Project Report
MAT1003 - D Slot

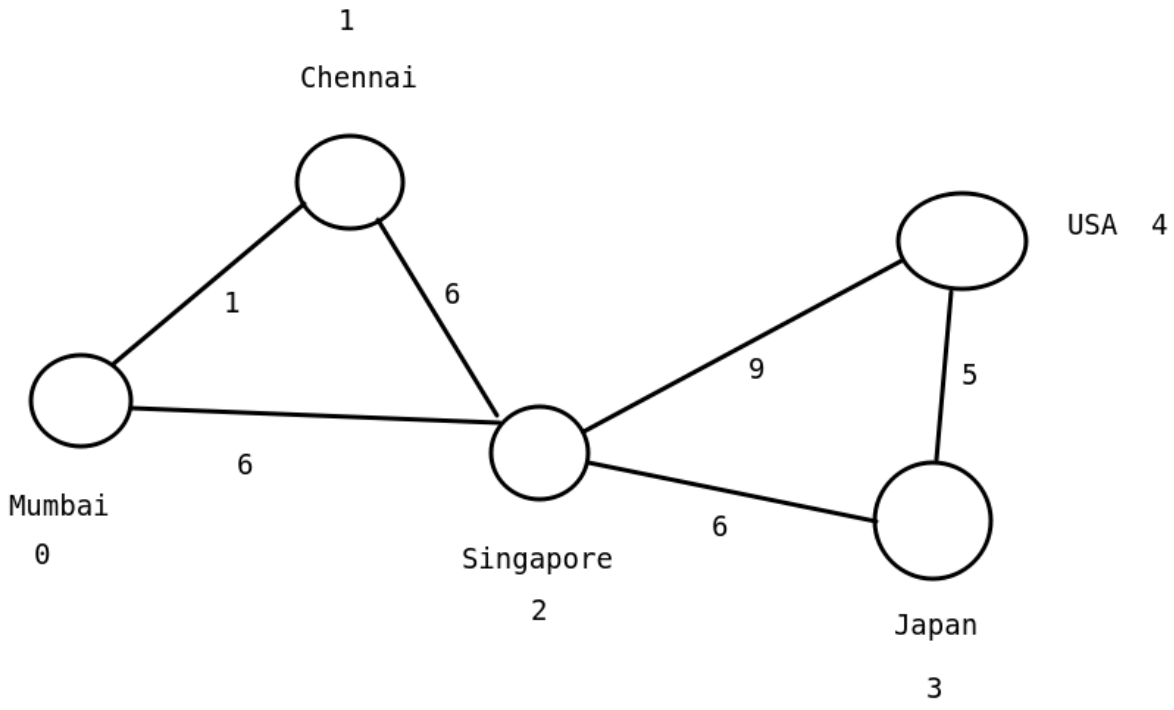
Title: Submarine cable route planner
Group: G7

Group members:

18BCE7147	Vadlapati Hari Praneeth
18BCE7291	Shreekar Reddy V.M.
18BCN7051	Putluri Sai Vishal Reddy
18BCE7335	Penumutchu Avinash
18BCE7339	Vellanki Pramod Sai Kumar

Everything done by: Praneeth

Map visualization:



matrix.txt:

0 1 6 0 0
1 0 6 0 0
6 6 0 6 9
0 0 6 0 5
0 0 9 5 0

Output:

▾ Running code

```
!java RouteFinder.java

        0 - Mumbai
        1 - Chennai
        2 - Singapore
        3 - Japan
        4 - USA

Enter your target: 3
    Vertex              Distance        Path
0 -> 1 Chennai           1              0 - 1 -
0 -> 2 Singapore                 6              0 - 2 -
0 -> 3 Japan             12             0 - 2 - 3 - <- target
0 -> 4 USA               15             0 - 2 - 4 -
```

Code:

```
import java.util.*;
import java.io.File;

class RouteFinder   {
```

```
    static int[][] arr;
    public static void main(String[] args)  {
        boolean status = read_data();
        if (status != true)
            return;
        HashMap<Integer, String> hm = new HashMap<Integer,
String>();
        hm.put(0, "Mumbai");
        hm.put(1, "Chennai");
        hm.put(2, "Singapore");
        hm.put(3, "Japan");
        hm.put(4, "USA");

        // display all nodes from hashmap
        System.out.println("");
        for (int num: hm.keySet())
            System.out.println("\t" + num + " - " + hm.get(num));
        System.out.println("");

        Scanner scan = new Scanner(System.in);
        System.out.print("Enter your target: ");
        int target = scan.nextInt();
        scan.close();

        DijkstrasAlgorithm d = new DijkstrasAlgorithm();
        d.process(target, arr, hm);
        System.out.println("");
    }
    static boolean read_data() {
        try {
            Scanner sc = new Scanner(new File("matrix.txt"));
            int rows = 0;
            while (sc.hasNextLine()) {
                rows++;  // count rows
```

```
                sc.nextLine();
            }
            int columns = rows;
            arr = new int[rows][columns];
            sc.close();

            // read in the data
            sc = new Scanner(new File("matrix.txt"));
            for (int row = 0; row < rows; row++)
                for (int col = 0; col < columns; col++)
                    if (sc.hasNextInt())
                        arr[row][col] = sc.nextInt();
            sc.close();

            return true;
        } catch (Exception e) {
            System.out.println("Sorry... matrix file not found");
            return false;
        }
    }
}


class DijkstrasAlgorithm {

static int NO_PARENT = -1;
static HashMap<Integer, String> hm;
static int target_node;

// Function that implements Dijkstra's
// single source shortest path
// algorithm for a graph represented
// using adjacency matrix representation
static void dijkstra(int[][] matrix, int startVertex)  {
```

```
int numNodes = matrix[0].length;

// shortestDists[i] will hold the
// shortest distance from src to i
int[] shortestDists = new int[numNodes];

// added[i] will true if vertex i is
// included / in shortest path tree
// or shortest distance from src to
// i is finalized
boolean[] added = new boolean[numNodes];

// Initialize all distances as INFINITE and added[] as false
for (int nodeIndex = 0;
        nodeIndex < numNodes; nodeIndex++) {
    shortestDists[nodeIndex] = Integer.MAX_VALUE;
    added[nodeIndex] = false;
}

// Distance of source vertex from itself is always 0
shortestDists[startVertex] = 0;

// Parent array to store shortest path tree
int[] parentArr = new int[numNodes];

// The starting vertex does not have a parent
parentArr[startVertex] = NO_PARENT;

// Find shortest path for all vertices
for (int i = 1; i < numNodes; i++) {

    // Pick the minimum distance vertex
    // from the set of unprocessed vertices
    // nearestVertex is always equal
```

```
        // to startNode in first iteration.
        int nearestVertex = -1;
        int shortestDist = Integer.MAX_VALUE;
        for (int nodeIndex = 0; nodeIndex < numNodes;
                    nodeIndex++)  {
            if (!added[nodeIndex]
                    && shortestDists[nodeIndex] <
                    shortestDist) {
                nearestVertex = nodeIndex;
                shortestDist = shortestDists[nodeIndex];
            }
        }

        // Mark the picked vertex as processed
        added[nearestVertex] = true;

        // Update dist value of the adjacent
        // vertices of the picked vertex.
        for (int nodeIndex = 0; nodeIndex < numNodes;
nodeIndex++) {
            int edgeDistance = matrix[nearestVertex][nodeIndex];

            // if current shortest dist < existing shortest dist,
update array
            if (edgeDistance > 0
                    && (shortestDist + edgeDistance <
                    shortestDists[nodeIndex])) {
                parentArr[nodeIndex] = nearestVertex;
                shortestDists[nodeIndex] =
                        shortestDist + edgeDistance;
            }
        }
    }
    printSolution(startVertex, shortestDists, parentArr);
```

```
}

// function to print the constructed
// distances array and shortest paths
static void printSolution(int startVertex,
                          int[] distances,
                          int[] parentArr)  {
   int numNodes = distances.length;
   System.out.print("    Vertex\t\tDistance\tPath");

   for (int nodeIndex = 0;
             nodeIndex < numNodes; nodeIndex++)  {
       if (nodeIndex != startVertex)  {
           System.out.print("\n" + startVertex);
           System.out.print(" -> " + nodeIndex + " ");
           System.out.print(hm.get(nodeIndex) + " \t\t ");
           System.out.print(distances[nodeIndex] + "\t\t");
           printPath(nodeIndex, parentArr);
           if (target_node == nodeIndex)
               System.out.print("<- target");
       }
   }
}

// Function to print shortest path from
// source to curNode using parentArr array
static void printPath(int curNode, int[] parentArr) {
   // Source node has been processed
   if (curNode == NO_PARENT)
       return;
   printPath(parentArr[curNode], parentArr);
   System.out.print(curNode + " - ");
}
```

```
static void process(int target1, int[][] matrix,
HashMap<Integer, String> hm1)    {
   target_node = target1;
   hm = hm1;
   dijkstra(matrix, 0);
}
}
```

Project Synopsis

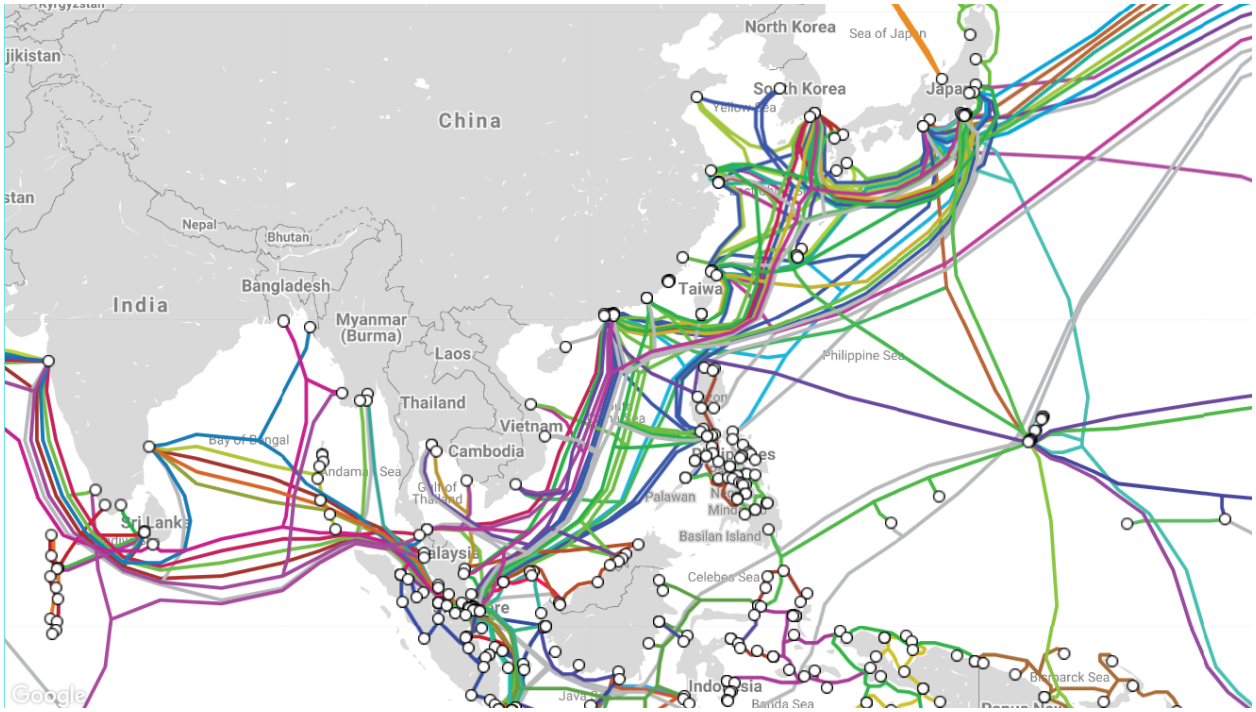
Objectives:

To help the networking companies manage the routes in an efficient way to save the operating costs, and send data in a minimal amount of time.

Methodology:

Map data is entered into text files. We use Dijkstra's algorithm to find the route. The algorithm finds the shortest path between the mentioned source and destination which is entered by the user.
Code is uploaded to GitHub repository:
https://github.com/vh-praneeth/Submarine_cable_route

Map screenshot:



Other parts from Project Proposal document

Abstract:

We need to send data from our computer to a different server using the internet. It is sent to different countries through Submarine cables which are installed in the sea. The cables route about 99% of global internet traffic. These cables are also called Undersea cables.

There are many possible paths using which we can send data. Each cable has a different capacity and different amount of other traffic which is currently being sent. Cable map is available on the website: https://www.submarinecablemap.com/.

Expected Outcome:

Using this project, we write an algorithm that finds the best path to send the data to the destination, with the minimum distance possible.