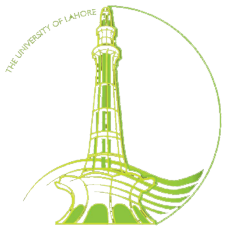

THE UNIVERSITY OF LAHORE
ISLAMABAD CAMPUS

Data Structure and Algorithms

Lab Report

Name:	Asad Abbas
Registration #:	CSU-F16-109
Lab Report #:	12
Dated:	28-06-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 12
Minimum Spanning Tree

Objective

Software Tool
1. Dev C++

1 Theory

Below are the steps for finding MST using Kruskals algorithm.

Sort all the edges in non-decreasing order of their weight.
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.
Here are some key points which will be useful for us in implementing the Kruskals algorithm using STL.
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.
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.
Use the inbuilt std::sort to sort the edges in the non-decreasing order; by default the sort function sort in non-decreasing order. 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).

2 Program

```
#include<bits/stdc++.h>
using namespace std;
typedef  pair<int, int> iPair;
```

```
struct Graph
{
    int V, E;
    vector< pair<int, iPair> > edges;
    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }
    void addEdge(int u, int v, int w)
    {
        edges.push_back({w, {u, v}});
    }
    int kruskalMST();
};
struct DisjointSets
{
    int *parent, *rnk;
    int n;.
    DisjointSets(int n)
    {
        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];.
        for (int i = 0; i <= n; i++)
        {
            rnk[i] = 0;
            parent[i] = i;
        }
    }
    int find(int u)
    {
        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }
    void merge(int x, int y)
    {
        x = find(x), y = find(y);
```

```
        if (rnk[x] > rnk[y])
            parent[y] = x;
        else // If rnk[x] <= rnk[y]
            parent[x] = y;

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

int Graph::kruskalMST()
{
    int mst_wt = 0;

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

    DisjointSets ds(V);

    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);
        if (set_u != set_v)
        {
            cout << u << " - " << v << endl;
            mst_wt += it->first;
            ds.merge(set_u, set_v);
        }
    }

    return mst_wt;
}
int main()
{
    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;
}
```