Kruskal in C++

```
#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[])
```

Input

You are given:

```
V = 5;
edges = {
    {0, 1, 2},
    {0, 2, 1},
    {1, 2, 1},
    {2, 3, 2},
    {3, 4, 1},
    {4, 2, 2}
};
```

📦 Step 1: Adjacency List Construction (Undirected Graph)

adj[i] stores {neighbour, weight}:

Node	Adjacents
0	[1, 2], [2, 1]
1	[0, 2], [2, 1]
2	[0, 1], [1, 1], [3, 2], [4, 2]
3	[2, 2], [4, 1]
4	[3, 1], [2, 2]

🧱 Step 2: Edge List Formation

Collected as {weight, {u, v}} (both directions included):

Edge	Format
0-1	{2, {0, 1}}
0-2	{1, {0, 2}}
1-2	{1, {1, 2}}
2-3	{2, {2, 3}}
3-4	{1, {3, 4}}
4-2	{2, {4, 2}}
🔁 duplicates (undirected, so reverse edges too!)	

🔻 Step 3: Sort Edges by Weight

Sorted edges:

```
edges = {
  {1, {0, 2}},
  {1, {1, 2}},
  {1, {3, 4}},
  {2, {0, 1}},
  {2, {2, 3}},
  {2, {4, 2}}
}
```

```
    {
        // 1 - 2 wt = 5
        /// 1 - > (2, 5)
        // 2 -> (1, 5)

        // 5, 1, 2
        // 5, 2, 1
        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;
        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);
            }
        }

        return mstWt;
    }
};

int main() {

    int V = 5;
    vector<vector<int>> edges = {{0, 1, 2}, {0,
2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};
    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;
    int mstWt = obj.spanningTree(V, adj);
    cout << "The sum of all the edge weights: "
<< mstWt << endl;
    return 0;
}
```

🛠 Step 4: Disjoint Set Initialization

- Each node starts as its own parent.
- parent[] = {0, 1, 2, 3, 4}
- size[] = {1, 1, 1, 1, 1}

🔁 Step 5: Process Edges

Edge	Find UParent(u)	Find UParent(v)	Cycle?	Union?	MST Weight
{1, {0, 2}}	0	2	No	Union(0, 2)	1
{1, {1, 2}}	1	0 (from 2)	No	Union(1, 0)	2
{1, {3, 4}}	3	4	No	Union(3, 4)	3
{2, {0, 1}}	0	0	**Yes**	✕ Skip	3
{2, {2, 3}}	0	3	No	Union(0, 3)	5
{2, {4, 2}}	0	0	**Yes**	✕ Skip	5

✅ Final MST Weight

The sum of all the edge weights: 5

🧠 Disjoint Set Status (Final)

Node	Parent
0	0
1	0
2	0
3	0
4	0

All nodes are connected — ✅ valid spanning tree.

Output:-

```
The sum of all the edge weights: 5
```