```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

#define edge pair<int, int>

class Graph
{
private:
    vector<pair<int, edge>> G; // graph
    vector<pair<int, edge>> T; // mst
    int *parent;
    int V; // number of vertices/nodes in graph
public:
    Graph(int V);
    void AddWeightedEdge(int u, int v, int w);
    int find_set(int i);
    void union_set(int u, int v);
    void kruskal();
    void print();
};
Graph::Graph(int V)
{
    parent = new int[V];

    //i 0 1 2 3 4 5
    //parent[i] 0 1 2 3 4 5
    for (int i = 0; i < V; i++)
        parent[i] = i;

    G.clear();
    T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w)
{
    G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i)
{
    // If i is the parent of itself
    if (i == parent[i])
        return i;
    else
        // Else if i is not the parent of itself
        // Then i is not the representative of his set,
        // so we recursively call Find on its parent
        return find_set(parent[i]);
}

void Graph::union_set(int u, int v)
{
    parent[u] = parent[v];
}
void Graph::kruskal()
{
    int i, uRep, vRep;
    sort(G.begin(), G.end()); // increasing weight
    for (i = 0; i < G.size(); i++)
    {
        uRep = find_set(G[i].second.first);
        vRep = find_set(G[i].second.second);
        if (uRep != vRep)
        {
            T.push_back(G[i]); // add to tree
            union_set(uRep, vRep);
        }
    }
}
void Graph::print()
{
    cout << "Edge :"
         << " Weight" << endl;
    for (int i = 0; i < T.size(); i++)
    {
        cout << T[i].second.first << " - " << T[i].second.second << "
: "
             << T[i].first;
        cout << endl;
    }
}
int main()
{
    Graph g(6);
    g.AddWeightedEdge(0, 1, 4);
    g.AddWeightedEdge(0, 2, 4);
    g.AddWeightedEdge(1, 2, 2);
    g.AddWeightedEdge(1, 0, 4);
    g.AddWeightedEdge(2, 0, 4);
    g.AddWeightedEdge(2, 1, 2);
    g.AddWeightedEdge(2, 3, 3);
    g.AddWeightedEdge(2, 5, 2);
    g.AddWeightedEdge(2, 4, 4);
    g.AddWeightedEdge(3, 2, 3);
    g.AddWeightedEdge(3, 4, 3);
    g.AddWeightedEdge(4, 2, 4);
    g.AddWeightedEdge(4, 3, 3);
    g.AddWeightedEdge(5, 2, 2);
    g.AddWeightedEdge(5, 4, 3);
    g.kruskal();
    g.print();
    return 0;
}
```