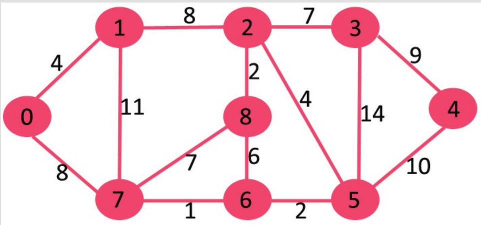
Kruskal's Minimum Spanning Tree using STL in C++

Given an undirected, connected and weighted graph, find **M**inimum **S**panning **T**ree (MST) of the graph using Kruskal's algorithm.




```
Input :   Graph as an array of edges
Output :  Edges of MST are
          6 - 7
          2 - 8
          5 - 6
          0 - 1
          2 - 5
          2 - 3
          0 - 7
          3 - 4

          Weight of MST is 37

Note :  There are two possible MSTs, the other
        MST includes edge 1-2 in place of 0-7.
```

We have discussed below Kruskal's MST implementations.

Greedy Algorithms | Set 2 (Kruskal's Minimum Spanning Tree Algorithm)

Below are the steps for finding MST using Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Here are some key points which will be useful for us in implementing the Kruskal's algorithm using STL.

1. Use a vector of edges which consist of all the edges in the graph and each item of a vector will contain 3 parameters: source, destination and the cost of an edge between the source and destination.

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

   Here in the outer pair (i.e pair<int,pair<int,int> >) the first element corresponds to the cost of a edge while the second element is itself a pair, and it contains two vertices of edge.

2. Use the inbuilt std::sort to sort the edges in the non-decreasing order; by default the sort function sort in non-decreasing order.
3. We use the Union Find Algorithm to check if it the current edge forms a cycle if it is added in the current MST. If yes discard it, else include it (union).

Pseudo Code:

```
// Initialize result
mst_weight = 0

// Create V single item sets
for each vertex v
 parent[v] = v;
 rank[v] = 0;

Sort all edges into non decreasing
order by weight w

for each (u, v) taken from the sorted list  E
    do if FIND-SET(u) != FIND-SET(v)
        print edge(u, v)
        mst_weight += weight of edge(u, v)
        UNION(u, v)
```

Below is C++ implementation of above algorithm.

```
// C++ program for Kruskal's algorithm to find Minimum
// Spanning Tree of a given connected, undirected and
// weighted graph
#include<bits/stdc++.h>
using namespace std;

// Creating shortcut for an integer pair
typedef  pair<int, int> iPair;

// Structure to represent a graph
struct Graph
{
    int V, E;
    vector< pair<int, iPair> > edges;

    // Constructor
    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }

    // Utility function to add an edge
    void addEdge(int u, int v, int w)
    {
        edges.push_back({w, {u, v}});
    }

    // Function to find MST using Kruskal's
    // MST algorithm
    int kruskalMST();
};

// To represent Disjoint Sets
struct DisjointSets
{
    int *parent, *rnk;
    int n;

    // Constructor.
    DisjointSets(int n)
    {
        // Allocate memory
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];

        // Initially, all vertices are in
        // different sets and have rank 0.
        for (int i = 0; i <= n; i++)
        {
            rnk[i] = 0;
```

```
            //every element is parent of itself
            parent[i] = i;
        }
    }

    // Find the parent of a node 'u'
    // Path Compression
    int find(int u)
    {
        /* Make the parent of the nodes in the path
           from u--> parent[u] point to parent[u] */
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }

    // Union by rank
    void merge(int x, int y)
    {
        x = find(x), y = find(y);

        /* Make tree with smaller height
           a subtree of the other tree  */
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else // If rnk[x] <= rnk[y]
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
    }
};

 /* Functions returns weight of the MST*/

int Graph::kruskalMST()
{
    int mst_wt = 0; // Initialize result

    // Sort edges in increasing order on basis of cost
    sort(edges.begin(), edges.end());

    // Create disjoint sets
    DisjointSets ds(V);

    // Iterate through all sorted edges
    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
        int set_v = ds.find(v);

        // Check if the selected edge is creating
        // a cycle or not (Cycle is created if u
        // and v belong to same set)
        if (set_u != set_v)
        {
            // Current edge will be in the MST
            // so print it
            cout << u << " - " << v << endl;

            // Update MST weight
            mst_wt += it->first;

            // Merge two sets
            ds.merge(set_u, set_v);
        }
    }
```

```
    return mst_wt;
}

// Driver program to test above functions
int main()
{
    /* Let us create above shown weighted
       and unidrected graph */
    int V = 9, E = 14;
    Graph g(V, E);

    //  making above shown graph
    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    cout << "Edges of MST are \n";
    int mst_wt = g.kruskalMST();

    cout << "\nWeight of MST is " << mst_wt;

    return 0;
}
```

Output :

```
Edges of MST are
         6 - 7
         2 - 8
         5 - 6
         0 - 1
         2 - 5
         2 - 3
         0 - 7
         3 - 4

         Weight of MST is 37
```

Optimization:
The above code can be optimized to stop the main loop of Kruskal when number of selected edges become V-1. We know that MST has V-1 edges and there is no point iterating after V-1 edges are selected. We have not added this optimization to keep code simple.

References:
Introduction to Algorithms by Cormen Leiserson Rivest and Stein(CLRS) 3

Time complexity and step by step illustration are discussed in previous post on Kruskal's algorithm.