Nama : Andyan Yogawardhana

NIM : 21/482180/PA/21030

Kelas : KOMB1

Tugas 8 – Prim’s Algorithm

Source Code

```
import java.util.Stack;

public class Prim {
    public static void main(String[] args) {
        // create graph (graph 8.2.2)
        MST t = new MST();
        int graph[][] = new int[][]{{0,5,0,5,4,0,0,0,0,0},
                                    {5,0,3,0,7,0,6,8,0,0},
                                    {0,3,0,0,0,0,0,0,5,0},
                                    {5,0,0,0,0,4,0,0,0,0},
                                    {4,7,0,0,0,5,0,0,0,0},
                                    {0,0,0,4,5,0,3,0,0,0},
                                    {0,6,0,0,0,3,0,6,0,6},
                                    {0,8,0,0,0,0,6,0,7,0},
                                    {0,0,5,0,0,0,0,7,0,4},
                                    {0,0,0,0,0,0,6,0,4,0}};

        // print the solution
        t.primMST(graph);
        t.spanTree(graph);
    }
}

class MST {
    // number of vertices in the graph
    private static final int V = 10;
    // total weight on a mst
    private int totalWeight = 0;
    // array to store constructed MST
    int[] parent = new int[V];
    // two dimensional array to store adjacency value
    private boolean[][] isConnected = new boolean[V][V];

    // MST class constructor
    public MST() {
        for(int i = 0; i < V; i++) {
            for(int j = 0; j < V; j++) {
```

```java
                // set the adjacency of i-th and j-th vertices to false
                isConnected[i][j] = false;
            }
        }
    }

    // a utility function to find the vertex with minimum key value, from the set of vertices not yet included in MST
    public int minKey(int[] key, boolean[] mstSet) {
        // initialize min value
        int min = Integer.MAX_VALUE, minIndex = -1;

        for(int v = 0; v < V; v++) {
            if(mstSet[v] == false && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }

    // a utility function to print the constructed MST stored in parent[]
    private void printMST(int[] parent, int[][] graph) {
        System.out.println("---- Graph 8.2.2 ----\n");
        System.out.println("Edge \tWeight");
        for(int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + "\t  " + graph[i][parent[i]]);
        }
        System.out.println("\nTotal MST Cost = " + totalWeight);
    }

    // a function to construct and print MST for a graph represented using adjacency matrix representation
    public void primMST(int[][] graph) {
        // key values used to pick minimum weight edge in cut
        int[] key = new int[V];
        // to represent set of vertices included in MST
        boolean[] mstSet = new boolean[V];
        // initialize all keys as infinite
        for(int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        // always include first 1st vertex in mst
```

```java
        key[0] = 0; // make key 0 so that this vertex is picked as first vertex
        parent[0] = -1; // first node is always root of MST

        // the MST will have V vertices
        for(int count = 0; count < V; count++) {
            // pick the minimum key vertex from the set of vertices not yet included in MST
            int u = minKey(key, mstSet);

            // add the picked vertex to the MST set
            mstSet[u] = true;

            // update key value and parent index of the adjacent vertices of the picked vertex
            // consider only those vertices which are not yet included in MST
            for(int v = 0; v < V; v++) {
                // graph[u][v] is nonzero only for adjacent vertices of m
                // mstSet[v] is false for vertices not yet included in MST
                // update the key only if graph[u][v] is smaller than key[v]
                if(graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                        key[v] = graph[u][v];
                    }
                }
            }

            // count the total weight using loop
            for(int i = 1; i < V; i++) {
                totalWeight += graph[i][parent[i]];
            }

            // print the constructed MST
            printMST(parent, graph);
        }

        // a function to count the possible spanning trees and the minimum tree costs among all of them
        public void spanTree(int[][] graph) {
            // to store number of possible trees
            int trees = 0;
            // to store cost of minimum tree and set its initial value to infinity
```

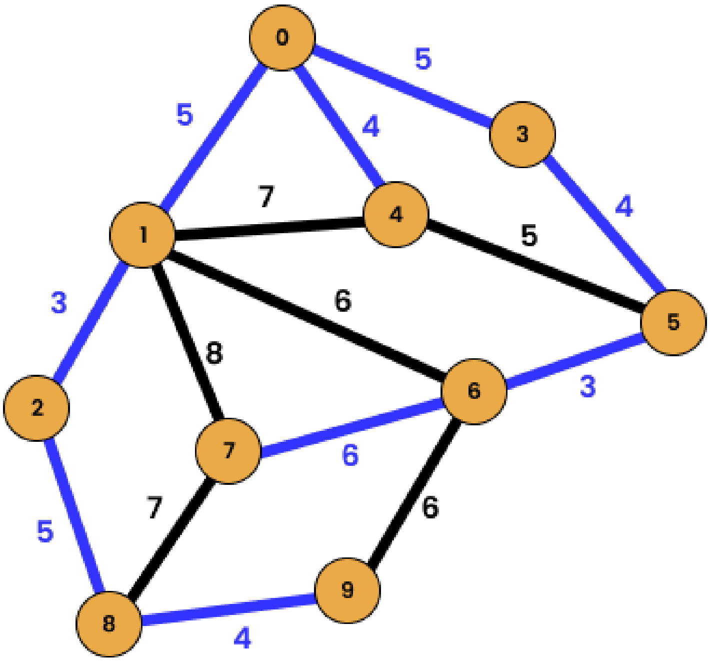
```
            int cost = Integer.MAX_VALUE;
            // to store temporary value of cost that will be used to compare the minimal cost from trees
            int costTemp = 0;

            // connect the i-th vertex to its parent
            for(int i = 1; i < V; i++) {
                isConnected[i][parent[i]] = true;
                isConnected[parent[i]][i] = true;
            }

            for(int j = 1; j < V; j++) {
                // set the j-th vertex and its parent adjacency to false
                isConnected[j][parent[j]] = false;
                isConnected[parent[j]][j] = false;

                for(int k = 0; k < V; k++) {
                    for(int l = k + 1; l < V; l++) {
                        // skip the iteration if the index of k equals to the value of current vertex
                        if(k == parent[j] && k == j) {
                            continue;
                        }

                        // check if k-th and l-th vertices are not connected but it has an edge value
                        if(isConnected[k][l] == false && graph[k][l] != 0) {
                            // set the adjacency of those vertices to true (connected)
                            isConnected[k][l] = true;
                            isConnected[l][k] = true;

                            // check if the graph is a valid tree or not
                            if(isValidTree(isConnected)) {
                                for(int m = 0; m < V; m++) {
                                    for(int n = m + 1; n < V; n++) {
                                        if(isConnected[m][n] == true) {
                                            // add the temporary cost by the value of m-n edge value
                                            costTemp += graph[m][n];
                                        }
                                    }
```

```
                            }

                            // check if the temporary cost value is between the total weight and current cost
                            if(costTemp > totalWeight && costTemp < cost) {
                                // set the cost to current temporary cost
                                cost = costTemp;
                            }

                            // check if the temporary cost value is more than the total weight
                            if(costTemp > totalWeight) {
                                // add the total possible spanning tree
                                trees++;
                            }

                            // reset the temporary cost
                            costTemp = 0;
                        }
                        // reset the vertex adjacency
                        isConnected[k][l] = false;
                        isConnected[l][k] = false;
                    }
                }
            }
            // reset the vertex and its parent adjacency
            isConnected[j][parent[j]] = true;
            isConnected[parent[j]][j] = true;
        }
        // print the result
        System.out.println("Spanning Trees = " + trees);
        System.out.println("Minimum Cost   = " + cost);
    }

    // function to check whether the graph is a valid tree or not (using dfs method)
    private boolean isValidTree(boolean[][] graph) {
        boolean visit[] = new boolean[V];
        Stack<Integer> s = new Stack<Integer>();
        int start = 0;
        // variable to count the total number of visited vertices
        int vertices = 1;

        visit[start] = true;
```

```
                s.push(start);

                while(!s.isEmpty()){
                    start = s.pop();

                    for(int i = 1; i  < V; i++){
                        if(graph[start][i] && !visit[i]){
                                visit[i] = true;
                                s.push(i);
                                // increase the vertices count after a vertex is being visited
                                vertices++;
                        }
                    }
                }
                // if the total counted vertices equals to the graph's initial vertices, then it is a valid tree
                if(vertices == V) {
                    return true;
                }

                // else, it is not a valid tree
                return false;
            }
        }

```

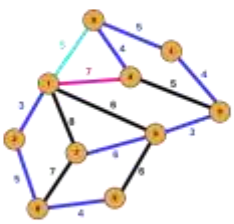
Manual Calculation

1. MST cost

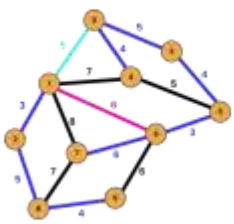


5 + 5 + 4 + 4 + 3 + 5 + 4 + 6 + 3 = 39

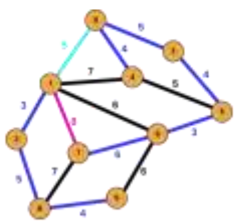
2. Spanning Tree



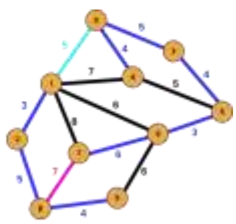
7+5+4+4+3+5+4+6+3=41



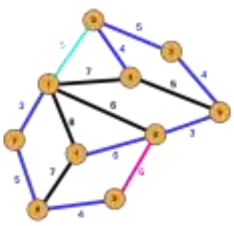
6+5+4+4+3+5+4+6+3=40



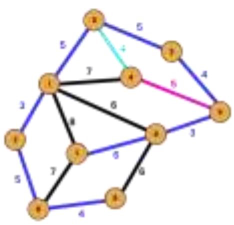
8+5+4+4+3+5+4+6+3=42



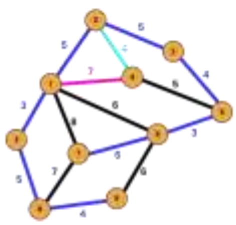
7+5+4+4+3+5+4+6+3=41



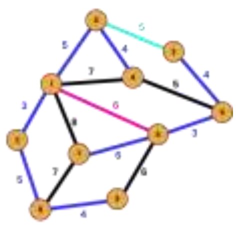
6+5+4+4+3+5+4+6+3=40



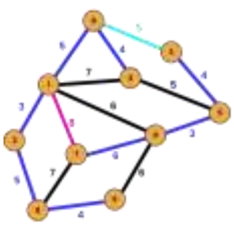
5+5+5+4+3+5+4+6+3=40



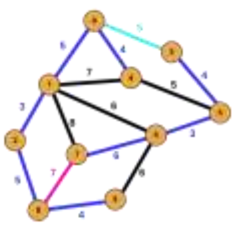
7+5+5+4+3+5+4+6+3=42



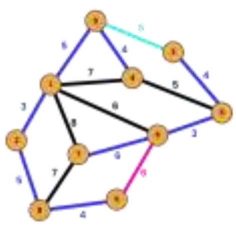
6+4+5+4+3+5+4+6+3=40



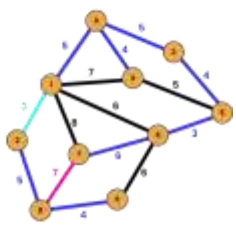
8+4+5+4+3+5+4+6+3=42



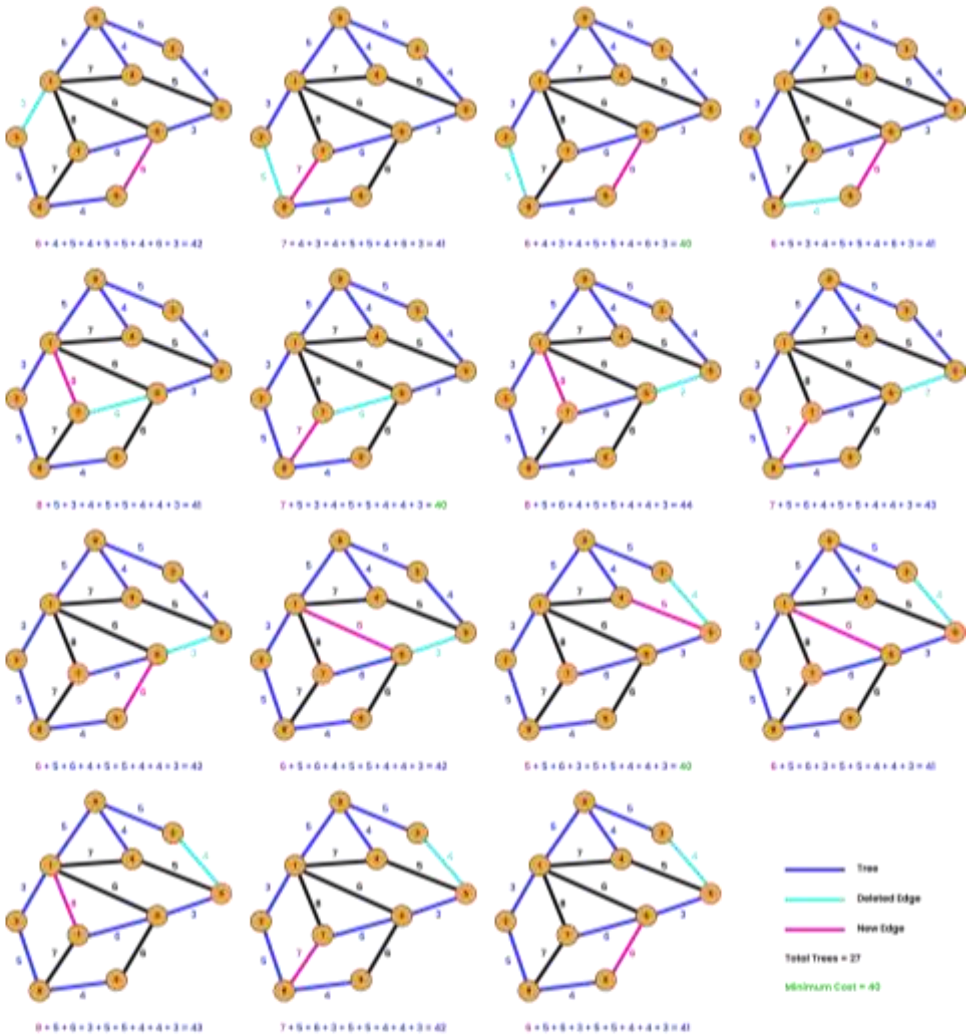
7+4+5+4+3+5+4+6+3=41



6+4+5+4+3+5+4+6+3=40



7+4+5+4+5+5+4+6+3=43



Output Terminal

```
---- Graph 8.2.2 ----

Edge    Weight
0 - 1     5
1 - 2     3
0 - 3     5
0 - 4     4
3 - 5     4
5 - 6     3
6 - 7     6
2 - 8     5
8 - 9     4

Total MST Cost = 39
Spanning Trees = 27
Minimum Cost   = 40
```