Name: Swarnali Ghosh

Java Pre-Skilling Training Session

Assignment -5.3 (Task1 to Task 3)

Module-5 (DAY 9 and 10)

Mail-id: swarnalighosh666@gmail.com

TASK-1

Dijkstra's Shortest Path Finder

Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.

SOLUTION:

```
package com.wipro.graph;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.PriorityQueue;

     public class Dijkstra {
         private HashMap<String, ArrayList<Edge>> adjList = new
HashMap<>();
         private HashMap<String, Integer> distance = new HashMap<>();
         private HashMap<String, String> previous = new HashMap<>();

         public static void main(String[] args) {
             Dijkstra myGraph = new Dijkstra();
             myGraph.addVertex("A");
             myGraph.addVertex("B");
```

```
            myGraph.addVertex("C");
            myGraph.addVertex("D");
            myGraph.addVertex("E");
            myGraph.addVertex("F");

            myGraph.addEdge("A", "B", 2);
            myGraph.addEdge("A", "D", 8);
            myGraph.addEdge("B", "D", 5);
            myGraph.addEdge("B", "E", 6);
            myGraph.addEdge("D", "E", 3);
            myGraph.addEdge("D", "F", 2);
            myGraph.addEdge("E", "F", 1);
            myGraph.addEdge("E", "C", 9);
            myGraph.addEdge("F", "C", 3);
            myGraph.startingpont("A");
            System.out.println("Shortest distance from A to C: " +
myGraph.distance.get("C"));
            System.out.println("Shortest path from A to C: " +
myGraph.getPath("C"));
        }



        private void startingpont(String startVertex) {
            PriorityQueue<String> queue = new PriorityQueue<>((v1,
v2) -> distance.get(v1) - distance.get(v2));
            distance.put(startVertex, 0);
            queue.add(startVertex);

            while (!queue.isEmpty()) {
                String currentVertex = queue.poll();
                for (Edge edge : adjList.get(currentVertex)) {
                    int newDistance = distance.get(currentVertex) +
edge.weight;
                    if (!distance.containsKey(edge.vertex) ||
newDistance < distance.get(edge.vertex)) {
                        distance.put(edge.vertex, newDistance);
                        previous.put(edge.vertex, currentVertex);
                        queue.add(edge.vertex);
                    }
                }
            }
        }
```

```java
    private String getPath(String endVertex) {
        StringBuilder path = new StringBuilder();
        while (endVertex!= null) {
            path.insert(0, endVertex);
            endVertex = previous.get(endVertex);
            if (endVertex!= null) {
                path.insert(0, " -> ");
            }
        }
        return path.toString();
    }

        public boolean addEdge(String vertex1, String vertex2, int 
weight) {
        if (adjList.get(vertex1)!= null && adjList.get(vertex2)!= 
null) {
            adjList.get(vertex1).add(new Edge(vertex2, weight));
            adjList.get(vertex2).add(new Edge(vertex1, weight));
            return true;
        }
        return false;
    }

    class Edge {
        String vertex;
        int weight;

        public Edge(String vertex, int weight) {
            this.vertex = vertex;
            this.weight = weight;
        }
    }

    public boolean addVertex(String vertex) {
        if (adjList.get(vertex) == null) {
            adjList.put(vertex, new ArrayList<Edge>());
            return true;
        }
        return false;
    }

    public void printGraph() {
        System.out.println(adjList);
    }
}
```

Output: -

```
Problems  Javadoc  Declaration  Console
<terminated> Dijkstra [Java Application] C:\Users\DELL\Downloads\eclipse-java-2022-09-R-win32-x86_6
Shortest distance from A to C: 12
Shortest path from A to C: A -> B -> D -> F -> C
```

TASK-2

Kruskal's Algorithm for MST

Implement Kruskal's algorithm to find the minimum spanning tree of a given connected, undirected graph with non-negative edge weights.

SOLUTION:

```java
package com.wipro.graph;

import java.util.Arrays;
import java.util.HashMap;

class Edge implements Comparable<Edge> {
    char src, dest;
    int weight;

    // sorting edges based on their weight
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
}

class Subset {
    int parent, rank;
}
```

```java
public class Kruskal_Algo {
    int V, E; // V = Number of vertices, E = Number of edges
    Edge edge[]; // Collection of all edges
    HashMap<Character, Integer> vertexMap; // Map to convert char
vertices to integer indices

    // Constructor
    Kruskal_Algo(int v, int e) {
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
        vertexMap = new HashMap<>();
    }

    // A utility function to find set of an element i (uses path
compression technique)
    int find(Subset subsets[], int i) {
        if (subsets[i].parent != i)
            subsets[i].parent = find(subsets, subsets[i].parent);
        return subsets[i].parent;
    }

    // A function that does union of two sets of x and y (uses union
by rank)
    void union(Subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    // The main function to construct MST using Kruskal's algorithm
    void kruskalMST() {
        Edge result[] = new Edge[V]; // This will store the resultant
MST
        int e = 0; // An index variable, used for result[]
        int i = 0; // An index variable, used for sorted edges
```

```
        int totalCost = 0;

        for (i = 0; i < V; ++i)
            result[i] = new Edge();

        // Step 1: Sort all the edges in non-decreasing order of their
weight
        Arrays.sort(edge);

        Subset subsets[] = new Subset[V];
        for (i = 0; i < V; ++i)
            subsets[i] = new Subset();

        for (int v = 0; v < V; ++v) {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        i = 0;

        while (e < V - 1) {
            Edge next_edge = edge[i++];

            int x = find(subsets, vertexMap.get(next_edge.src));
            int y = find(subsets, vertexMap.get(next_edge.dest));

            if (x != y) {
                result[e++] = next_edge;
                totalCost += next_edge.weight;
                union(subsets, x, y);
            }
        }

        System.out.println("Following are the edges in the constructed
MST:");
        for (i = 0; i < e; ++i)
            System.out.println(result[i].src + " -- " + result[i].dest
+ " == " + result[i].weight);

        System.out.println("Total cost of the Minimum Spanning Tree: "
+ totalCost);
    }

    public static void main(String[] args) {
        int V = 6; // Number of vertices in the given graph
        int E = 9; // Number of edges in the given graph
```

```
        Kruskal_Algo graph = new Kruskal_Algo(V, E);

        // Create a map to convert vertex labels to indices
        graph.vertexMap.put('a', 0);
        graph.vertexMap.put('b', 1);
        graph.vertexMap.put('c', 2);
        graph.vertexMap.put('d', 3);
        graph.vertexMap.put('e', 4);
        graph.vertexMap.put('f', 5);

        // Define the edges with their respective weights
        graph.edge[0] = new Edge();
        graph.edge[0].src = 'a';
        graph.edge[0].dest = 'b';
        graph.edge[0].weight = 2;

        graph.edge[1] = new Edge();
        graph.edge[1].src = 'd';
        graph.edge[1].dest = 'e';
        graph.edge[1].weight = 2;

        graph.edge[2] = new Edge();
        graph.edge[2].src = 'a';
        graph.edge[2].dest = 'c';
        graph.edge[2].weight = 3;

        graph.edge[3] = new Edge();
        graph.edge[3].src = 'd';
        graph.edge[3].dest = 'f';
        graph.edge[3].weight = 3;

        graph.edge[4] = new Edge();
        graph.edge[4].src = 'b';
        graph.edge[4].dest = 'd';
        graph.edge[4].weight = 3;

        graph.edge[5] = new Edge();
        graph.edge[5].src = 'b';
        graph.edge[5].dest = 'e';
        graph.edge[5].weight = 4;

        graph.edge[6] = new Edge();
        graph.edge[6].src = 'c';
        graph.edge[6].dest = 'e';
        graph.edge[6].weight = 4;
```

```
        graph.edge[7] = new Edge();
        graph.edge[7].src = 'e';
        graph.edge[7].dest = 'f';
        graph.edge[7].weight = 5;

        graph.edge[8] = new Edge();
        graph.edge[8].src = 'b';
        graph.edge[8].dest = 'c';
        graph.edge[8].weight = 5;

        graph.kruskalMST();
    }
}
```

Output: -

```
Problems  Javadoc  Declaration  Console
<terminated> Kruskal_Algo [Java Application] C:\Users\DELL\Downloads\eclipse-java-2022-09-R-win32-
Following are the edges in the constructed MST:
a -- b == 2
d -- e == 2
a -- c == 3
d -- f == 3
b -- d == 3
Total cost of the Minimum Spanning Tree: 13
```

TASK-3

Union-Find for Cycle Detection

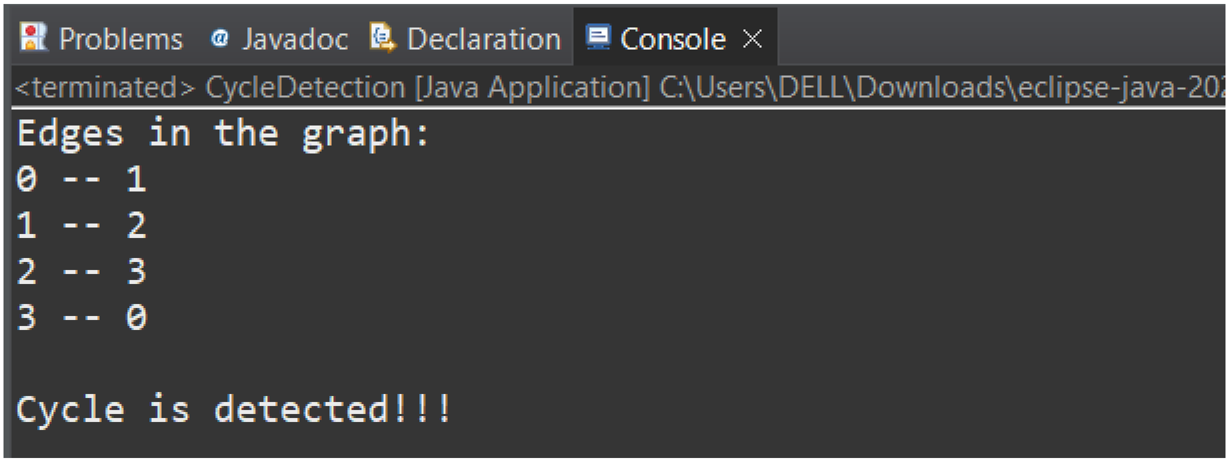
Write a Union-Find data structure with path compression. Use this data structure to detect a cycle in an undirected graph.

SOLUTION:

```java
package com.wipro.swarnali;

import java.util.*;

class UnionFind {
    private int[] parent;
    private int[] rank;

    public UnionFind(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

    public int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]); // path compression
        }
        return parent[x];
    }

    public void union(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX == rootY) {
            return;
        }
        if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }
}

public class CycleDetection {

    public static boolean detectCycle(List<List<Integer>> graph) {
        int n = graph.size();
        UnionFind uf = new UnionFind(n);
```

```java
        for (int u = 0; u < n; u++) {
            for (int v : graph.get(u)) {
                int parentU = uf.find(u);
                int parentV = uf.find(v);
                if (parentU == parentV) {
                    return true; // Cycle detected
                }
                uf.union(parentU, parentV);
            }
        }
        return false; // No cycle detected
    }

    public static void main(String[] args) {
        int n = 4; // Number of vertices
        List<List<Integer>> graph = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            graph.add(new ArrayList<>());
        }

        // Add edges to the graph
        graph.get(0).add(1);
        graph.get(1).add(2);
        graph.get(2).add(3);
        graph.get(3).add(0);

        System.out.println("Edges in the graph:");
    for (int i = 0; i < graph.size(); i++) {
        for (int j : graph.get(i)) {
            System.out.println(i + " -- " + j);
        }
    }

        if (detectCycle(graph)) {
            System.out.println("\nCycle is detected!!!");
        } else {
            System.out.println("\nNo cycle detected!!");
        }
    }
}
```

Output: -


Problems
Javadoc
Declaration
Console
<terminated> CycleDetection [Java Application] C:\Users\DELL\Downloads\eclipse-java-20
Edges in the graph:
0 -- 1
1 -- 2
2 -- 3
3 -- 0
Cycle is detected!!!