Procedimentos implementados para resolução de US17/18

Classe Graph:

É a classe que recebe e faz a respetivas leituras dos ficheiros .csv e que também contém os métodos que transformam os grafos em imagens.

```
package mdisc.SprintC;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Graph {
    private int[][] adjacencyMatrix;
    private String[] points;

    public Graph(String matrixFile, String pointsFile) throws
IOException {
        readMatrix(matrixFile);
        readPoints(pointsFile);
    }

    private void readMatrix(String file) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line;
        int size = 0;
        while ((line = br.readLine()) != null) {
            size++;
        }
        br.close();

        adjacencyMatrix = new int[size][size];
        br = new BufferedReader(new FileReader(file));
        int row = 0;
        while ((line = br.readLine()) != null) {
            String[] values = line.split(";");
            for (int col = 0; col < values.length; col++) {
                adjacencyMatrix[row][col] =
Integer.parseInt(values[col].replaceAll("[^\\d]", ""));
            }
            row++;
        }
        br.close();
    }

    private void readPoints(String file) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader(file));
        String line = br.readLine();
        if (line != null) {
            points = line.replaceAll("[^\\x20-\\x7E]", "").split(";");
        }
        br.close();
```

```
    }

    public int[][] getAdjacencyMatrix() {
        return adjacencyMatrix;
    }

    public String[] getPoints() {
        return points;
    }

    public void generateDotFile(String filename) throws IOException {
        try (FileWriter writer = new FileWriter(filename)) {
            writer.write("graph G {\n");
            for (int i = 0; i < adjacencyMatrix.length; i++) {
                for (int j = i + 1; j < adjacencyMatrix[i].length;
j++) {
                    if (adjacencyMatrix[i][j] != 0) {
                        writer.write(points[i] + " -- " + points[j] +
" [label=\"" + adjacencyMatrix[i][j] + "\"];\n");
                    }
                }
            }
            writer.write("}\n");
        }
    }

    public void renderDotFile(String dotFileName, String
outputFileName) throws IOException {
        String[] cmd = {"dot", "-Tpng", dotFileName, "-o",
outputFileName};
        ProcessBuilder pb = new ProcessBuilder(cmd);
        pb.redirectErrorStream(true);
        Process process = pb.start();

        try {
            process.waitFor();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public void generateDotFileWithShorterRoute(String filename,
String[] path) throws IOException {
        try (FileWriter writer = new FileWriter(filename)) {
            writer.write("graph G {\n");
            for (int i = 0; i < adjacencyMatrix.length; i++) {
                for (int j = i + 1; j < adjacencyMatrix[i].length;
j++) {
                    if (adjacencyMatrix[i][j] != 0) {
                        boolean isRed = isEdgeInPath(path, points[i],
points[j]);
                        if (isRed) {
                            writer.write(points[i] + " -- " +
points[j] + " [label=\"" + adjacencyMatrix[i][j] + "\",
color=red];\n");
                        } else {
                            writer.write(points[i] + " -- " +
points[j] + " [label=\"" + adjacencyMatrix[i][j] + "\"];\n");
                        }
                    }
                }
```

```
            }
            writer.write("}\n");
        }
    }

    private boolean isEdgeInPath(String[] path, String start, String
end) {
        for (int i = 0; i < path.length - 1; i++) {
            if ((path[i].equals(start) && path[i + 1].equals(end)) ||
(path[i].equals(end) && path[i + 1].equals(start))) {
                return true;
            }
        }
        return false;
    }

}
```

Classe Dijkstra:

É a classe que implementa o algoritmo de Dijkstra, que nos permite conhecer o caminho mais curto de cada ponto.

```
package mdisc.SprintC;

public class Dijkstra {
    private int[] distances;
    private int[] predecessors;
    private boolean[] visited;
    private int numVertices;

    public Dijkstra(int numVertices) {
        this.numVertices = numVertices;
        distances = new int[numVertices];
        predecessors = new int[numVertices];
        visited = new boolean[numVertices];
    }

    public void computeShortestPathsUs17(int[][] graph, int
startVertex) {
        for (int i = 0; i < numVertices; i++) {
            distances[i] = Integer.MAX_VALUE;
            predecessors[i] = -1;
            visited[i] = false;
        }
        distances[startVertex] = 0;
        for (int i = 0; i < numVertices - 1; i++) {
        int u = minDistance();
        visited[u] = true;

        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && graph[u][v] != 0 &&
                    distances[u] != Integer.MAX_VALUE &&
                    distances[u] + graph[u][v] < distances[v]) {
                distances[v] = distances[u] + graph[u][v];
                predecessors[v] = u;
            }
        }
    }
}
```

```
    public void computeShortestPathsUs18(int[][] graph, int[]
startVertices) {
        for (int i = 0; i < numVertices; i++) {
            distances[i] = Integer.MAX_VALUE;
            predecessors[i] = -1;
        }

        for (int startVertex : startVertices) {
            distances[startVertex] = 0;
        }

        for (int i = 0; i < numVertices; i++) {
            int u = minDistance();
            if (u == -1) break;
            visited[u] = true;

            for (int v = 0; v < numVertices; v++) {
                if (!visited[v] && graph[u][v] != 0 &&
                        distances[u] != Integer.MAX_VALUE &&
                        distances[u] + graph[u][v] < distances[v]) {
                    distances[v] = distances[u] + graph[u][v];
                    predecessors[v] = u;
                }
            }
        }
    }

    private int minDistance() {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;

        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && distances[v] <= min) {
                min = distances[v];
                minIndex = v;
            }
        }
        return minIndex;
    }

    public int[] getDistances() {
        return distances;
    }

    public int[] getPredecessors() {
        return predecessors;
    }

    public static String reconstructPathUs17(int[] predecessors, int
currentVertex, int startVertex, String[] points) {
        StringBuilder path = new StringBuilder();
        while (currentVertex != -1) {
            if (path.length() > 0) {
                path.insert(0, ",");
            }
            path.insert(0, points[currentVertex]);
            currentVertex = predecessors[currentVertex];
        }
        return path.toString();
    }
```

```
    public static String reconstructPathUs18(int[] predecessors, int
currentVertex, String[] points) {
        StringBuilder path = new StringBuilder();
        while (currentVertex != -1) {
            if (path.length() > 0) {
                path.insert(0, ",");
            }
            path.insert(0, points[currentVertex]);
            currentVertex = predecessors[currentVertex];
        }
        return path.toString();
    }

    public int getTotalDistance(int vertex) {
        return distances[vertex];
    }
}
```

Classe Main:

Além de correr o programa, esta classe é responsável por interagir com o utilizador e assim saber apartir de qual ponto este pretendo saber o caminho mais curto, também é responsável por fornecer os caminhos aos arquivos e ainda por procurar nos arquivos fornecidos o Ponto de Encontro.

```
package mdisc.SprintC;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        // US17
        Scanner scanner = new Scanner(System.in);
        System.out.println("Indique o nome do ponto de partida (us17):
");
        String startVertexName1 = scanner.nextLine();

        try {
            Graph graph = new Graph("datasets mdisc\\us17_matrix.csv",
"datasets mdisc\\us17_points_names.csv");
            int[][] adjacencyMatrix = graph.getAdjacencyMatrix();
            String[] points = graph.getPoints();

            try {

graph.generateDotFile("src\\main\\java\\mdisc\\SprintC\\grafoInputUs17
.dot");

graph.renderDotFile("src\\main\\java\\mdisc\\SprintC\\grafoInputUs17.d
ot", "src\\main\\java\\mdisc\\SprintC\\grafoInputUs17.png");
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            int startVertex = findAPIndex(graph.getPoints());
            int startVertexIndex =
findStartVertexIndex(graph.getPoints(), startVertexName1);

            Dijkstra dijkstra = new Dijkstra(adjacencyMatrix.length);
            dijkstra.computeShortestPathsUs17(adjacencyMatrix,
startVertex);

            try (FileWriter writer = new
FileWriter("src\\main\\java\\mdisc\\SprintC\\caminhosUS17.csv")) {
                for (int i = 0; i < adjacencyMatrix.length; i++) {
                    if (i != startVertex) {
                        String path =
Dijkstra.reconstructPathUs17(dijkstra.getPredecessors(), i,
startVertex, points);
                        String[] pathElements = path.split(",");
                        StringBuilder reversedPath = new
StringBuilder();
                        for (int j = pathElements.length - 1; j >= 0;
j--) {
                            reversedPath.append(pathElements[j]);
                            if (j != 0) {
                                reversedPath.append(",");
                            }
                        }
                        writer.write(reversedPath.toString() + "; " +
dijkstra.getTotalDistance(i) + "\n");
                    }
                    if (i == startVertexIndex) {
                        String path2 =
Dijkstra.reconstructPathUs17(dijkstra.getPredecessors(), i,
startVertex, points);
                        String[] pathElements2 = path2.split(",");
                        try {

graph.generateDotFileWithShorterRoute("src\\main\\java\\mdisc\\SprintC
\\grafoUs17ShortestRouteTo" + startVertexName1 + ".dot",
pathElements2);

graph.renderDotFile("src\\main\\java\\mdisc\\SprintC\\grafoUs17Shortes
tRouteTo" + startVertexName1 + ".dot",
"src\\main\\java\\mdisc\\SprintC\\grafoUs17ShortestRouteTo" +
startVertexName1 + ".png");
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        // US18
        System.out.println("Indique o nome do ponto de partida (us18):
");
        String startVertexName2 = scanner.nextLine();
        scanner.close();
```

```
        try {
            Graph graph = new Graph("datasets mdisc\\us18_matrix.csv",
"datasets mdisc\\us18_points_names.csv");
            int[][] adjacencyMatrix = graph.getAdjacencyMatrix();
            String[] points = graph.getPoints();

            try {

graph.generateDotFile("src\\main\\java\\mdisc\\SprintC\\grafoInputUs18
.dot");

graph.renderDotFile("src\\main\\java\\mdisc\\SprintC\\grafoInputUs18.d
ot", "src\\main\\java\\mdisc\\SprintC\\grafoInputUs18.png");
            } catch (IOException e) {
                e.printStackTrace();
            }

            List<Integer> startVertices = findAPIndexes(points);
            int startVertexIndex2 =
findStartVertexIndex(graph.getPoints(), startVertexName2);
            int[] startVerticesArray =
startVertices.stream().mapToInt(i -> i).toArray();

            Dijkstra dijkstra = new Dijkstra(adjacencyMatrix.length);
            dijkstra.computeShortestPathsUs18(adjacencyMatrix,
startVerticesArray);

            try (FileWriter writer = new
FileWriter("src\\main\\java\\mdisc\\SprintC\\caminhosUS18.csv")) {
                for (int i = 0; i < adjacencyMatrix.length; i++) {
                    if (!startVertices.contains(i)) {
                        String path =
Dijkstra.reconstructPathUs18(dijkstra.getPredecessors(), i, points);
                        String[] pathElements = path.split(",");
                        StringBuilder reversedPath = new
StringBuilder();
                        for (int j = pathElements.length - 1; j >= 0;
j--) {
                            reversedPath.append(pathElements[j]);
                            if (j != 0) {
                                reversedPath.append(",");
                            }
                        }
                        writer.write(reversedPath + "; " +
dijkstra.getTotalDistance(i) + "\n");
                    }
                    if (i == startVertexIndex2) {
                        String path2 =
Dijkstra.reconstructPathUs18(dijkstra.getPredecessors(), i, points);
                        String[] pathElements2 = path2.split(",");
                        try {

graph.generateDotFileWithShorterRoute("src\\main\\java\\mdisc\\SprintC
\\grafoUs18ShortestRouteTo" + startVertexName2 + ".dot",
pathElements2);

graph.renderDotFile("src\\main\\java\\mdisc\\SprintC\\grafoUs18Shortes
tRouteTo" + startVertexName2 + ".dot",
"src\\main\\java\\mdisc\\SprintC\\grafoUs18ShortestRouteTo" +
startVertexName2 + ".png");
```

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

    private static int findAPIndex(String[] points) {
        for (int i = 0; i < points.length; i++) {
            if (points[i].equals("AP")) {
                return i;
            }
        }
        return -1;
    }

    private static List<Integer> findAPIndexes(String[] points) {
        List<Integer> apIndexes = new ArrayList<>();
        for (int i = 0; i < points.length; i++) {
            if (points[i].startsWith("AP")) {
                apIndexes.add(i);
            }
        }
        return apIndexes;
    }

    private static int findStartVertexIndex (String[] points, String
nameVertex) {
        for (int i = 0; i < points.length; i++) {
            if (points[i].equals(nameVertex)) {
                return i;
            }
        }
        return -1;
    }
}
```