**Assignment No. 03**

Name:- Anirudha Shripad Kale.

Roll No. :- 223029

PRN No :- 22210288

Batch: C1

**Problem Statement:** You are given an undirected weighted graph with nodes and edges. The nodes are numbered from and to. Find the total weight of the minimum spanning tree, as well as one specific minimum spanning tree using Prim’s algorithm. Note that there may be multiple different minimum spanning trees. You need to construct any one of them.

**Objectives:**

1. To understand the methods of finding minimum spanning tree.
2. To find minimum spanning tree using prim’s algorithm.
3. To analyse prim’s algorithm and understand it’s implementation.

**Theory:**

Prim's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, undirected graph. The minimum spanning tree of a graph is a subset of the edges that form a tree connecting all vertices with the minimum possible total edge weight.

Prim's algorithm can be implemented using various data structures such as priority queues (min-heap), adjacency list, or adjacency matrix.

Priority queues are commonly used to efficiently select the vertex with the minimum key value at each step.

Time Complexity:

The time complexity of Prim's algorithm depends on the implementation and the chosen data structure.

Using a binary heap (priority queue) for efficient key value updates results in a time complexity of O((V + E) log V).

Using Fibonacci heap for priority queue operations can reduce the time complexity to O(V log V + E).

**Algorithm:**

1. Initialization:

Choose an arbitrary vertex as the starting point and add it to the MST.

Initialize the key values of all other vertices to infinity.

1. Greedy Step:

At each iteration, select the vertex u that has the minimum key value and is not yet in the MST.

Add u to the MST and update the key values of its neighbouring vertices.

Update the key values based on the weight of the edge connecting the neighbouring vertex to u, if it is less than the current key value.

1. Termination:

Repeat the greedy step until all vertices are included in the MST.

**Input:**

import java.util.\*;

class Edge implements Comparable<Edge> {

    int src;

    int dest;

    int weight;

    Edge(int src, int dest, int weight) {

        this.src = src;

        this.dest = dest;

        this.weight = weight;

    }

    @Override

    public int compareTo(Edge other) {

        return this.weight - other.weight;

    }

}

class Graph {

    int v;

    List<Edge> graph[];

    Graph(int v) {

        this.v = v;

        graph = new ArrayList[v];

        // initialize the arraylist of each vertex as empty arraylist

        for (int i = 0; i < v; i++) {

            graph[i] = new ArrayList<>();

        }

    }

    public void addEdge(int src, int dest, int weight) {

        graph[src].add(new Edge(src, dest, weight));

        graph[dest].add(new Edge(src, dest, weight));

    }

    public int minSpanningTree() {

        boolean inMST[] = new boolean[v]; // to check whether it is present in mst or not

        int parent[] = new int[v]; // to store the parent vertex of respective vertex

        int key[] = new int[v]; // to store the minimum weight to that vertex

        Arrays.fill(key, Integer.MAX\_VALUE); // initially all vertices are at infinity

        PriorityQueue<Edge> pq = new PriorityQueue<>(); // sort the edges according to weight

        key[0] = 0;

        pq.add(new Edge(0, 0, 0));

        while (!pq.isEmpty()) {

            int u = pq.poll().dest;

            inMST[u] = true;

            for (int i = 0; i < graph[u].size(); i++) {

                Edge e = graph[u].get(i);

                int v = e.dest;

                int weight = e.weight;

                if (!inMST[v] && weight < key[v]) { // new weight is less than previous weight then replace it

                    key[v] = weight;

                    pq.add(new Edge(u, v, key[v]));

                    parent[v] = u;

                }

            }

        }

        for (int i = 1; i < v; i++) {

            System.out.println(parent[i] + " - " + i + " : " + key[i]);

        }

        // minimum weight is sum of key array

        int totalWeight = 0;

        for (int i = 1; i < v; i++) {

            totalWeight += key[i];

        }

        return totalWeight;

    }

}

public class Assignment3 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("------------------------------------------------------------");

        System.out.println("\t\*\*\* Finding MST using Prim's Algorithm \*\*\*\t");

        System.out.println("------------------------------------------------------------");

        System.out.print("Enter the no. of vertices : ");

        int v = sc.nextInt();

        Graph G1 = new Graph(v);

        System.out.print("Enter the no. of edges : ");

        int e = sc.nextInt();

        System.out.println("Enter the details of each edge (source, destination, weight):");

        for (int i = 0; i < e; i++) {

            int src = sc.nextInt();

            int dest = sc.nextInt();

            int weight = sc.nextInt();

            G1.addEdge(src, dest, weight);

        }

        System.out.println("The one of the spanning tree is : ");

        int totalWeight = G1.minSpanningTree();

        System.out.println("Total weight of the minimum spanning tree is : " + totalWeight);

    }

}

**Output:**

------------------------------------------------------------

\*\*\* Finding MST using Prim's Algorithm \*\*\*

------------------------------------------------------------

Enter the no. of vertices : 5

Enter the no. of edges : 6

Enter the details of each edge (source, destination, weight):

0 1 2

1 2 3

0 3 6

1 4 5

2 4 7

1 3 8

The one of the spanning tree is :

0 - 1 : 2

1 - 2 : 3

0 - 3 : 6

1 - 4 : 5

Total weight of the minimum spanning tree is : 16