DATA STRUCTURE
(CS13217)

Lab Report

Name:	Zain ul abideen
Registration #:	SEU-F16-133
Lab Report #:	11
Dated:	28-06-2018
Submitted To:	Mr. Usman Ahmed

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

Experiment # 1
Implementing the kruskal algorothms,

Objective
To understand and implement the kruskal algorithms.

Software Tool
1.
dev c++

1 Theory

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.[1] It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.[1] This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, create a graph F (a set of trees), where each vertex in the graph is a separate tree create a set S containing all the edges in the graph while S is nonempty and F is not yet spanning remove an edge with minimum weight from S if the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

.

2 Task

2.1 procedure: Task 1

```
#include<bits/stdc++.h>
using namespace std;


typedef  pair<int, int> iPair;


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 unidirected 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;
}
        
```

3 Conclusion

In this lab we perform the kruskal algorithm sort edge by weighted smallest first for each edge is in oeder.