Assignment 5

Title: To write a program for Graph creation and find its minimum cost using Prim's or Kruskal's algorithm.

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 Prims or Kruskal using adjacency matrix.

Learning Objectives:

- To understand concept of graph and minimum cost spanning tree.
- To understand minimum cost spanning tree algorithms.

Learning Outcomes:

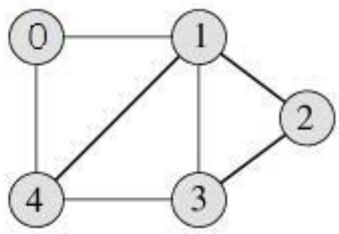
After successful completion of this assignment, students will be able to

- Implement graph using adjacency matrix or adjacency list.
- Create minimum cost spanning tree using Prim's or Kruskal's algorithm.

Concepts related Theory:

- **Representation of Graph**

Following is an example undirected graph with 5 vertices.



➢ **Using Adjacency Matrix**

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from

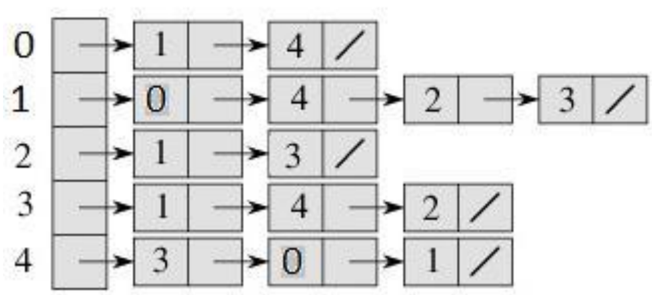
vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

- **Using Adjacency list**

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



- **Minimum Spanning Tree:**

Give a graph G = (V, E), the minimum spanning tree (MST) is a weighted graph G' = (V, E')

such that:

- E' Í E
- G' is connected
- G' has the minimum cost

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. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of the minimum spanning trees for its connected components. There are quite a few use cases for minimum spanning trees. One example would be a telecommunications company which is trying to lay out cables in new neighbourhood.

Prim's Algorithm

Step 1: Select any vertex

Step 2: Select the shortest edge connected to that vertex.

Step 3: Select the shortest edge connected to any vertex already connected

Step 4: Repeat step 3 until all vertices have been connected

Class Definition:

```
class node {
```

```
        string data;
        node* next;
public:
        node(string s) {
                data = s;
                next = NULL;
        }
        friend class Graph;
};

class Graph {
        int **G;
        int n;
        node* head;
public:
        Graph() {
                head = NULL;
                cout << "Enter number of vertices: ";
                cin >> n;
                G = new int*[n];
                for (int i = 0; i < n; i++) {
                        G[i] = new int[n];
                }
        }
        void read();
        int posn(string);
```

```
    void add(string);
    void prim();
    void temp();
};
```

Pseudo Codes:

```
int Graph::posn(string s) {
    node* p = head;
    int cnt = 0;
    while (p != NULL && p->data != s) {
        cnt++;
        p = p->next;
    }
    return cnt;
}

void Graph::add(string s) {
    if (head == NULL) {
        head = new node(s);
        return;
    }
    node* p = head;
    node* q;
    int flag = 0;
    while (p != NULL && p->data != s) {
        q = p;
        flag = 1;
```

```
			p = p->next;
		}
		if (flag == 1) {
			q->next = new node(s);
		}
}

void Graph::read() {
		string u, v;
		int w;
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++) {
				G[i][j] = 0;
			}
		}
		while (1) {
			cout << "Source: ";
			cin.ignore(1);
			getline(cin, u);
			if (u == "stop") {
				return;
			}
			add(u);
			cout << "Target: ";
			getline(cin, v);
			add(v);
```

```
        cout << "Cost: ";
        cin >> w;
        G[posn(u)][posn(v)] = w;
        G[posn(v)][posn(u)] = w;
    }
}

void Graph::prim() {
    int mincost = 0;
    int *visited, *dist, *from;
    visited = new int[n];
    dist = new int[n];
    from = new int[n];
    visited[0] = 1;
    dist[0] = inf;
    from[0] = 0;
    int **cost = new int*[n];
    for (int i = 0; i < n; i++) {
        cost[i] = new int[n];
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (G[i][j] == 0) {
                cost[i][j] = inf;
            } else {
                cost[i][j] = G[i][j];
```

```
        }
    }
}
for (int i = 1; i < n; i++) {
    visited[i] = 0;
    dist[i] = cost[0][i];
    from[i] = 0;
}
int mindist;
int ne = n - 1;
int v;
while (ne > 0) {
    mindist = inf;
    for (int i = 0; i < n; i++) {
        if (visited[i] == 0 && mindist > dist[i]) {
            mindist = dist[i];
            v = i;
        }
    }
    visited[v] = 1;
    int u = from[v];
    cout << u << "->" << v << endl;
    mincost += cost[u][v];
    for (int i = 0; i < n; i++) {
        if (visited[i] == 0 && dist[i] > cost[v][i]) {
            dist[i] = cost[v][i];
```

```
                    from[i] = v;
                }
            }
            ne--;
        }
        cout << "Minimum cost: " << mincost;
}
```

Test Case:

Enter number of vertices: 5

Source: Mumbai

Target: Pune

Cost: 10

Source: Mumbai

Target: Delhi

Cost: 20

Source: Mumbai

Target: Chennai

Cost: 25

Source: Delhi

Target: Chennai

Cost: 8

Source: Chennai

Target: Kolkatta

Cost: 12

Source: Delhi

Target: Kolkatta

Cost: 18

Source: Pune

Target: Kolkatta

Cost: 27

Source: stop

Mumbai->Pune

Mumbai->Delhi

Delhi->Chennai

Chennai->Kolkatta

Minimum cost: 50

Conclusion:

We have successfully calculated total minimum cost of graph using minimum spanning tree algorithm.