# US18 – Procedures

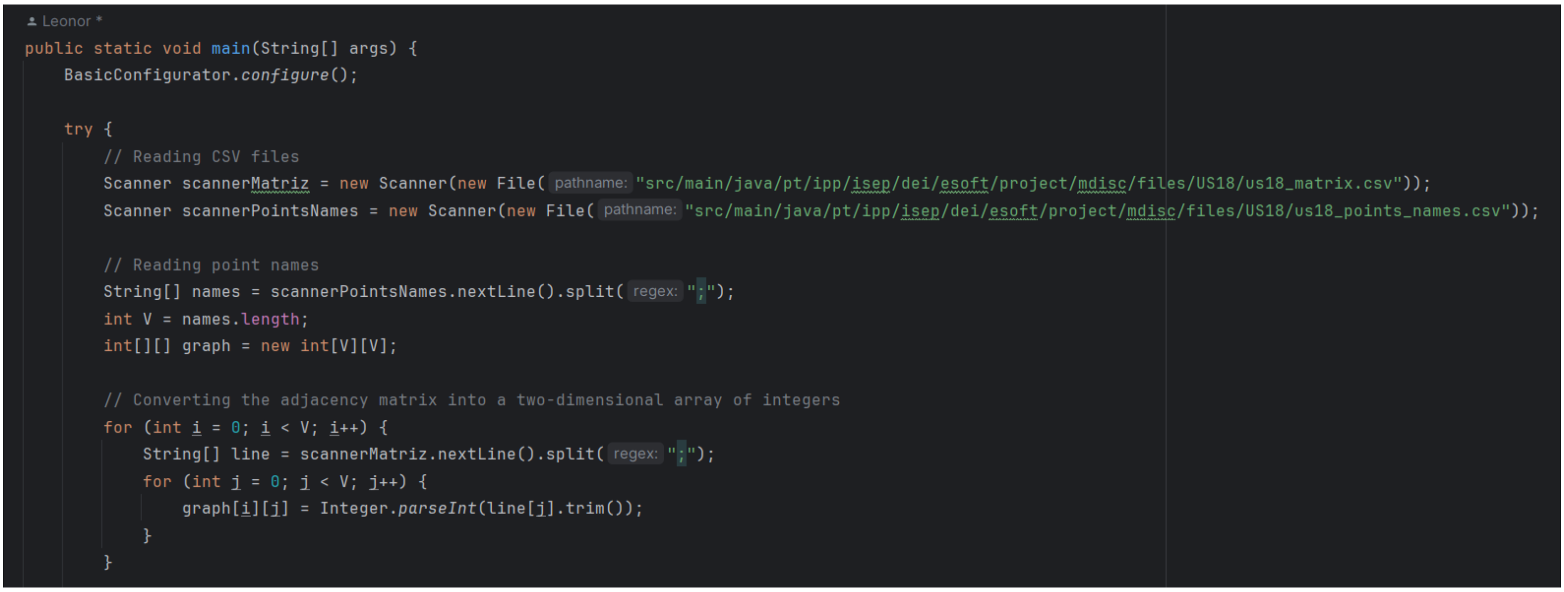
Dinis Araújo (1230767)

Gabriela Teixeira (1230609)

Leonor Marinho (1230977)

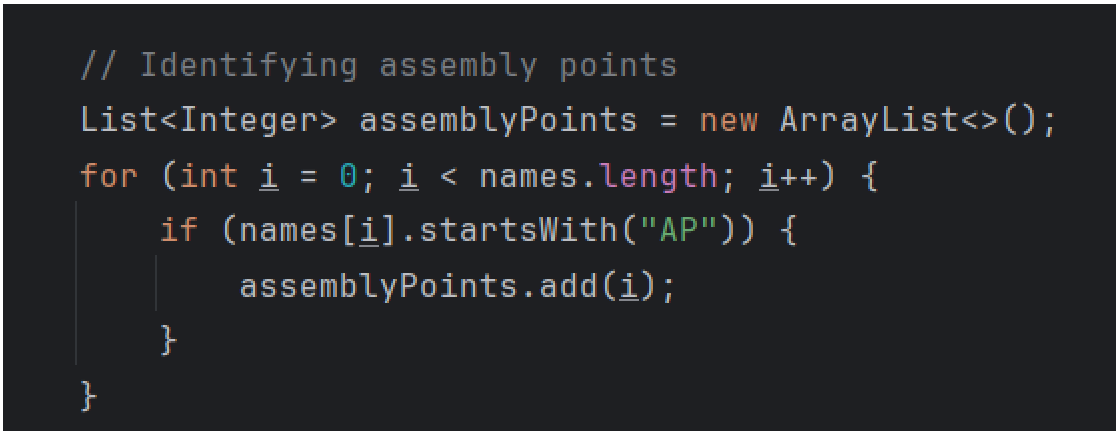
Vasco Azevedo (1230776)

1. We start by reading the csv file containing the adjacency matrix and turn it into a two-dimensional array of integers:



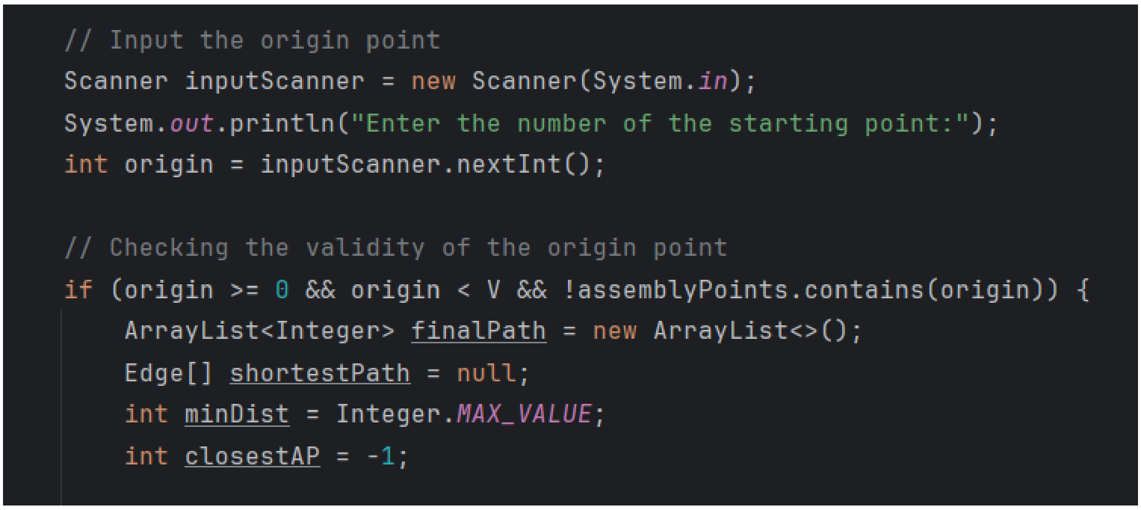
```
Leonor *
public static void main(String[] args) {
    BasicConfigurator.configure();

    try {
        // Reading CSV files
        Scanner scannerMatriz = new Scanner(new File( pathname: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18/us18_matrix.csv"));
        Scanner scannerPointsNames = new Scanner(new File( pathname: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18/us18_points_names.csv"));

        // Reading point names
        String[] names = scannerPointsNames.nextLine().split( regex: ";");
        int V = names.length;
        int[][] graph = new int[V][V];

        // Converting the adjacency matrix into a two-dimensional array of integers
        for (int i = 0; i < V; i++) {
            String[] line = scannerMatriz.nextLine().split( regex: ";");
            for (int j = 0; j < V; j++) {
                graph[i][j] = Integer.parseInt(line[j].trim());
            }
        }
```

2. Them, the system identifies the assembly points, whose names start with "AP":



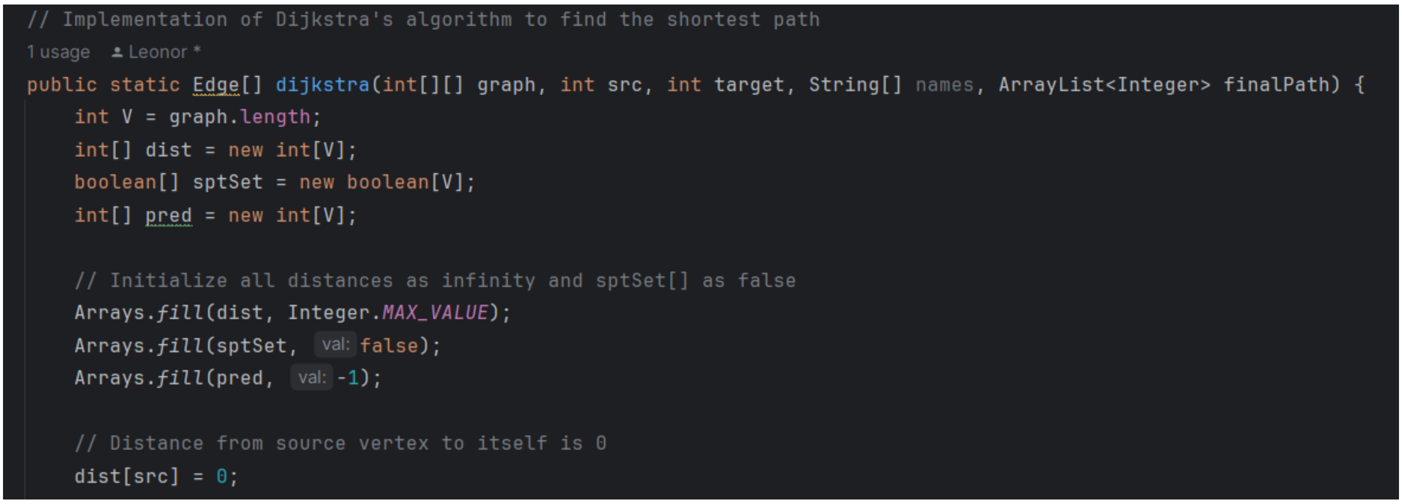
```
// Identifying assembly points
List<Integer> assemblyPoints = new ArrayList<>();
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("AP")) {
        assemblyPoints.add(i);
    }
}
```

3. The user enters the number of the starting point. The system verifies the validity of this poin:

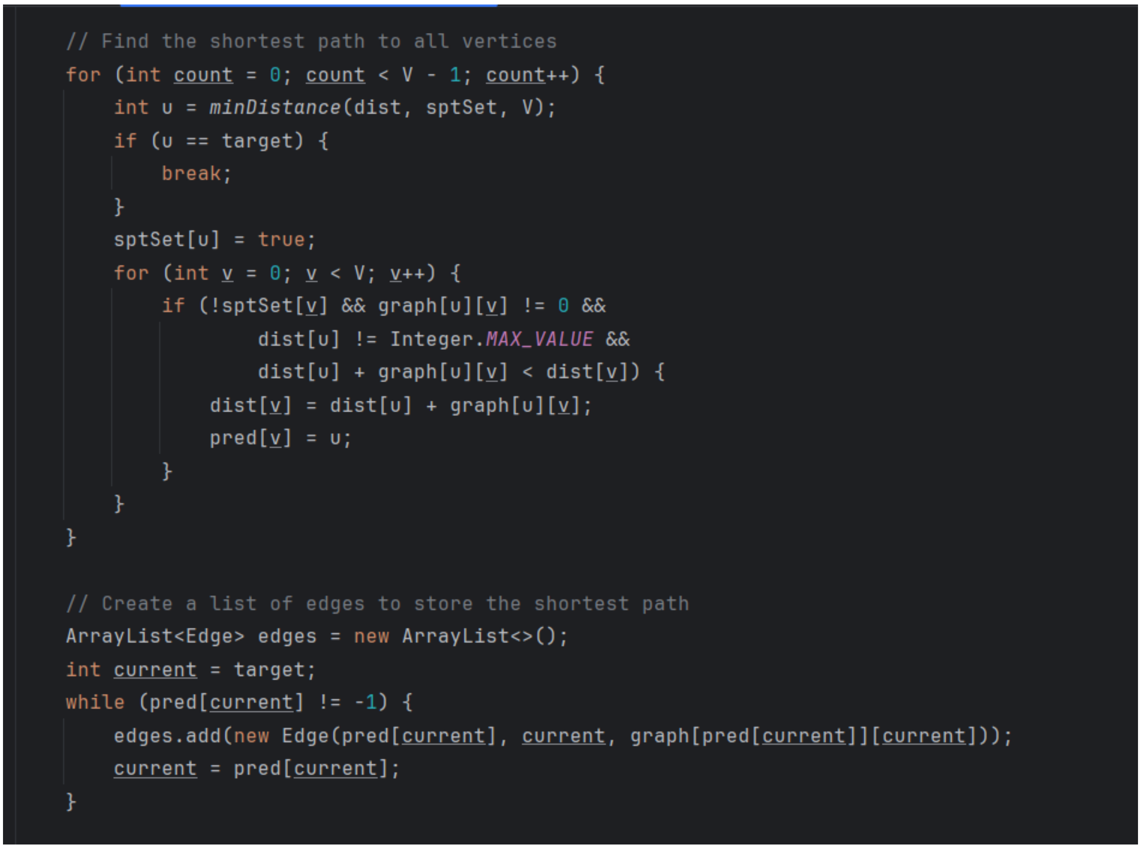


```
// Input the origin point
Scanner inputScanner = new Scanner(System.in);
System.out.println("Enter the number of the starting point:");
int origin = inputScanner.nextInt();

// Checking the validity of the origin point
if (origin >= 0 && origin < V && !assemblyPoints.contains(origin)) {
    ArrayList<Integer> finalPath = new ArrayList<>();
    Edge[] shortestPath = null;
    int minDist = Integer.MAX_VALUE;
    int closestAP = -1;
```

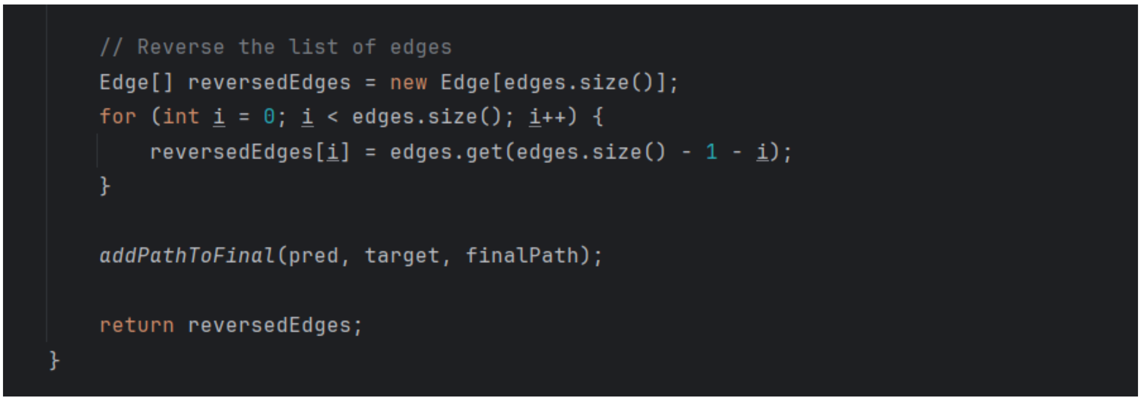
4. The system uses Dijkstra's algorithm to find the shortest path from the starting point to each assembly point, selecting the shortest path among all calculated paths:



```
// Implementation of Dijkstra's algorithm to find the shortest path
1 usage  Leonor *
public static Edge[] dijkstra(int[][] graph, int src, int target, String[] names, ArrayList<Integer> finalPath) {
    int V = graph.length;
    int[] dist = new int[V];
    boolean[] sptSet = new boolean[V];
    int[] pred = new int[V];

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

    // Distance from source vertex to itself is 0
    dist[src] = 0;
```

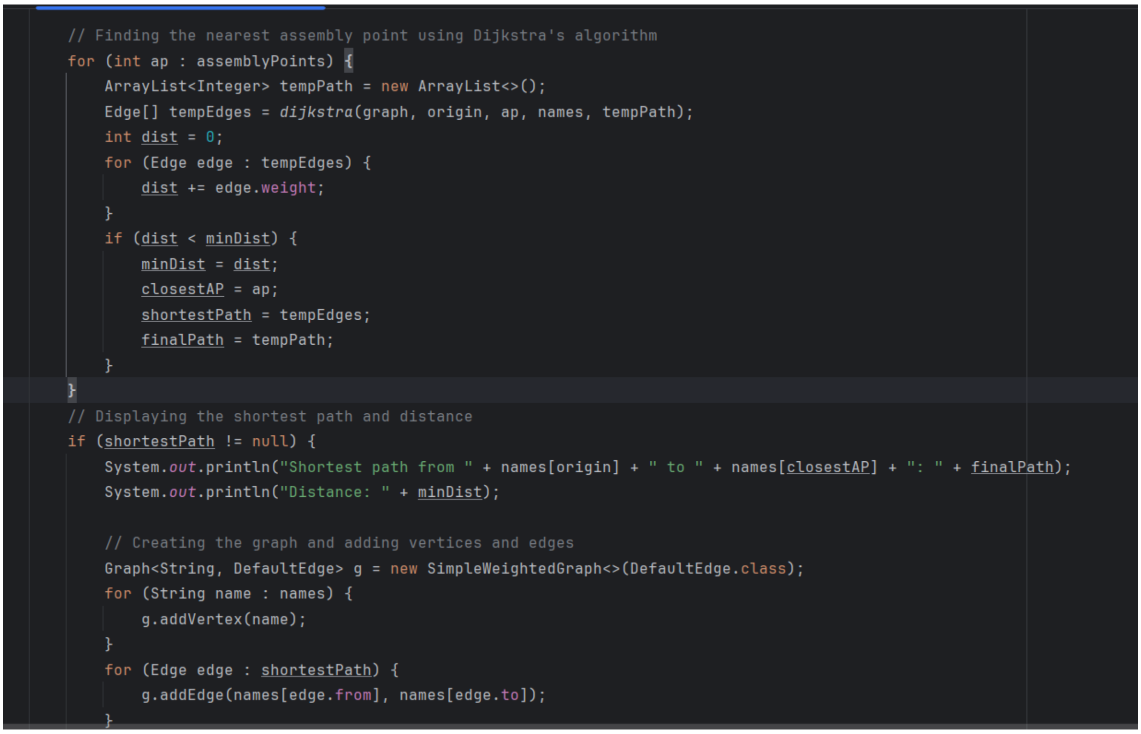


```java
// Find the shortest path to all vertices
for (int count = 0; count < V - 1; count++) {
    int u = minDistance(dist, sptSet, V);
    if (u == target) {
        break;
    }
    sptSet[u] = true;
    for (int v = 0; v < V; v++) {
        if (!sptSet[v] && graph[u][v] != 0 &&
                dist[u] != Integer.MAX_VALUE &&
                dist[u] + graph[u][v] < dist[v]) {
            dist[v] = dist[u] + graph[u][v];
            pred[v] = u;
        }
    }
}

// Create a list of edges to store the shortest path
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];
}
```



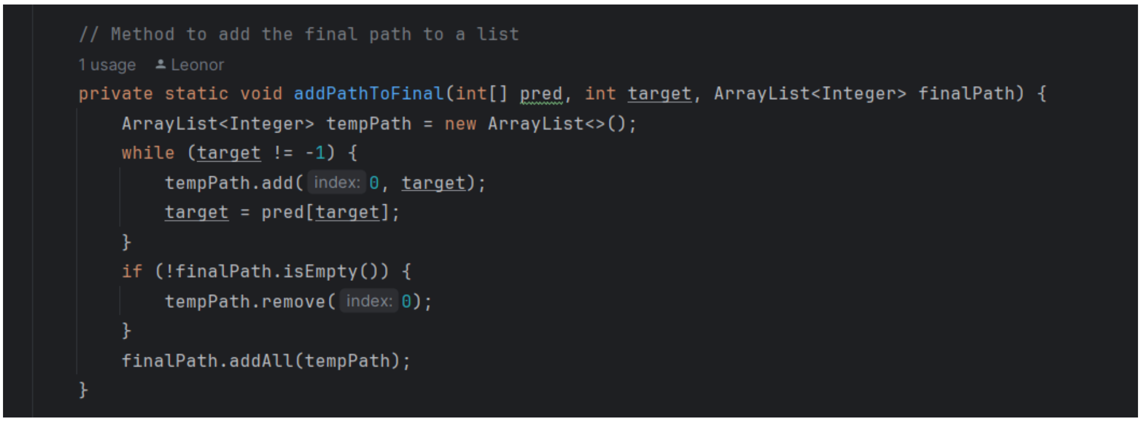
```java
    // Reverse the list of edges
    Edge[] reversedEdges = new Edge[edges.size()];
    for (int i = 0; i < edges.size(); i++) {
        reversedEdges[i] = edges.get(edges.size() - 1 - i);
    }

    addPathToFinal(pred, target, finalPath);

    return reversedEdges;
}
```

5. Finding the nearest assembly point using Dijkstra's algorithm and displaying the shortest path and distance:



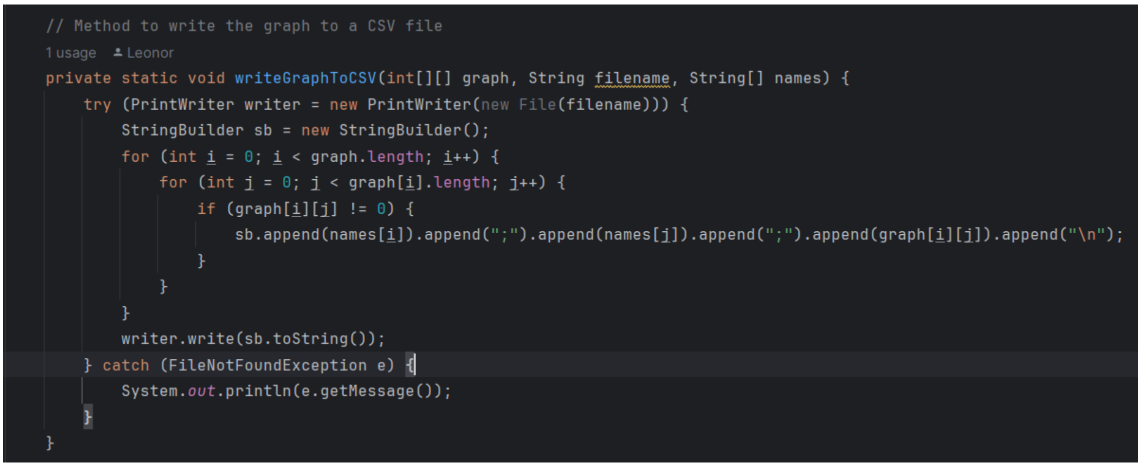
```
// Finding the nearest assembly point using Dijkstra's algorithm
for (int ap : assemblyPoints) {
    ArrayList<Integer> tempPath = new ArrayList<>();
    Edge[] tempEdges = dijkstra(graph, origin, ap, names, tempPath);
    int dist = 0;
    for (Edge edge : tempEdges) {
        dist += edge.weight;
    }
    if (dist < minDist) {
        minDist = dist;
        closestAP = ap;
        shortestPath = tempEdges;
        finalPath = tempPath;
    }
}
// Displaying the shortest path and distance
if (shortestPath != null) {
    System.out.println("Shortest path from " + names[origin] + " to " + names[closestAP] + ": " + finalPath);
    System.out.println("Distance: " + minDist);

    // Creating the graph and adding vertices and edges
    Graph<String, DefaultEdge> g = new SimpleWeightedGraph<>(DefaultEdge.class);
    for (String name : names) {
        g.addVertex(name);
    }
    for (Edge edge : shortestPath) {
        g.addEdge(names[edge.from], names[edge.to]);
    }
```

6. Method to add the final path to a list:



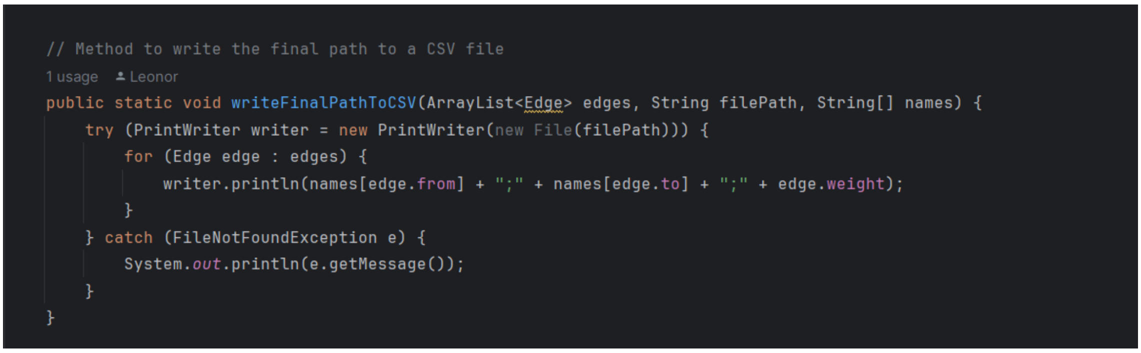
```
// Method to add the final path to a list
1 usage  Leonor
private static void addPathToFinal(int[] pred, int target, ArrayList<Integer> finalPath) {
    ArrayList<Integer> tempPath = new ArrayList<>();
    while (target != -1) {
        tempPath.add( index: 0, target);
        target = pred[target];
    }
    if (!finalPath.isEmpty()) {
        tempPath.remove( index: 0);
    }
    finalPath.addAll(tempPath);
}
```

7. Method to write the graph to a CSV file:



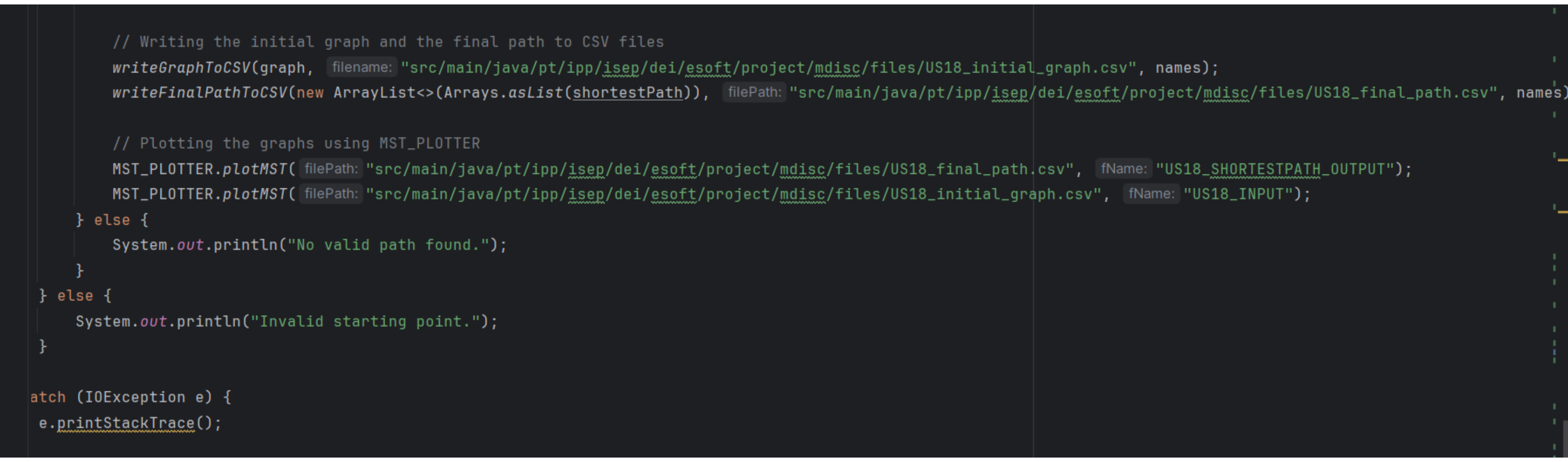
```java
// Method to write the graph to a CSV file
private static void writeGraphToCSV(int[][] graph, String filename, String[] names) {
    try (PrintWriter writer = new PrintWriter(new File(filename))) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < graph.length; i++) {
            for (int j = 0; j < graph[i].length; j++) {
                if (graph[i][j] != 0) {
                    sb.append(names[i]).append(";").append(names[j]).append(";").append(graph[i][j]).append("\n");
                }
            }
        }
        writer.write(sb.toString());
    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    }
}
```

8. Method to write the final path to a CSV file:



```java
// Method to write the final path to a CSV file
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:



```
         // Writing the initial graph and the final path to CSV files
         writeGraphToCSV(graph,  filename: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18_initial_graph.csv", names);
         writeFinalPathToCSV(new ArrayList<>(Arrays.asList(shortestPath)),  filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18_final_path.csv", names)

         // Plotting the graphs using MST_PLOTTER
         MST_PLOTTER.plotMST( filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18_final_path.csv",  fName: "US18_SHORTESTPATH_OUTPUT");
         MST_PLOTTER.plotMST( filePath: "src/main/java/pt/ipp/isep/dei/esoft/project/mdisc/files/US18_initial_graph.csv",  fName: "US18_INPUT");
     } else {
         System.out.println("No valid path found.");
     }
 } else {
     System.out.println("Invalid starting point.");
 }

atch (IOException e) {
 e.printStackTrace();
```