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: 12/05/2024

Lecturer: Alexandra Antunes Gavina

Minimum SpanningTree(Methods) algorithm:

sortPipes:

This method sorts an array of pipes based on their distances in ascending order.
It uses a simple selection sort algorithm to achieve this.

```
private void sortPipes(Pipe[] pipes) {
    // Iterate through each element in the array Pipes US13
    for (int i = 0; i < pipes.length - 1; i++) {
        int minIndex = i; // Assume the current index has the minimum distance
        // Iterate through the remaining elements to find the minimum distance
        for (int j = i + 1; j < pipes.length; j++) {
            if (pipes[j].getDistance() < pipes[minIndex].getDistance()) {
                minIndex = j; // Update the index of the minimum distance
            }
        }
        // Swap the positions of the current element and the element with minimum distance
        Pipe temp = pipes[minIndex];//Variavel temporaria
        pipes[minIndex] = pipes[i];
        pipes[i] = temp;
    }
}
```

find:

This method finds the representative of the set containing a given element using the union-find algorithm.
It traverses through the parent array until it finds the root element, which is the representative of the set.

```
private int find(int[] parent, int i) {
    while (parent[i] != i) {
        i = parent[i];
    }
    return i;
}
```

union:

This method combines two sets by their representatives.
It uses the union-find algorithm to merge sets based on their ranks.

```
private void union(int[] parent, int[] rank, int x, int y) {
    int xRoot = find(parent, x); // Find the representative of the first set
    int yRoot = find(parent, y); // Find the representative of the second set

    // Compare the ranks of the sets and merge them accordingly
    if (rank[xRoot] < rank[yRoot]) {
        parent[xRoot] = yRoot;
    } else if (rank[xRoot] > rank[yRoot]) {
        parent[yRoot] = xRoot;
    } else {
        parent[yRoot] = xRoot; // If ranks are equal, merge by making y's root the parent of x's root
        rank[xRoot]++; // Increment the rank of the new root
    }
}
```

findNumVertices:

This method finds the unique vertices from an array of pipes.
It iterates through each pipe and adds the designations of their water points to a list if they are not duplicates.

```
public List<String> findNumVertices(Pipe[] pipes) {
    List<String> vertices = new ArrayList<>();
    for (Pipe pipe : pipes) {
        String designationX = pipe.getWaterPoint_X().getDesignation();
        String designationY = pipe.getWaterPoint_Y().getDesignation();
        if (isDuplicate(vertices, designationX)) {
            vertices.add(designationX);
        }
        if (isDuplicate(vertices, designationY)) {
            vertices.add(designationY);
        }
    }
    return vertices;
}
```

kruskalMinSpanningTree:

This method computes the minimum spanning tree of a graph using Kruskal's algorithm.
It sorts the pipes by distance, finds the unique vertices, and iterates through the sorted pipes to construct the minimum spanning tree.

```java
public List<Pipe> kruskalMinSpanningTree(Pipe[] pipes) {

    sortPipes(pipes); // Sort pipes by distance

    List<String> vertices = findNumVertices(pipes); // Find unique vertices
    int verticeMax = vertices.size(); // Get the total number of vertices
    List<Pipe> minSpanningTree = new ArrayList<>(); // List to store the minimum spanning tree

    int[] parent = new int[verticeMax]; // Array to store the parent of each vertex
    int[] rank = new int[verticeMax]; // Array to store the height of sets

    // Initialize each vertex to be its own parent
    for (int i = 0; i < verticeMax; i++) {
        parent[i] = i;
        rank[i] = 0;
    }

    int edgesAdded = 0; // Counter for added edges
    int pipeIndex = 0; // Pipe index

    // Iterate until all edges are added or all pipes are checked
    while (edgesAdded < verticeMax - 1 && pipeIndex < pipes.length) {
        Pipe currentPipe = pipes[pipeIndex]; // Current pipe
        int x = vertices.indexOf(currentPipe.getWaterPoint_X().getDesignation()); // Index of the start point
        int y = vertices.indexOf(currentPipe.getWaterPoint_Y().getDesignation()); // Index of the end point

        int xRoot = find(parent, x); // Representative of the set for the start point
        int yRoot = find(parent, y); // Representative of the set for the end point

        // Check if adding this edge forms a cycle
        if (xRoot != yRoot) {
            minSpanningTree.add(currentPipe); // Add the pipe to the minimum spanning tree
            edgesAdded++; // Increment the counter for added edges
            union(parent, rank, xRoot, yRoot); // Merge the sets of start and end points
        }
        pipeIndex++; // Next pipe
    }

    return minSpanningTree; // Return the minimum spanning tree
}
```

importRoutesFromCSV:

This method reads routes from a CSV file and creates an array of pipes.
It reads each line of the CSV file, parses the data, and creates a Pipe object for each route.

```java
public Pipe[] importRoutesFromCSV(String filePath) {
    List<Pipe> routesList = new ArrayList<>();

    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String line;
        // Skip header if exists
        br.readLine();

        while ((line = br.readLine()) != null) {
            String[] parts = line.split( regex: ";");
            WaterPoint waterPointX = new WaterPoint(parts[0]);
            WaterPoint waterPointY = new WaterPoint(parts[1]);
            int distance = Integer.parseInt(parts[2]);
            Pipe pipe = new Pipe(waterPointX, waterPointY, distance);
            routesList.add(pipe);
        }

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

    // Convert list to array
    Pipe[] routesArray = new Pipe[routesList.size()];
    routesArray = routesList.toArray(routesArray);
    return routesArray;
}
```

Input and Output
Graphs 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(List<Pipe> minSpanningTree) {
    StringBuilder csvContent = new StringBuilder();
    int totalCost = 0;

    // Append headers
    csvContent.append("Vertice,Vertice,Edge Cost\n");

    // Append edges and calculate total cost
    for (Pipe pipe : minSpanningTree) {
        csvContent.append(pipe.getWaterPoint_X().getDesignation()).append(",")
                .append(pipe.getWaterPoint_Y().getDesignation()).append(",")
                .append(pipe.getDistance()).append("\n");
        totalCost += pipe.getDistance(); // Accumulate the distance for total cost
    }

    // Append total cost
    csvContent.append("\nTotal Cost:,").append(totalCost);

    return csvContent.toString(); // Return the CSV content
}
```

writeCSVToFile:

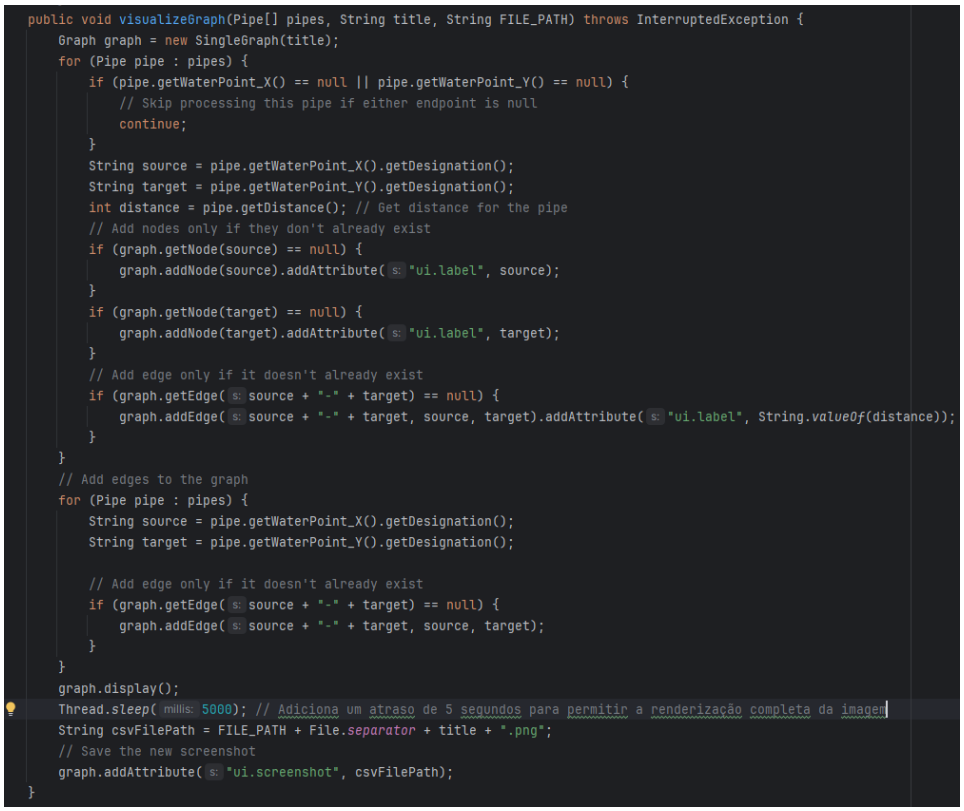
This method writes CSV content to a file.
It takes the CSV content and writes it to the specified file path.

```
1 usage  new *
public void writeCSVToFile(String csvContent, String filePath) {
    String csvFilePath = filePath + File.separator + "output_subgraph.csv";

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

visualizeGraph:

This method visualizes a graph using the GraphStream library.
It creates a graph object, adds nodes and edges to it based on the pipes provided, and displays the graph.
It also saves a screenshot of the graph as a PNG file.



```
public void visualizeGraph(Pipe[] pipes, String title, String FILE_PATH) throws InterruptedException {
    Graph graph = new SingleGraph(title);
    for (Pipe pipe : pipes) {
        if (pipe.getWaterPoint_X() == null || pipe.getWaterPoint_Y() == null) {
            // Skip processing this pipe if either endpoint is null
            continue;
        }
        String source = pipe.getWaterPoint_X().getDesignation();
        String target = pipe.getWaterPoint_Y().getDesignation();
        int distance = pipe.getDistance(); // Get distance for the pipe
        // Add nodes only if they don't already exist
        if (graph.getNode(source) == null) {
            graph.addNode(source).addAttribute( s: "ui.label", source);
        }
        if (graph.getNode(target) == null) {
            graph.addNode(target).addAttribute( s: "ui.label", target);
        }
        // Add edge only if it doesn't already exist
        if (graph.getEdge( s: source + "-" + target) == null) {
            graph.addEdge( s: source + "-" + target, source, target).addAttribute( s: "ui.label", String.valueOf(distance));
        }
    }
    // Add edges to the graph
    for (Pipe pipe : pipes) {
        String source = pipe.getWaterPoint_X().getDesignation();
        String target = pipe.getWaterPoint_Y().getDesignation();

        // Add edge only if it doesn't already exist
        if (graph.getEdge( s: source + "-" + target) == null) {
            graph.addEdge( s: source + "-" + target, source, target);
        }
    }
    graph.display();
    Thread.sleep( millis: 5000); // Adiciona um atraso de 5 segundos para permitir a renderização completa da imagem
    String csvFilePath = FILE_PATH + File.separator + title + ".png";
    // Save the new screenshot
    graph.addAttribute( s: "ui.screenshot", csvFilePath);
}
```

Results (Jardim Especies Nucleo Rural)

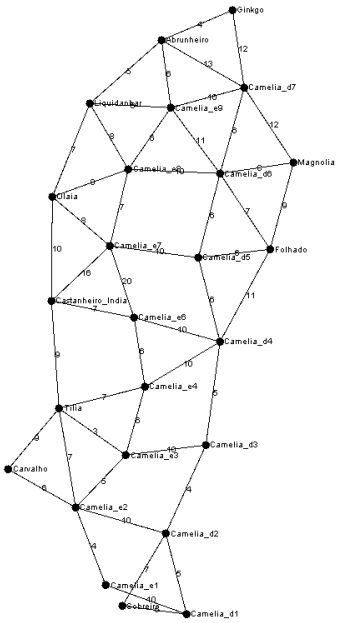
Console Result:

- Graph Dimension = 50 typically refers to the number of edges in the graph.
- Graph Order = 25 usually denotes the number of vertices or nodes in the graph.
- Cost of a Minimum Spanning Tree = 143 suggests the total weight or cost of the minimum spanning tree of the graph.

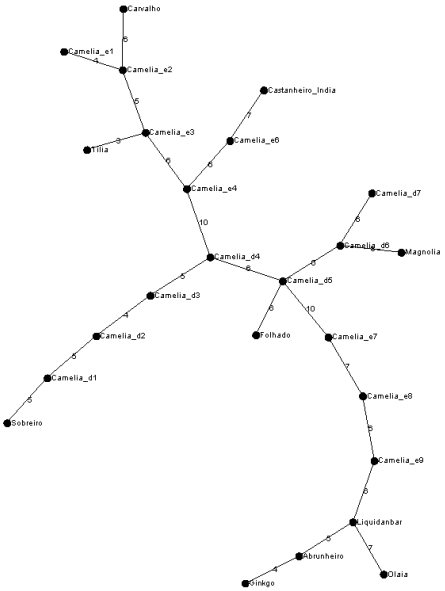
```
--- Minimum Spanning Tree Visualization ------------------------
Enter the path to the CSV file: C:\ISEP\2Semetre\lei-24-s2-g22\MDISC\US13\US13_JardimEspeciesNucleoRural.csv
Cost of a Minimum spanning tree: 143
Graph Order: 25
Graph Dimension: 50
```

Normal graph and minimum spanning three

Input Graph



output Graph



CSV File Infomation export:

```
Vertice,Vertice,Edge Cost
Tilia,Camelia_e3,3
Abrunheiro,Ginkgo,4
Camelia_e1,Camelia_e2,4
Camelia_d3,Camelia_d2,4
Liquidanbar,Abrunheiro,5
Sobreiro,Camelia_d1,5
Camelia_e2,Camelia_e3,5
Camelia_d4,Camelia_d3,5
Camelia_d2,Camelia_d1,5
Liquidanbar,Camelia_e9,6
Folhado,Camelia_d5,6
Camelia_e3,Camelia_e4,6
Camelia_e4,Camelia_e6,6
Camelia_e8,Camelia_e9,6
Camelia_d7,Camelia_d6,6
Camelia_d6,Camelia_d5,6
Camelia_d5,Camelia_d4,6
Carvalho,Camelia_e2,6
Olaia,Liquidanbar,7
Castanheiro_India,Camelia_e6,7
Camelia_e7,Camelia_e8,7
Magnolia,Camelia_d6,8
Camelia_e4,Camelia_d4,10
Camelia_e7,Camelia_d5,10

Total Cost:,143
```

Results US14

CSV File Information Export

```
Input Size,Execution Time (ms)
150,24
300,50
450,22
600,34
750,49
900,70
1050,117
1200,79
1350,101
1500,105
1650,131
1800,149
1950,175
2100,218
2250,265
2400,270
2550,297
2700,376
2850,403
3000,432
3150,429
3300,507
3450,518
3600,636
3750,651
3900,713
4050,816
4200,908
4350,902
4500,1007
```

Execution Times Graph: