Kruskal in C++

```cpp
#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[])
   {
      // 1 - 2 wt = 5
```

The graph represented by `edges` is:

```
0 -- 1 -- 2
|    |    |
1    1    2
|    |    |
3 -- 4 -- 5
```

Step 1: Create the Edge List

The adjacency list `adj[]` is converted into an edge list `edges[]`, which is a vector of pairs representing the edges:

```
edges = [(2, (0, 1)), (1, (0, 2)),
(1, (1, 2)), (2, (2, 3)), (1, (3,
4)), (2, (4, 2))]
```

Step 2: Sort the Edges by Weight

The edges are sorted in ascending order by their weights:

```
edges = [(1, (0, 2)), (1, (1, 2)),
(1, (3, 4)), (2, (0, 1)), (2, (2,
3)), (2, (4, 2))]
```

Step 3: Apply Kruskal's Algorithm with Disjoint Set

- Initialize the Disjoint Set for 5 vertices: `parent = [0, 1, 2, 3, 4]`, `size = [1, 1, 1, 1, 1]`.
- Process each edge:
  1. **Edge (0, 2, 1)**:
     - `find(0) != find(2)`, so add the edge to MST.
     - `parent[2] = 0`, `size[0] = 2`.
     - Add `1` to `mstWt`. Now `mstWt = 1`.
  2. **Edge (1, 2, 1)**:
     - `find(1) != find(2)`, so add the edge to MST.
     - `parent[2] = 1`, `size[1] = 2`.
     - Add `1` to `mstWt`. Now `mstWt = 2`.
  3. **Edge (3, 4, 1)**:
     - `find(3) != find(4)`, so add the edge to MST.

```
        /// 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;
}
```

   - `parent[4] = 3`, `size[3] = 2`.
   - Add `1` to `mstWt`. Now `mstWt = 3`.
4. **Edge (0, 1, 2)**:
   - `find(0) == find(1)`, so ignore this edge (it forms a cycle).
5. **Edge (2, 3, 2)**:
   - `find(2) != find(3)`, so add the edge to MST.
   - `parent[3] = 2`, `size[2] = 4`.
   - Add `2` to `mstWt`. Now `mstWt = 5`.
6. **Edge (4, 2, 2)**:
   - `find(4) == find(2)`, so ignore this edge (it forms a cycle).

Step 4: Return the MST Weight

The total weight of the Minimum Spanning Tree is `5`.

Output:-
The sum of all the edge weights: 5