Subject: Data Structures and Algorithms Lab

Assignment No. 07

Roll No: **21118**

Date: 26-May-2021

Problem Statement:

You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

Objectives:

- To understand about graph data structure
- Learn about various graph representations in computer memory
- To Lean about two most efficient algorithms to calculate MST
  - Prim's algorithms
  - Kruskal's algorithm

Outcomes:

- Implement adjacency matrix presentation of graph
- Implement Prim's algorithm and Kruskal's algorithm to find minimum spanning tree in a graph

Hardware Requirements:

- Manufacturer: Acer
- System Type: x-64 based PC
- Memory and Processor: Intel core i5-8265U@1.60GHz, 8 GB DDR4 RAM, 512 GB SSD

Software Requirements:

- Operating System: Windows-10 (Home Edition)
- IDE: eclipse (2020 – Edition)
- Compiler: GCC (version 6.3.0)

Theory:

1. Graphs are mathematical abstractions that are useful for solving many types of problems in computer science.

2. Any graph finite number of vertices and edges.
3. There are two major ways to represent graphs in computer memory. One can use either one depends on situation and problem.
    a. Adjacency Matrix Representation – It is time efficient to check edge between pair of vertices but it is space inefficient
    b. Adjacency List Representation – It is space efficient but not time efficient when we want to check edge between pair of vertices.
4. Minimum spanning tree: A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.
5. There are two popular algorithms to find mst in graph:
    a. Prim's algorithm: It is a greedy algorithm. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.
    b. Kruskal's algorithm: It finds a minimum spanning forest of an undirected edge-weighted graph. If the graph is connected, it finds a minimum spanning tree. For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component. It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.

ADT:

```
class Edge {
private:
	int u, v, wt;
public:
	Edge() {u=0, v=0, wt=0;}
	Edge(int x, int y, int w);
	friend class mst;
};

class mst {
private:
	int vertices, edges;
	int** adj_list;
	int* par; // to keep track of root node of mst in disjoint mst's (kruskal's algo)

	int find_mst(int u);
	void union_mst(int u1, int u2);
public:
	mst(int n, int m);
	void printGraph();
	void addEdge(int u, int v, int w);
	int prims_mst_wt();
```

```
        int kruskals_mst_wt();
        ~mst();
};
```

<u>Pseudocodes:</u>

Prims algorithm to find minimum spanning tree:

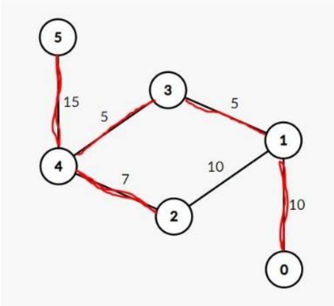
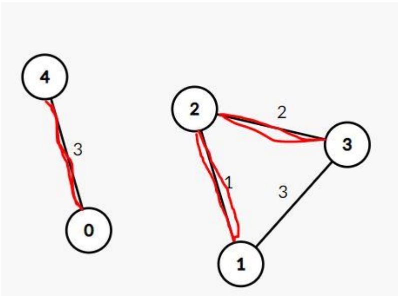
```
    Procedure Prims_MST(G, V)
        mst_vert = empty set, rem_vert = {0, 1, 2, .. , V-1}
        vert_wt(V), mst_wt = 0.
        for each vertex v of graph G:
            mark vert_wt(v) as infinity
        start with any random vertex v
        mark vert_wt[v] as 0.
        add vertex v to mst_vert.
        remove vertex v from rem_vert.
        loop until rem_vert is not empty:
            Find and remove a vertex v from rem_vert having the minimum
            possible value of vert_wt.
            add v to mst_vert.
            add vert_wt of v to mst_wt.
            for all adjacent vertices v_adj of v:
                if v_adj is present in rem_vert and vert_wt of v_adj is greater
                than cost(v, v_adj):
                    update vert_wt of v_adj to cost(v, v_adj)
        return mst_wt;
```

Kruskal's algorithm to find minimum spanning tree:

```
    Procedure Kruskal(G) is
        F:= ∅, mst_wt=0
        for each v ∈ G.V do
            MAKE-SET(v)
        for each (u, v) in G.E ordered by weight(u, v), increasing do
            if FIND-SET(u) ≠ FIND-SET(v) then
                F:= F ∪ {(u, v)} ∪ {(v, u)}
                UNION(FIND-SET(u), FIND-SET(v))
                Add weight to mst_wt
        return mst_wt
```

<u>Testcases</u>: (0-based indexing)

Testcase No.	Input	Expected Output	Actual Output	Verdict
1	Vertices: 6 Edges: 6 0 1 10	Prims algo: 42	Prims algo: 42	Passed

<table>
<tr><th></th><th></th><th></th><th></th><th></th></tr>
<tr><td></td><td>1 3 5
3 4 5
4 5 15
1 2 10
2 4 7

</td><td>Kruskal’s
algo: 42</td><td>Kruskal’s
algo: 42</td><td></td></tr>
<tr><td>2</td><td>Vertices: 5
Edges: 4
0 4 3
1 2 1
2 3 2
3 1 3

</td><td>Prim’s
Algo: 6
Kruskals
Algo: 6</td><td>Prim’s
Algo: 6
Kruskals
Algo: 6</td><td>Passed</td></tr>
</table>

Analysis of algorithms:

No.	Algorithm	Time Complexity	Space Complexity
1	Prims algorithm	O(n^2) n: vertices in graph	O(n) for storing set of vertices
2	Kruskal’s algorithm	O(n*m) n: vertices in graph m: edges in graph (find and union operations are naïve i.e. without any optimizations)	O(m) for storing edges

Applications:

Graph data structure has wide range of applications. Applications of minimum spanning tree are mentioned below.

- Cluster Analysis.
- Real-time face tracking and verification (i.e. locating human faces in a video stream).

- Protocols in computer science to avoid network cycles.
- Entropy based image registration.
- Max bottleneck paths.
- Dithering (adding white noise to a digital recording in order to reduce distortion).

Conclusion:

I have learned about representation of graph data structure in computer memory and about algorithms to calculate MST in graph. Also implemented two algorithms: Prim's algorithm and Kruskal's algorithm.