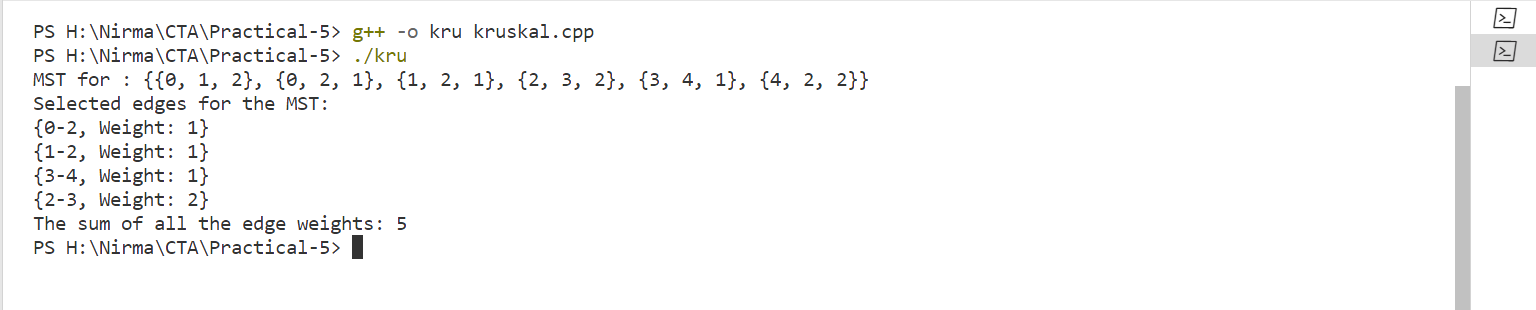
**Test Case – 1**

****

**Test Case – 2**

****

**Test Case – 3**

****