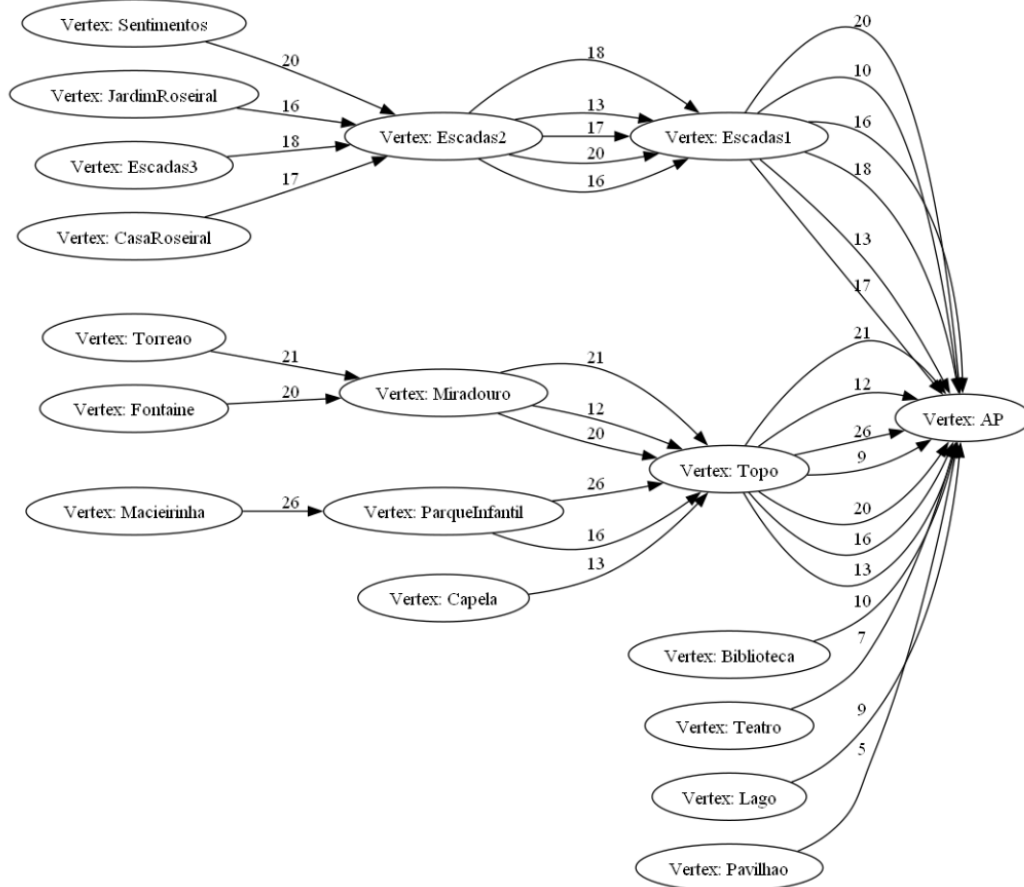


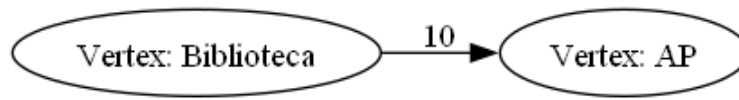
Palácio de Cristal

InputGraph

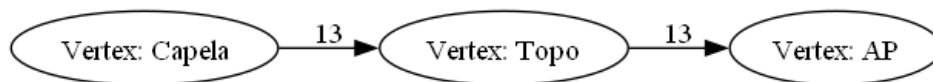


ShortestRoutes

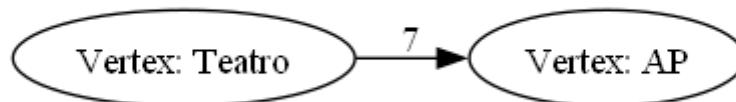
- Biblioteca:



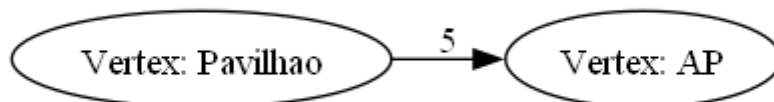
-Capela:



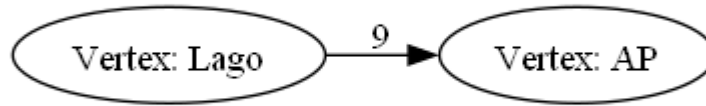
-Teatro:



-Pavilhão:



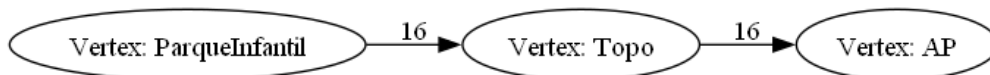
-Lago:



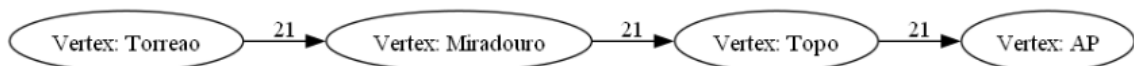
Fontaine:



ParqueInfantil:



Torreao:



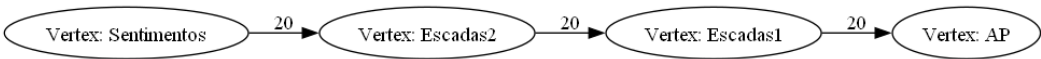
CasaRoseiral:



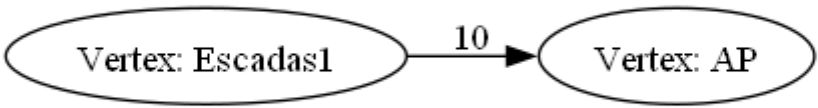
Miradouro:



Sentimentos:



Escadas1:



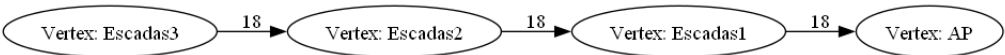
JardimRoseiral:



Macieirinha:



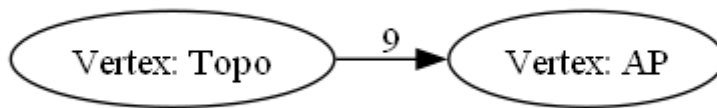
Escadas3:



Escadas2:



Topo:



Implementation

Using the Dijkstra Algorithm to calculate the Evacuation route:

```
private Way calculateShortestWay(List<String> points, int[][] matrix, String start, String end) {
    int size = points.size();
    int startIndex = points.indexOf(start);
    int endIndex = points.indexOf(end);

    int[] distances = new int[size];
    Arrays.fill(distances, Integer.MAX_VALUE);
    distances[startIndex] = 0;

    boolean[] visited = new boolean[size];
    String[] predecessors = new String[size];

    for (int i = 0; i < size; i++) {
        int u = findMinimumDistance(distances, visited);
        visited[u] = true;

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

    List<String> path = new ArrayList<>();
    for (String point = end; point != null; point = predecessors[points.indexOf(point)]) {
        path.add(0, point);
    }

    int totalDistance = distances[endIndex];

    return new Way(path, totalDistance);
}
```

```

public static int findMinimumDistance(int[] distances, boolean[] visited) {
    int minIndex = -1;
    for (int i = 0; i < distances.length; i++) {
        if (!visited[i] && (minIndex == -1 || distances[i] < distances[minIndex])) {
            minIndex = i;
        }
    }
    return minIndex;
}

```

Results in csv:

```

private static void writeShortestWays(String filename, Map<String, Way> shortestWays) throws
IOException {
    try (BufferedWriter writer = Files.newBufferedWriter(Paths.get(filename))) {
        for (Map.Entry<String, Way> entry : shortestWays.entrySet()) {
            String path = entry.getValue().getPoints().stream()
                .map(point -> "Vertex: " + point)
                .collect(Collectors.joining(", "));
            writer.write(String.format("(%s); Cost: %d%n", path, entry.getValue().getDistance()));
        }
    }
}

```

```

private static List<String> readPoints(String filename) throws IOException {
    String content = Files.readString(Paths.get(filename));
    if (content.startsWith("¥uFEFF")) {
        content = content.substring(1);
    }
    return Arrays.asList(content.split(DELIMITER));
}

```

```

private static int[][] readMatrix(String filename, int size) throws IOException {
    int[][] matrix = new int[size][size];
    List<String> lines = Files.readAllLines(Paths.get(filename));

    for (int i = 0; i < size; i++) {
        String[] values = lines.get(i).split(DELIMITER);
        for (int j = 0; j < size; j++) {

```



```

        String value = values[j];
        if (value.startsWith("¥uFEFF")) {
            value = value.substring(1);
        }
        matrix[i][j] = Integer.parseInt(value);
    }
}

return matrix;
}

private String getUserInput() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the name of the point: ");
    return scanner.nextLine();
}

private void writeShortestWay(String filename, String point, Way shortestWay) throws IOException {
    try (BufferedWriter writer = Files.newBufferedWriter(Paths.get(filename))) {
        String path = shortestWay.getPoints().stream()
            .map(p -> "Vertex: " + p)
            .collect(Collectors.joining(", "));
        writer.write(String.format("(%s); Cost: %d%n", path, shortestWay.getDistance()));
    }
}

private static List<String> readData(String filename) throws IOException {
    return Files.readAllLines(Paths.get(filename));
}

```

GraphGenerate:

```

package mdisc.sprintc;

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

```

```

import java.io.IOException;
import java.nio.file.Paths;

public class GraphGenerator {

    public void generateGraph(String csvFile, String graphTitle) throws IOException,
    InterruptedException {
        String dotFile = "src/main/java/mdisc/sprintc/graphsAndPng/" + graphTitle + ".dot";
        String pngFile = "src/main/java/mdisc/sprintc/graphsAndPng/" + graphTitle + ".png";

        try (BufferedReader reader = new BufferedReader(new FileReader(csvFile));
            BufferedWriter writer = new BufferedWriter(new FileWriter(dotFile))) {

            writer.write("digraph " + graphTitle + " {\n");
            writer.write("rankdir=LR;\n");

            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split("; Cost: ");
                String path = parts[0].substring(1, parts[0].length() - 1);
                int cost = Integer.parseInt(parts[1]);

                String[] vertices = path.split(",");
                for (int i = 0; i < vertices.length - 1; i++) {
                    writer.write("{" + vertices[i] + " -> " + vertices[i + 1] + " [label=" + vertices[i] + " cost = " +
                    cost + "];\n");
                }
            }

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

        Process process = new ProcessBuilder("dot", "-Tpng", dotFile, "-o", pngFile).start();
        process.waitFor();
    }
}

```

```
GraphGenerator graphGenerator = new GraphGenerator();  
try {  
    graphGenerator.generateGraph("src/main/java/mdisc/sprintc/output/us17_allpoints.csv",  
    "AllPoints");  
    graphGenerator.generateGraph("src/main/java/mdisc/sprintc/output/us17_output.csv", "Output");  
} catch (InterruptedException e) {  
    e.printStackTrace();  
}
```