Activity 10.1

Ramzy Izza Wardhana - 21/472698/PA/20322

Dijkstra.java

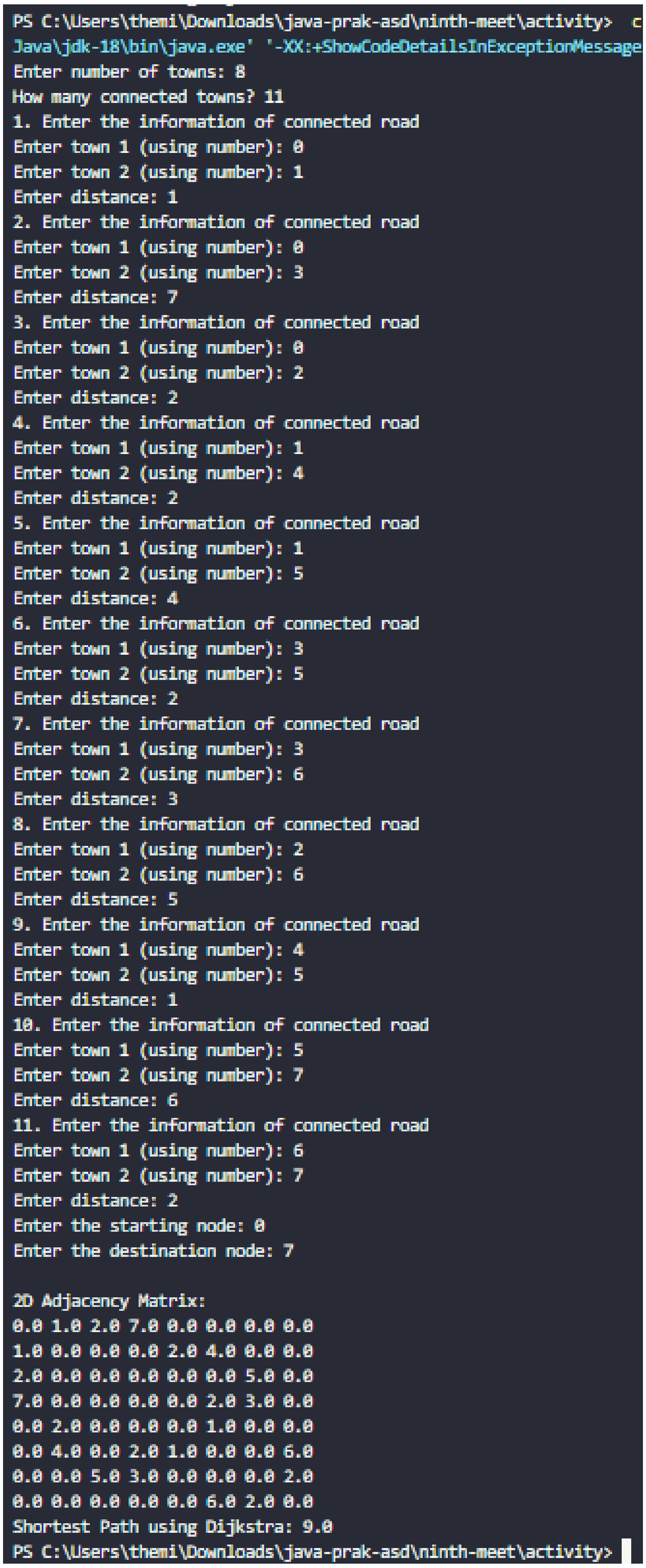
```java
public class Dijkstra {
    int nTown;
    double[][] map;
    double[] distance;
    int src;

    Dijkstra(double[][] map) {
        this.map = map;
        nTown = map.length; // number of towns
    }

    public void solve(int src, int dst) {
        this.src = src; //Set Starting Node
        boolean[] selected = new boolean[nTown]; //To verify whether node
already visisted or not
        distance = new double[nTown]; //To store distance from source to each
node

        for(int i = 0; i < nTown; i++){
            distance[i] = Double.MAX_VALUE; //Set all distance to infinity
            selected[i] = false; //Set all node to unvisited
        }
        distance[src] = 0; //Set distance from source to source to 0

        while (true) {
            int marked = minIndex(distance, selected);

            if (marked < 0) return; //If all node already visited, stop
            if (distance[marked] == Double.MAX_VALUE) return; //If all node is
unreachable, stop
            selected[marked] = true; //Mark node as visited
            if (marked == dst) {
                return; // If destination node is reached, stop
            }


            for (int j = 0; j < nTown; j++){ //For every connected node
                if(map[marked][j] > 0 && !selected[j]){
                    double newDistance = distance[marked] + map[marked][j];
//Calculate new distance
                    if(newDistance < distance[j]) distance[j] = newDistance;
//If new distance is shorter, update distance
                }
            }
        }
    }

    private int minIndex(double[] distance, boolean[] selected){
        double dist  = Double.MAX_VALUE; //Set distance to infinity
        int index = -1; //Set index to -1

        for (int i = 0; i < nTown; i++) {
            if (!selected[i] && distance[i] < dist) { // If node is unvisited
and distance is shorter than current distance
                dist = distance[i]; // Update distance
                index = i; // Update index
            }
        }
        return index;
    }

    public double getDistance(int dst) {
        return (int)distance[dst];
    }
}
```

RunDjikstra.java

```
import java.util.*;

public class RunDijkstra {
    static double[][] map;
    static int src;
    static int dst;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of towns: ");
        int nTown = sc.nextInt(); // number of towns
```

```java
        map = new double[nTown][nTown];

        //I think this would be very tedious to input all distances manually
one by one, especially if nTown is large (n^2)
        // for(int i = 0; i < nTown; i++){
        //     for(int j = 0; j < nTown; j++){
        //         System.out.print("Enter distance from town " + i + " to
town " + j + ": ");
        //         int val = sc.nextInt();
        //         map[i][j] = val;
        //     }
        // }

        //So I made this instead
        System.out.print("How many connected towns? ");
        int nConnect = sc.nextInt();
        for(int i = 0; i < nConnect; i++){
            System.out.println((i+1)+". Enter the information of connected
road " );
            System.out.print("Enter town 1 (using number): ");
            int town1 = sc.nextInt();
            System.out.print("Enter town 2 (using number): ");
            int town2 = sc.nextInt();
            System.out.print("Enter distance: ");
            int dist = sc.nextInt();
            map[town1][town2] = dist;
            map[town2][town1] = dist;
        }


        System.out.print("Enter the starting node: ");
        src = sc.nextInt();
        System.out.print("Enter the destination node: ");
        dst = sc.nextInt();

        //Result of 2D Adjacency Matrix
        System.out.println("\n2D Adjacency Matrix: ");
        for(int i = 0; i < nTown; i++){
            for(int j = 0; j < nTown; j++){
                System.out.print(map[i][j] + " ");
            }
            System.out.println();
        }

        Dijkstra dj = new Dijkstra(map);
        dj.solve(src, dst);
        System.out.println("Shortest Path using Dijkstra: " +
dj.getDistance(dst));
    }
}
```

Output



```
PS C:\Users\themi\Downloads\java-prak-asd\ninth-meet\activity>  c
Java\jdk-18\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessage
Enter number of towns: 8
How many connected towns? 11
1. Enter the information of connected road
Enter town 1 (using number): 0
Enter town 2 (using number): 1
Enter distance: 1
2. Enter the information of connected road
Enter town 1 (using number): 0
Enter town 2 (using number): 3
Enter distance: 7
3. Enter the information of connected road
Enter town 1 (using number): 0
Enter town 2 (using number): 2
Enter distance: 2
4. Enter the information of connected road
Enter town 1 (using number): 1
Enter town 2 (using number): 4
Enter distance: 2
5. Enter the information of connected road
Enter town 1 (using number): 1
Enter town 2 (using number): 5
Enter distance: 4
6. Enter the information of connected road
Enter town 1 (using number): 3
Enter town 2 (using number): 5
Enter distance: 2
7. Enter the information of connected road
Enter town 1 (using number): 3
Enter town 2 (using number): 6
Enter distance: 3
8. Enter the information of connected road
Enter town 1 (using number): 2
Enter town 2 (using number): 6
Enter distance: 5
9. Enter the information of connected road
Enter town 1 (using number): 4
Enter town 2 (using number): 5
Enter distance: 1
10. Enter the information of connected road
Enter town 1 (using number): 5
Enter town 2 (using number): 7
Enter distance: 6
11. Enter the information of connected road
Enter town 1 (using number): 6
Enter town 2 (using number): 7
Enter distance: 2
Enter the starting node: 0
Enter the destination node: 7

2D Adjacency Matrix:
0.0 1.0 2.0 7.0 0.0 0.0 0.0 0.0
1.0 0.0 0.0 0.0 2.0 4.0 0.0 0.0
2.0 0.0 0.0 0.0 0.0 0.0 5.0 0.0
7.0 0.0 0.0 0.0 0.0 2.0 3.0 0.0
0.0 2.0 0.0 0.0 0.0 1.0 0.0 0.0
0.0 4.0 0.0 2.0 1.0 0.0 0.0 6.0
0.0 0.0 5.0 3.0 0.0 0.0 0.0 2.0
0.0 0.0 0.0 0.0 0.0 6.0 2.0 0.0
Shortest Path using Dijkstra: 9.0
PS C:\Users\themi\Downloads\java-prak-asd\ninth-meet\activity>
```