Instituto Superior de Engenharia do Porto

Licenciatura em Engenharia Informática

MDISC – 2023/2024

Report Summary

Analysing the algorithm and results

Authors:

1191330 Luigy Lima

1170499 Daniel Silva

1191377 Tomás Pereira

1200356 Diogo Almeida

Class: 1DB **Group:** 22

Date: 08/06/2024

Lecturer: Alexandra Antunes Gavina

Dijkstra's algorithm(Method):

methodToReplaceSize:

The method methodToReplaceSize counts non-null elements in a list of SignalPoint objects. It initializes a counter and iterates through the list, incrementing the counter for each non-null element. The loop continues until it encounters a null or goes out of the list bounds, which triggers an IndexOutOfBoundsException. When this exception occurs, the method returns the count of non-null elements. If the loop exits normally, it returns -1, although this scenario is unlikely due to the infinite loop structure.

```
3 usages   1191330
private int methodToReplaceSize(List<SignalPoint> vertices) {
    int count = 0;

    try {
        while (true) {
            if (vertices.get(count) != null) {
                count++;
                continue;
            }
            break;
        }
    } catch (IndexOutOfBoundsException io) {
        return count;

    }
    return -1;
}
```

methodToReplaceIndexOf:

The methodToReplaceIndexOf function finds the index of a given SignalPoint object within a list of SignalPoint objects by comparing their names. It first determines the list size, iterates through the list, and returns the index of the matching object if found, otherwise, it returns -1.

```
private int methodToReplaceIndexOf(List<SignalPoint> vertices, SignalPoint signalPoint) {
    int size = methodToReplaceSize(vertices);

    for (int i = 0; i < size; i++) {
        if (vertices.get(i).getName().equals(signalPoint.getName())) {
            return i;
        }
    }
    return -1;
}
```

importNamesFromCSV:

The importNamesFromCSV function reads a CSV file specified by filePath, converts each line into a SignalPoint object, and stores them in a list. It then returns this list of SignalPoint objects.

```
public List<SignalPoint> importNamesFromCSV(String filePath) {

    List<SignalPoint> listaNomes = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String linha;

        while ((linha = br.readLine()) != null) {
            String[] partes = linha.split( regex: ";");
            for (String parte : partes) {
                listaNomes.add(new SignalPoint(parte));
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
    return listaNomes;
}
```

importRouteFromCSV:

The importRouteFromCSV function reads data from a CSV file and creates Route objects based on the information. It uses a list of SignalPoint objects to determine the route connections. Finally, it returns the list of created Route objects.

```
public List<Route> importRouteFromCSV(List<SignalPoint> signalPoints, String filePath) {
    List<Route> routeList = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String line;
        int l = 0;

        while ((line = br.readLine()) != null) {

            String[] parts = line.split( regex: ";");
            int c = 0;
            for (String part : parts) {
                try {
                    int number = Integer.parseInt(part);
                    if (c != l && number != 0) {
                        routeList.add(new Route(number, signalPoints.get(l), signalPoints.get(c)));
                    }
                } catch (NumberFormatException e) {
                }
                ++c;
            }

            ++l;
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
    return routeList;
}
```

findShortestPathToNearestAP:

The findShortestPathToNearestAP function calculates the shortest path from a specified source signal point to the nearest assembly point (AP). It iterates through signal points, updating distances and predecessors until the shortest path to all points is determined. Then, it reconstructs the path to the nearest assembly point and returns it as a list of routes. If no path is found, it returns an empty list.

```java
public List<Route> findShortestPathToNearestAP(SignalPoint source, List<SignalPoint> signalPoints, List<Route> routes, boolean[] isAssemblyPoint) {

    // Número total de pontos de sinalização
    int numPoints = methodToReplaceSize(signalPoints);

    // Arrays para armazenar distâncias, pontos visitados e predecessores
    int[] distances = new int[numPoints];
    boolean[] visited = new boolean[numPoints];
    int[] previous = new int[numPoints];

    // Inicializar arrays de distâncias e visitados
    for (int i = 0; i < numPoints; i++) {
        distances[i] = Integer.MAX_VALUE; // Inicialmente, todas as distâncias são definidas como infinito
        visited[i] = false; // Nenhum ponto foi visitado inicialmente
        previous[i] = -1; // Inicialmente, não há predecessores para nenhum ponto
    }

    // Índice do ponto de origem
    int sourceIndex = methodToReplaceIndexOf(signalPoints, source);
    distances[sourceIndex] = 0; // A distância do ponto de origem para si mesmo é zero

    // Iterar até encontrar o caminho mais curto para todos os pontos ou até que todos os pontos sejam visitados
    for (int i = 0; i < numPoints - 1; i++) {
        // Encontrar o ponto não visitado mais próximo
        int closest = -1; // Índice do ponto mais próximo
        int closestDistance = Integer.MAX_VALUE; // Distância do ponto mais próximo
        for (int j = 0; j < numPoints; j++) {
            // Verificar se o ponto não foi visitado e se sua distância é menor que a distância mais próxima atual
            if (!visited[j] && distances[j] < closestDistance) {
                closest = j;
                closestDistance = distances[j];
            }
        }

        // Se não houver ponto acessível a partir do ponto de origem, sair do loop
        if (closest == -1) {
            break;
        }
```

```java
        // Marcar o ponto escolhido como visitado
        visited[closest] = true;

        // Atualizar as distâncias para os pontos vizinhos do ponto escolhido
        for (Route route : routes) {
            int fromIndex = methodToReplaceIndexOf(signalPoints, route.getS1());
            int toIndex = methodToReplaceIndexOf(signalPoints, route.getS2());
            // Verificar se o ponto atual é o ponto de origem da rota e se o destino não foi visitado ainda
            if (fromIndex == closest && !visited[toIndex]) {
                // Calcular a nova distância
                int newDist = distances[closest] + route.getDistance();
                // Atualizar a distância se a nova distância for menor
                if (newDist < distances[toIndex]) {
                    distances[toIndex] = newDist;
                    previous[toIndex] = closest; // Atualizar o predecessor
                }
            }
        }
    }

    // Reconstruir o caminho mais curto até o ponto de montagem mais próximo (AP)
    List<SignalPoint> path = new ArrayList<>();
    int nearestAPIndex = findNearestAssemblyPointIndex(signalPoints, distances, isAssemblyPoint);
    for (int at = nearestAPIndex; at != -1; at = previous[at]) {
        path.add( index: 0, signalPoints.get(at));
    }

    // Se o ponto de origem não estiver no início, retornar um caminho vazio (nenhum caminho encontrado)
    if (path.isEmpty() || path.get(0) != source) {
        return new ArrayList<>();
    }

    // Construir a rota com base no caminho encontrado
    return constructRoute(path, routes);
}
```

findNearestAssemblyPointIndex:

The findNearestAssemblyPointIndex function finds the index of the nearest assembly point (AP) from a list of signal points based on their distances and whether they are marked as assembly points. It initializes variables to track the nearest AP index and minimum distance. Then, it iterates through the signal points, updating the nearest AP index and minimum distance if an assembly point with a shorter distance is found. Finally, it returns the index of the nearest assembly point.

```
private int findNearestAssemblyPointIndex(List<SignalPoint> signalPoints, int[] distances, boolean[] isAssemblyPoint) {
    int nearestAPIndex = -1;
    int minDistance = Integer.MAX_VALUE;
    int size = methodToReplaceSize(signalPoints);
    for (int i = 0; i < size; i++) {
        if (isAssemblyPoint[i] && distances[i] < minDistance) {
            minDistance = distances[i];
            nearestAPIndex = i;
        }
    }
    return nearestAPIndex;
}
```

constructRoute:

The constructRoute function creates a list of routes based on a list of signal points and existing routes between them. It iterates through adjacent signal points, searching for corresponding routes in the list of routes. If a matching route is found between the current and next signal points, it adds it to the list of new routes. Finally, it returns the list of new routes.

```
private List<Route> constructRoute(List<SignalPoint> signalPoints, List<Route> routes) {
    List<Route> newRoute = new ArrayList<>();
    int size = methodToReplaceSize(signalPoints);

    for (int i = 0; i < size - 1; i++) {
        SignalPoint current = signalPoints.get(i);
        SignalPoint next = signalPoints.get(i + 1);
        for (Route route : routes) {
            if ((route.getS1().equals(current) && route.getS2().equals(next)) || (route.getS1().equals(next) && route.getS2().equals(current))) {
                newRoute.add(new Route(route.getDistance(), current, next));
                break; // Found the corresponding route, so exit the loop
            }
        }
    }

    return newRoute;
}
```

totalDistance:

The totalDistance function calculates the total distance of a list of routes. It iterates through each route in the list and adds its distance to a running total. Finally, it returns the total distance as an integer value.

```
public int totalDistance(List<Route> routes) {
    int total = 0;
    for (Route route : routes) {
        total += route.getDistance();
    }

    return total;
}
```

Input and Output Methods

generateSubgraphCSV:

This method generates CSV content representing a subgraph.
It constructs CSV content by appending vertices, edges, and their costs to a StringBuilder object.

```
public String generateSubgraphCSV(String content, List<Route> shortestPath) {

    StringBuilder csvContent = new StringBuilder();
    StringBuilder contentBuilder = new StringBuilder(content);

    // Anexar arestas
    for (int i = 0; i < shortestPath.size(); i++) {

        if (i == shortestPath.size() - 1) {
            contentBuilder.append(csvContent.append(shortestPath.get(i).getS1().getName()).append(",")
                    .append(shortestPath.get(i).getS2().getName()).append(";"));
            csvContent.delete(0, csvContent.length());
        } else {
            contentBuilder.append(csvContent.append(shortestPath.get(i).getS1().getName()).append(","));
            csvContent.delete(0, csvContent.length());

        }
    }

    // Anexar custo total e calcular o custo total
    content = contentBuilder.append("Total Cost: ").append(totalDistance(shortestPath)).append("\n").toString();

    return content;
}
```

writeCSVToFile:

This method writes CSV content to a file.
It takes the CSV content and writes it to the specified file path.

```
public void writeCSVToFile(String csvContent, String filePath) {
    String csvFilePath = filePath + File.separator + "us18_routes.csv";

    try (BufferedWriter writer = new BufferedWriter(new FileWriter(csvFilePath))) {
        writer.write(csvContent);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

displayPaths:

The displayPaths method finds and displays paths from each signal point to the nearest assembly point (AP). It imports signal point names and routes, determines assembly points, and initializes CSV content. For each signal point, it finds the shortest path to the nearest AP, prints the routes and total distance, and generates CSV content. Finally, it writes the CSV content to a file.

```
public void displayPaths() {

    List<SignalPoint> listPointNames = getController().importNamesFromCSV(FILE_PATH_NAMES);
    List<Route> listRoutes = getController().importRouteFromCSV(listPointNames, FILE_PATH_MATRIX);
    boolean[] isAssemblyPoint = new boolean[listPointNames.size()];
    String content = "";

    for (int i = 0; i < listPointNames.size(); i++) {
        if (listPointNames.get(i).getName().startsWith("AP")) {
            isAssemblyPoint[i] = true;
        }
    }

    for (SignalPoint sp : listPointNames) {

        if (sp.getName().startsWith("AP")) {
            continue;
        }

        List<Route> paths = getController().findShortestPathToNearestAP(sp, listPointNames, listRoutes, isAssemblyPoint);

        for (Route r : paths) {
            System.out.println(r.toString());
        }

        System.out.printf("\nTotal Distance: %d%n", getController().totalDistance(paths));

        content = getController().generateSubgraphCSV(content, paths);
    }

    getController().writeCSVToFile(content, FILE_PATH);
}
```

Results (Display One Graph)

Console Result:

All the routes from a point to the nearest Assembly Point

- Route, this is route of points that leads to an Assembly point

The print below is part of the console and shows the route for the Point 0.

```
--- Shortest Path to Assembly Point ------------------------
Route{distance=3, s1=SignalPoint{name='0'}, s2=SignalPoint{name='7'}}
Route{distance=6, s1=SignalPoint{name='7'}, s2=SignalPoint{name='AP1'}}
```

CSV File Infomation export:

0,7,AP1	Total Cost: 9
1,3,5,AP5	Total Cost: 8
2,3,5,AP5	Total Cost: 10
3,5,AP5	Total Cost: 5
5,AP5	Total Cost: 3
6,5,AP5	Total Cost: 5
7,AP1	Total Cost: 6
8,9,AP4	Total Cost: 4
9,AP4	Total Cost: 2
11,AP4	Total Cost: 2
12,11,AP4	Total Cost: 3
13,12,11,AP4	Total Cost: 4
14,9,AP4	Total Cost: 5
15,AP1	Total Cost: 5
17,AP1	Total Cost: 8
18,AP1	Total Cost: 7
19,AP1	Total Cost: 8
20,22,23,AP2	Total Cost: 7
21,22,23,AP2	Total Cost: 7
22,23,AP2	Total Cost: 3
23,AP2	Total Cost: 1
25,23,AP2	Total Cost: 4
26,28,AP3	Total Cost: 3
27,31,28,AP3	Total Cost: 4
28,AP3	Total Cost: 2
30,AP3	Total Cost: 1
31,28,AP3	Total Cost: 3
32,35,36,AP4	Total Cost: 7
33,35,36,AP4	Total Cost: 7
34,36,AP4	Total Cost: 5
35,36,AP4	Total Cost: 4
36,AP4	Total Cost: 3
37,12,11,AP4	Total Cost: 5
38,40,22,23,AP2	Total Cost: 7
39,38,40,22,23,AP2	Total Cost: 8
40,22,23,AP2	Total Cost: 6