**Practical 14**

**Name: Shantanu Sethi**

**Roll no. 163**

**Aim: To implement minimum spanning tree**

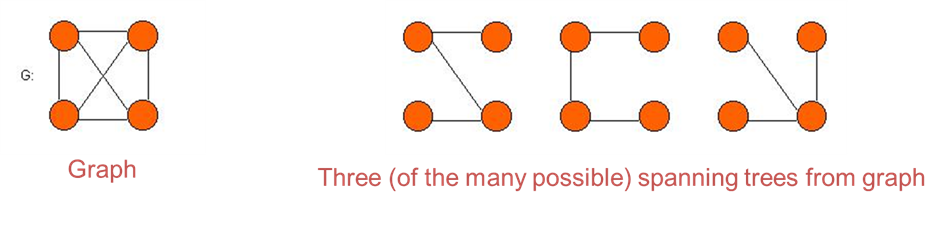
**Objectives:**

1. To create minimum spanning tree using Prim's algorithm and Kruskal's algorithm.

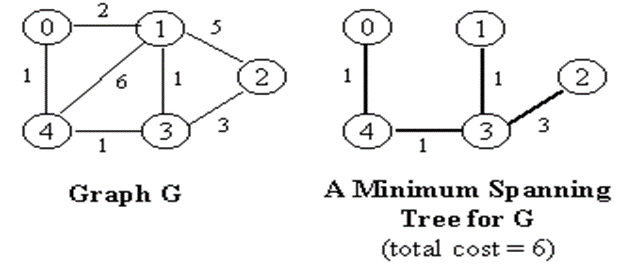
**Theory:**

• **Spanning tree:** A spanning tree of a connected graph is the smallest set of edges such that all nodes of the graph are connected. Addition of one more edge will result into a cycle.

For example:



• **Minimum spanning tree:** A minimum spanning tree T of an graph G is a subgraph of G that connects all the vertices in G at the lowest total cost.



• If weights of edges in a network are unique, then there will be a unique MST. If there are duplicate weights, there can be more than one MST’s.

• MST is used as one of the most important tools to analyze computer networks (e.g. cabling, network load capacity, optimal flow).

**Prim's Algorithm:**

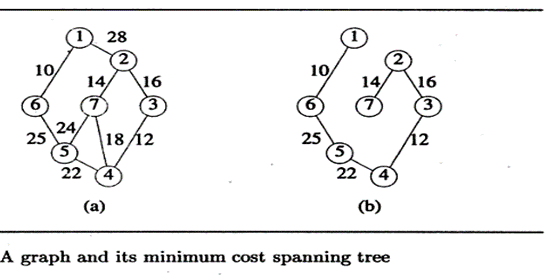
• Include an edge of minimum cost in your Spanning Tree

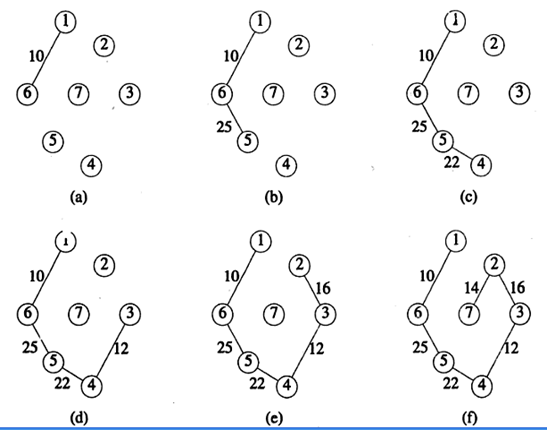
• Repeatedly include the next edge (u, v)

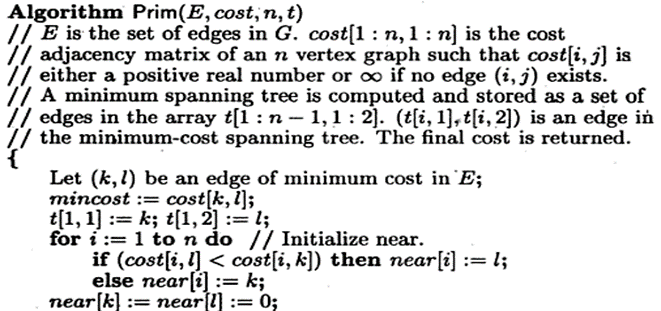
– u should be in the tree , v should not be.

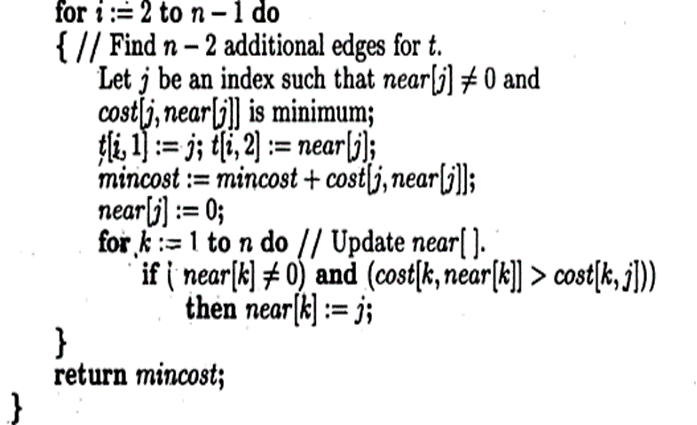
– Cost of (u, v) is minimum among all alternative edges (i.e., all edges connected from any vertex/node in the tree built so far).

• The algorithm stops when all the nodes have been reached









**Program:**

package javaAssignment2;

import java.util.Scanner;

public class MST {

private int n;

private int[][] cost = new int[10][10];

private int[] near1 = new int[10];

private int[][] t = new int[10][3];

private int mincost;

public void get() {

Scanner sc = new Scanner(System.*in*);

System.*out*.println("184-ABHINAV SINGH");

System.*out*.print("Enter the number of vertices: ");

n = sc.nextInt();

System.*out*.println("Enter the Adjacency matrix");

int minEdge = 9999;

int k = 0, l = 0;

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= n; j++) {

cost[i][j] = sc.nextInt();

if (cost[i][j] == 0) {

cost[i][j] = 9999;

}

else if (cost[i][j] < minEdge && i != j) {

minEdge = cost[i][j];

k = i;

l = j;

}

}

}

t[1][1] = k;

t[1][2] = l;

mincost = cost[k][l];

for (int i = 1; i <= n; i++) {

if (cost[i][l] < cost[i][k]) {

near1[i] = l;

}

else {

near1[i] = k;

}

}

near1[k] = near1[l] = 0;

}

public void prim() {

for (int i = 2; i <= n - 1; i++) {

int minj = 9999;

int jindex = 0;

for (int j = 1; j <= n; j++) {

if (near1[j] != 0 && cost[j][near1[j]] < minj) {

minj = cost[j][near1[j]];

jindex = j;

}

}

t[i][1] = jindex;

t[i][2] = near1[jindex];

mincost += cost[jindex][near1[jindex]];

near1[jindex] = 0;

for (int k = 1; k <= n; k++) {

if (near1[k] != 0 && cost[k][near1[k]] > cost[k][jindex]) {

near1[k] = jindex;

}

}

}

System.*out*.println("\nMinimum Cost of MST: " + mincost);

}

public void display() {

System.*out*.println("\nMinimum Spanning Tree Path:");

System.*out*.print(t[1][1] + " -> " + t[1][2]);

for (int i = 2; i <= n - 1; i++) {

System.*out*.print(" -> " + t[i][1]);

}

System.*out*.println();

}

public static void main(String[] args) {

MST graph = new MST();

graph.get();

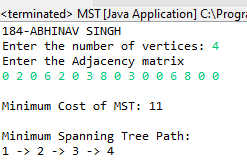
graph.prim();

graph.display();

}

}

**OutPut:**



**Conclusion**: Successfully implemented different minimum spanning tree algorithm