IARE

**IARE**
**INSTITUTE OF AERONAUTICAL ENGINEERING**
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

# LABORATORY WORK BOOK

Name of the Student : RAGHERLA SANTHOSH

| Roll Number | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 9 | 5 | 1 | A | 1 | 2 | C | 3 |

Class : IT-B Semester : 03

Course Code : AGSD11 Course Name : PS laboratory

Name of the Course Faculty : Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 13 Week Number : 13 Date : 26/11/2024

| S. No. | Exercise Number | EXERCISE NAME | MARKS AWARDED | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Aim/ Preparation | Algorithm / Procedure | Performance in the Lab | Source Code | Program Execution | Viva - Voce | Total |
| | | | | | | Calculations and Graphs | Results and Error Analysis | | |
| | | | 4 | 4 | | 4 | 4 | 4 | 20 |
| 1 | 13.1 | Kruskal's Algorithm | 4 | 4 | - | 4 | 4 | 4 | 20 |
| 2 | 13.2 | Prim's Algorithm | | | | | | | |
| 3 | 13.3 | Total No. of Spanning Trees In A Graph | | | | | | | |
| 4 | 13.4 | Minimum Product Spanning Tree | | | | | | | |
| 5 | 13.5 | Reverse Delete Algorithm for MST. | | | | | | | |
| 6 | | | | | | | | | |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |
| 9 | | | | | | | | | |
| 10 | | | | | | | | | |
| 11 | | | | | | | | | |
| 12 | | | | | | | | | |

Santhosh

Signature of the Student

Signature of the Faculty

# 13. Minimum Spanning Tree (MST).

## 13.1 Kruskal's Algorithm :-

AIM :- Write a Program using the Kruskal's Algorithm.

PROGRAM :-

```
import java.util.*;
Public class Graph {
  class Edge implements Comparable<Edge> {
    int src, dest, weight;
    Public Edge (int src, int dest, int weight) {
      this.src = src;
      this.dest = dest;
      this.weight = weight;
    }
    Public int compareTo (Edge compareEdge) {
      return this.weight - compareEdge.weight;
    }
  }
  Private int V;
```

```
Private List<Edge> edges;

Public Graph(int vertices) {
  this.V = vertices;
  this.edges = new ArrayList<>();
}

Public void addEdge(int u, int v, int w) {
  edges.add(new Edge(u, v, w));
}

Private int find(int[] Parent, int i) {
  if (Parent[i] != i) {
    Parent[i] = find(Parent, Parent[i]);
  }
  return Parent[i];
}

Private void union(int[] Parent, int[] rank, int x,
                   int y) {
  int rootX = find(Parent, x);
  int rootY = find(Parent, y);
  if (rank[rootX] < rank[rootY]) {
    Parent[rootX] = rootY;
```

```
    } else if (rank[rootX] > rank[rootY]) {
        parent[rootY] = rootX;
    } else {
        parent[rootY] = rootX;
        rank[rootX]++;
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of vertices: ");
    int vertices = scanner.nextInt();
    System.out.println("Enter the number of edges: ");
    int edges = scanner.nextInt();
    Graph graph = new Graph(vertices);
    System.out.println("Enter edges in the format
                        (src dest weight): ");
    for (int i=0; i< edges; i++) {
        int src = scanner.nextInt();
        int dest = scanner.nextInt();
```

```
        int weight = scanner.nextInt();
        graph.addEdge(src, dest, weight);
    }
    graph.KruskalMST();
  }
}
```

OUTPUT :-

```
Enter the number of vertices : 4
Enter the number of edges : 5
Enter edges in the format ( src dest weight) :
0   1   10
0   2   6
0   3   5
1   3   15
2   3   4
Following are the edges in the constructed MST :
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree : 19.
```

13.2 Prim's Algorithm :-

AIM :- Write a program using the Prim's Algorithm.

PROGRAM :-

```
import java.util.Arrays;
import java.util.Scanner;
Public class Graph {
  Private int V;
  Private int[][] graph;
  Public Graph(int vertices) {
    this.V = vertices;
    this.graph = new int[V][V];
  }
  Private int minKey(int[] Key, boolean[] mstSet) {
    int min = Integer.MAX_VALUE, minIndex = -1;
    for(int v=0; v<V; v++) {
      if(!mstSet[v] && Key[v] < min) {
        min = Key[v];
```

```
        minIndex = v;
      }
    }
    return minIndex;
}

Public void printMST(int[] parent) {
  System.out.println("Edge \tWeight");
  int totalWeight = 0;
  for (int i = 1; i<V; i++) {
    System.out.println(parent[i]+" - "+ i + "\t" +
    graph[i][parent[i]]);
    totalWeight += graph[i][parent[i]];
  }
  System.out.println("Total Weight of MST : " + totalWeight);
}

Public Static void main(String[] args) {
  Scanner Scanner = new Scanner(System.in);
  System.out.println("Enter the number of Vertices :");
  int vertices = Scanner.nextInt();
  Graph g = new Graph(vertices);
```

```
        System.out.println("Enter the adjacency matrix
                            (use 0 for no edge) :");
        for (int i=0; i< vertices; i++) {
          for (int j=0; j< vertices; j++) {
            g.graph[i][j] = Scanner.nextInt();
          }
        }
        g.primMST();
      }
}
```

OUTPUT :-

```
Enter the number of vertices : 4
Enter the adjacency matrix (use 0 for no edge):
0   10   6   5
10   0   0   15
6    0   0   4
5   15   4   0

Edge    Weight
0 - 1     10
3 - 2     4
```

0 - 3 5

Total Weight of MST : 19.

13.3 Total Number Of Spanning Trees In A Graph :-

AIM :- In Complete Graph, the task is equal to counting different labeled trees with n nodes for which have Cayley's formula.

PROGRAM :-

```
import java.util.Scanner;
Public class NumberOfSpanningTrees {
  Static final int MOD = 1000000007;
  Private long determinant(long[][] matrix, int n) {
    if (n == 1) {
      return matrix[0][0];
    }
    if (n == 2) {
      return (matrix[0][0] * matrix[1][1] -
              matrix[0][1] * matrix[1][0]);
    }
```

```
long det = 0;
long[][] temp = new long[n][n];
for (int p=0; p<n; p++) {
 int tempRow = 0;
 for (int i=1; i<n; i++) {
  int tempCol = 0;
  for (int j=0; j<n; j++) {
   if (j != p) {
    temp[tempRow][tempCol] = matrix[i][j];
    tempCol++;
   }
  }
  tempRow++;
 }
 det += matrix[0][p] * determinant(temp, n-1) *
        (p % 2 == 0 ? 1 : -1);
}
return det;
}
Public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter the number of vertices:");
int V = scanner.nextInt();
System.out.println("Enter the adjacency matrix
  (use 0 for no edge & 1 for an edge): ");
int[][] graph = new int[V][V];
for (int i=0; i<V; i++) {
  for (int j=0; j<V; j++) {
    graph[i][j] = scanner.nextInt();
  }
}
NumberOfSpanningTrees obj = new NumberOfSpanningTrees();
System.out.println("Number of Spanning trees: "+
  obj.numofSpanningTree(graph, V));
}
}
```

OUTPUT :-

```
Enter the number of vertices : 4
Enter the adjacency matrix (use 0 for no edge &
  1 for an edge):
```

0 1 1 1

1 0 1 1

1 1 0 1

1 1 1 0

Number Of Spanning Trees : 16.

13.4 Minimum Product Spanning Tree :-

AIM :- Write a Program On Minimum Product Spanning Tree for a weighted, connected, and undirected graph.

PROGRAM :-

```
import java.util.*;
Public class MinimumProductMST {
  int minKey(double[] Key, boolean[] mstSet, int V) {
    double min = Double.MAX_VALUE;
    int minIndex = -1;
    for (int v = 0; v < V; v++) {
      if (!mstSet[v] && Key[v] < min) {
```

```
        min = key[v];
        minIndex = v;
      }
    }
    return minIndex;
  }
  void primMST(int[][] graph, int V) {
    double[] key = new double[V];
    int[] parent = new int[V];
    boolean[] mstSet = new boolean[V];
    Arrays.fill(key, Double.MAX_VALUE);
    Arrays.fill(mstSet, false);
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < V-1; count++) {
      int u = minKey(key, mstSet, V);
      mstSet[u] = true;
      for (int v = 0; v < V; v++) {
        if (graph[u][v] != 0 && !mstSet[v] &&
            Math.log(graph[u][v]) < key[v]) {
```

```
        Parent[v] = u;
        Key[v] = Math.log(graph[u][v]);
      }
    }
  }
  PrintMST(Parent, V, graph);
}
Public Static void main(String[] args) {
  Scanner Scanner = new Scanner(System.in);
  System.out.println("Enter the number of vertices: ");
  int V = Scanner.nextInt();
  System.out.println("Enter the adjacency matrix
   (Positive weights only, 0 for no edge): ");
  int[][] graph = new int[V][V];
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++) {
      graph[i][j] = Scanner.nextInt();
    }
  }
  MinimumProductMST obj = new MinimumProductMST();
  obj.primMST(graph, V);
}
}
```

OUTPUT :-

Enter the number of vertices : 5

Enter the adjacency matrix ( Positive weights only, 0 for no edge) :

0 2 0 6 0

2 0 3 8 5

0 3 0 0 7

6 8 0 0 9

6 5 7 9 0

| Edge | Weight |
|---|---|
| 0-1 | 2 |
| 1-2 | 3 |
| 0-3 | 6 |
| 1-4 | 5 |

Minimum Product Of MST : 180.0

13.5 Reverse Delete Algorithm for Minimum Spanning Tree :-

AIM :- The Main idea is delete edge if its deletion does not lead to disconnection of graph.

PROGRAM :-

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;
class Edge implements Comparable<Edge> {
  int src, dest, weight;
  Edge (int src, int dest, int weight) {
    this.src = src;
    this.dest = dest;
    this.weight = weight;
  }
  @override
  Public int compareTo (Edge other) {
    return other.weight - this.weight;
  }
}
class Graph {
  Private int V;
  Private ArrayList<Edge> edges;
```

```
void addEdge(int u, int v, int w) {
    edges.add(new Edge(u, v, w));
    adjList.get(u).add(v);
    adjList.get(v).add(u);
}

void reverseDeleteMST() {
    Collections.sort(edges);
    ArrayList<Edge> mstEdges = new ArrayList<>(edges);
    for (Edge edge : edges) {
        adjList.get(edge.src).remove((Integer) edge.dest);
        adjList.get(edge.dest).remove((Integer) edge.src);
        if (!connected()) {
            adjList.get(edge.src).add(edge.dest);
            adjList.get(edge.dest).add(edge.src);
        } else {
            mstEdges.remove(edge);
        }
    }
}

Public class ReverseDeleteMST {
    Public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
System.out.print("Enter the number of vertices : ");
int V = Scanner.nextInt();
Graph graph = new Graph(V);
System.out.print("Enter the number of edges: ");
int E = Scanner.nextInt();
System.out.println("Enter the edges in the format : ");
for (int i = 0; i < E; i++) {
    int Src = Scanner.nextInt();
    int dest = Scanner.nextInt();
    int weight = Scanner.nextInt();
    graph.addEdge(Src, dest, weight);
}
graph.reverseDeleteMST();
}
}
```

OUTPUT :-

```
Enter the number of Vertices : 9
Enter the number of edges : 14
Enter the edges in the format (Src dest weight):
0 1 4
0 7 8
```

1 2 8

1 7 11

2 3 7

2 8 2

2 5 4

3 4 9

3 5 14

4 5 10

5 6 2

6 7 1

6 8 6

7 8 7

Edges in the MST :

3 - 4 : 9

1 - 2 : 8

2 - 3 : 7

0 - 1 : 4

2 - 5 : 4

2 - 8 : 2

5 - 6 : 2

6 - 7 : 1 Total Weight of MST : 37.

VIVA VOCE :-

1) Define Kruskal's Algorithm ?

A) Kruskal's Algorithm builds an MST by adding edges in increasing weight order, ensuring no cycles form, until all vertices are connected.

2) Define Prim's Algorithm ?

A) Prim's Algorithm finds an MST by starting from a vertex, adding the smallest edge connecting the tree to a new vertex, and repeating until all vertices are included.

3) Define Spanning Tree ?

A) A Spanning Tree is a subgraph of a connected graph that includes all vertices with the minimum number of edges, forming a tree (no cycles).

4) Explain reverse delete Algorithm ?

A) The Reverse Delete Algorithm removes edges in decreasing weight order, restoring edges if their removal disconnects the graph, until an MST is formed.