US17- Procedures

Dinis Araújo (1230767)

Gabriela Teixeira (1230609)

Leonor Marinho (1230977)

Vasco Azevedo (1230776)

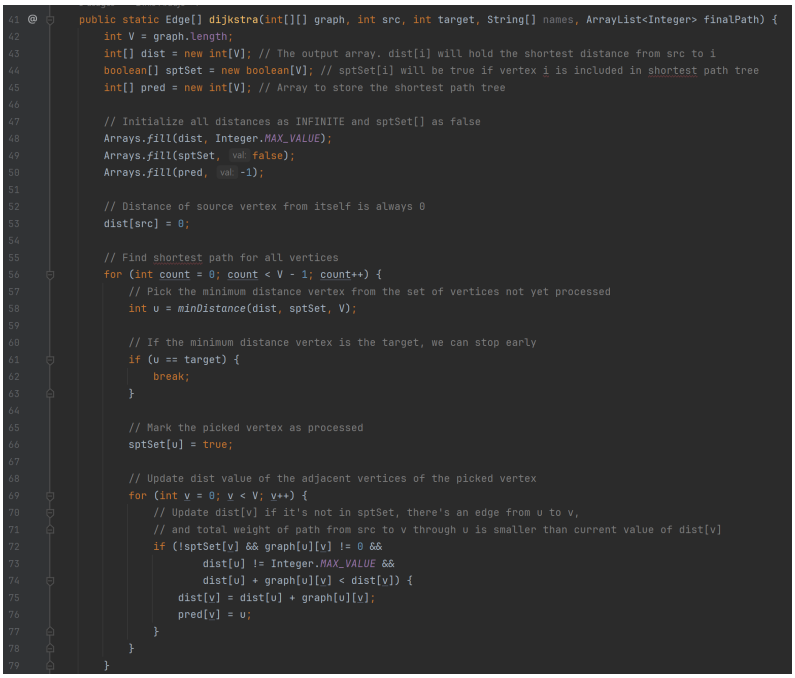
1. We start by reading the csv file containing the adjacency matrix and Turn it into a bidimensional array of ints

```
1 usage  Dinis Araújo +1
public static int[][] readGraphFromCSV(String filePath) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(filePath), charsetName: "UTF-8"));
    String line;
    int[][] graph = null;
    int row = 0;

    while ((line = br.readLine()) != null) {
        // Remove BOM if present
        if (row == 0 && line.startsWith("\uFEFF")) {
            line = line.substring( beginIndex: 1);
        }
        String[] values = line.split( regex: ";");
        if (graph == null) {
            graph = new int[values.length][values.length];
        }
        for (int col = 0; col < values.length; col++) {
            graph[row][col] = Integer.parseInt(values[col].trim());
        }
        row++;
    }

    br.close();
    return graph;
}
```

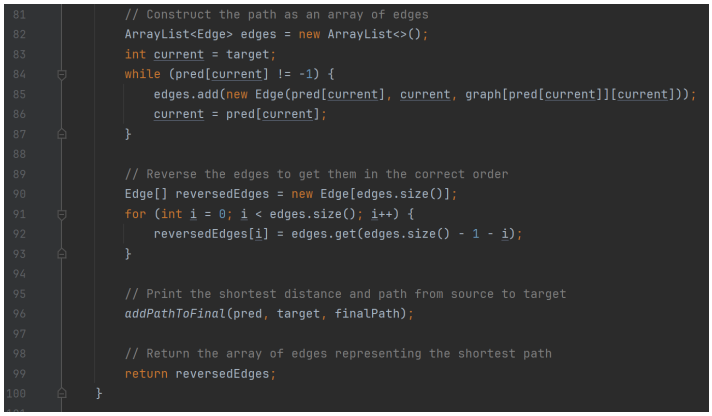
2. We read the names of each vertex from the csv file and store them in an array of strings

```
public static String[] readCSVIntoArray(String csvFile) {

    String line;
    String[] dataArray = null;
    try (       BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(csvFile), charsetName: "UTF-8"));
    ) {
        if ((line = br.readLine()) != null) {
            dataArray = line.split( regex: ";");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return dataArray;
}
```

3. Dijkstras algorithm implementation



```java
public static Edge[] dijkstra(int[][] graph, int src, int target, String[] names, ArrayList<Integer> finalPath) {
    int V = graph.length;
    int[] dist = new int[V]; // The output array. dist[i] will hold the shortest distance from src to i
    boolean[] sptSet = new boolean[V]; // sptSet[i] will be true if vertex i is included in shortest path tree
    int[] pred = new int[V]; // Array to store the shortest path tree

    // Initialize all distances as INFINITE and sptSet[] as false
    Arrays.fill(dist, Integer.MAX_VALUE);
    Arrays.fill(sptSet, val false);
    Arrays.fill(pred, val -1);

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not yet processed
        int u = minDistance(dist, sptSet, V);

        // If the minimum distance vertex is the target, we can stop early
        if (u == target) {
            break;
        }

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex
        for (int v = 0; v < V; v++) {
            // Update dist[v] if it's not in sptSet, there's an edge from u to v,
            // and total weight of path from src to v through u is smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] != 0 &&
                    dist[u] != Integer.MAX_VALUE &&
                    dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                pred[v] = u;
            }
        }
    }
```



```java
    // Construct the path as an array of edges
    ArrayList<Edge> edges = new ArrayList<>();
    int current = target;
    while (pred[current] != -1) {
        edges.add(new Edge(pred[current], current, graph[pred[current]][current]));
        current = pred[current];
    }

    // Reverse the edges to get them in the correct order
    Edge[] reversedEdges = new Edge[edges.size()];
    for (int i = 0; i < edges.size(); i++) {
        reversedEdges[i] = edges.get(edges.size() - 1 - i);
    }

    // Print the shortest distance and path from source to target
    addPathToFinal(pred, target, finalPath);

    // Return the array of edges representing the shortest path
    return reversedEdges;
}
```

4. 186-191 : name to vertex association

```java
public static void main(String[] args) throws IOException {
    int[][] graph = readGraphFromCSV( filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/us17_matrix.csv");
    String[] names = readCSVIntoArray( csvFile: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/us17_points_names.csv");
    System.out.println(names[0]);
    int target = 0;
    for (int i = 0; i < names.length; i++) {
        if (names[i].equals("AP")) {
            target = i;
        }
    }

    Scanner sc = new Scanner(System.in);
    boolean success = false;
    int src = 0;
    do {
        System.out.print("Enter the name of the starting sign: ");
        String name = sc.nextLine();
        for (int i = 0; i < names.length; i++) {
            if (names[i].equals(name)) {
                src = i;
                success = true;
            }
        }
    } while (!success);
```

5. 196-224 : request and storage of user input

```java
        do {
            System.out.print("Enter the name of the starting sign: ");
            String name = sc.nextLine();
            for (int i = 0; i < names.length; i++) {
                if (names[i].equals(name)) {
                    src = i;
                    success = true;
                }
            }
        } while (!success);

        ArrayList<Integer> mustpass = new ArrayList<>();
        String name = "";
        do {
            System.out.print("Insert the signs you want to pass through (type done to finish): ");
            name = sc.nextLine();
            if (!name.equals("done")) {
                boolean added = false;
                for (int i = 0; i < names.length; i++) {
                    if (names[i].equals(name)) {
                        mustpass.add(i);
                        added = true;
                    }
                }
                if (!added) {
                    System.out.println("Invalid sign name. Please try again.");
                }
            }
        } while (!name.equals("done"));
```

6. 227 - 258: dijkstras algorithm for each segment and output of the final result

```
227         int currentSrc = src;
228         int totalCost = 0;
229         ArrayList<Edge> finalEdges = new ArrayList<>();
230         ArrayList<Integer> finalPath = new ArrayList<>();
231         Edge[] edges = null;
232         for (int sign : mustpass) {
233             edges = dijkstra(graph, currentSrc, sign, names, finalPath);
234             for (Edge edge : edges) {
235                 totalCost += edge.weight;
236                 finalEdges.add(edge);
237             }
238             currentSrc = sign;
239         }
240
241         // Finally, find the path from the last must-pass node to the target
242         edges = dijkstra(graph, currentSrc, target, names, finalPath);
243         for (Edge edge : edges) {
244             totalCost += edge.weight;
245             finalEdges.add(edge);
246         }
247
248         // Print the total cost
249         System.out.println("Total cost: " + totalCost);
250
251         // Print the final path
252         System.out.print("Final Path: ");
253         for (int i = 0; i < finalPath.size(); i++) {
254             System.out.print(names[finalPath.get(i)]);
255             if (i < finalPath.size() - 1) {
256                 System.out.print(" -> ");
257             }
258         }
```

7. Write the initial graph to a CSV file

```
167 @   public static void writeGraphToCSV(int[][] graph, String filePath, String[] names) {
168         try (PrintWriter writer = new PrintWriter(new File(filePath))) {
169             for (int i = 0; i < graph.length; i++) {
170                 for (int j = 0; j < graph[i].length; j++) {
171                     if (graph[i][j] != 0) {
172                         writer.println(names[i] + ";" + names[j] + ";" + graph[i][j]);
173                     }
174                 }
175             }
176         } catch (FileNotFoundException e) {
177             System.out.println(e.getMessage());
178         }
179     }
```

8. write the final Path into a csv file

```java
// Write the final path to a CSV file
1 usage  Dinis Araújo
public static void writeFinalPathToCSV(ArrayList<Edge> edges, String filePath, String[] names) {
    try (PrintWriter writer = new PrintWriter(new File(filePath))) {
        for (Edge edge : edges) {
            writer.println(names[edge.from] + ";" + names[edge.to] + ";" + edge.weight);
        }
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    }
}
```

9. plot the input and final graphs

```java
    writeGraphToCSV(graph, filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/US17_initial_graph.csv", names);
    writeFinalPathToCSV(finalEdges, filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/US17_final_path.csv", names);
    BasicConfigurator.configure();
    System.out.println("-----------------------------------------------------------------------------PLOTTING GRAPHS-----------------------------------------------------------------------------");
    MST_PLOTTER.plotMST( filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/US17_final_path.csv", fName: "US17/US17_SHORTESTPATH_OUTPUT");
    MST_PLOTTER.plotMST( filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US17/US17_initial_graph.csv", fName: "US17/US17_INPUT");
    System.out.println("-----------------------------------------------------------------------------GRAPHS PLOTTED SUCCESSFULLY-----------------------------------------------------------------------------");

}
```