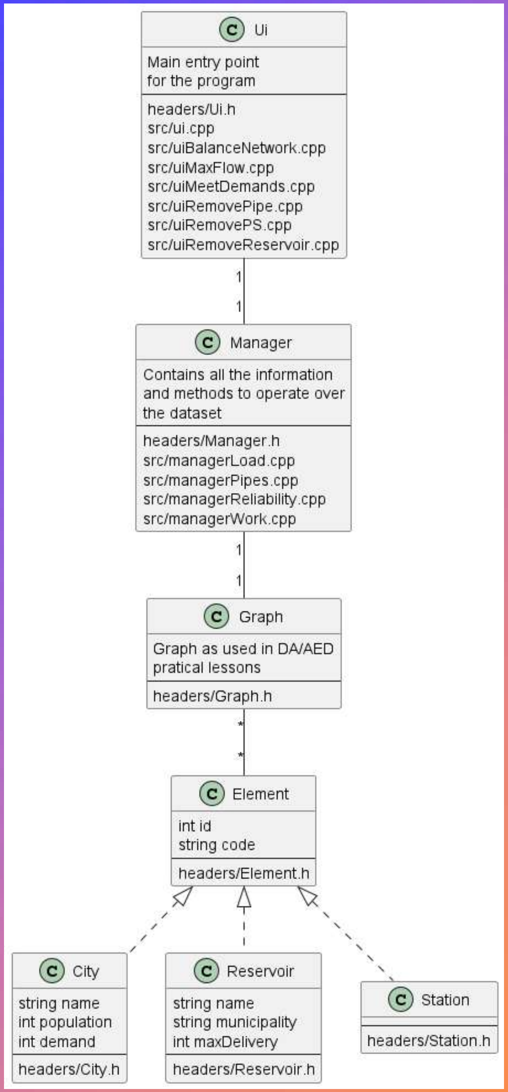
Eduardo Ferreira (up202206628)
Gabriel Lima (up202206693)
Xavier Martins (up202206632)

FEUP – LEIC016
Desenho de Algoritmos

ANALYSIS TOOL FOR WATER SUPPLY MANAGEMENT



Class Diagram

This diagram can be also found in better resolution in:

(base folder)/docs/ClassScheme.png

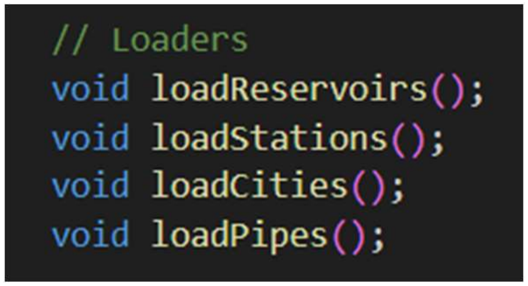
Reading the dataset

Functions

There are 4 function responsible for reading the dataset:

- loadReservoirs()
- loadStations()
- loadCities()
- loadPipes()

Each one read the respective file, create the respective element and add it to the graph.



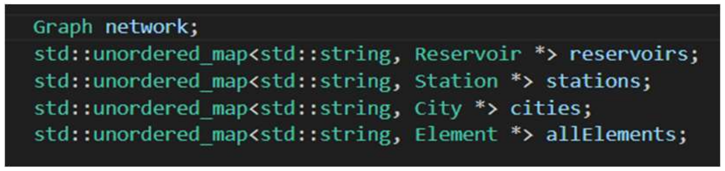
```
// Loaders
void loadReservoirs();
void loadStations();
void loadCities();
void loadPipes();
```

Reading the dataset

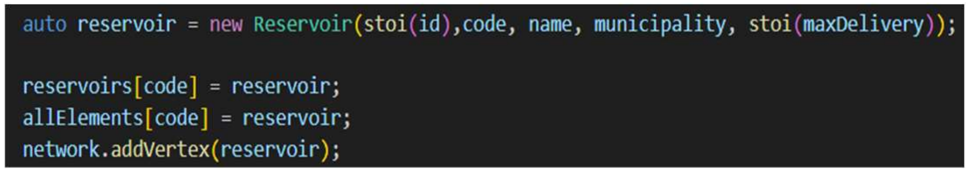
Store Information

The reservoirs, cities and stations are stored in the Graph as Vertexes. The pipes represent edges of the graph, since they are the connection between the various elements.

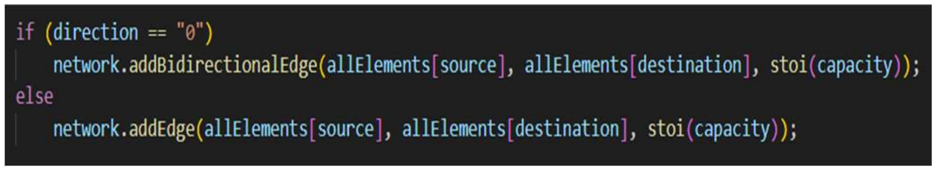
In addition, the reservoirs, cities and stations are stored each one in unordered maps in order to facilitate their future use.



```
Graph network;
std::unordered_map<std::string, Reservoir *> reservoirs;
std::unordered_map<std::string, Station *> stations;
std::unordered_map<std::string, City *> cities;
std::unordered_map<std::string, Element *> allElements;
```



```
auto reservoir = new Reservoir(stoi(id),code, name, municipality, stoi(maxDelivery));

reservoirs[code] = reservoir;
allElements[code] = reservoir;
network.addVertex(reservoir);
```

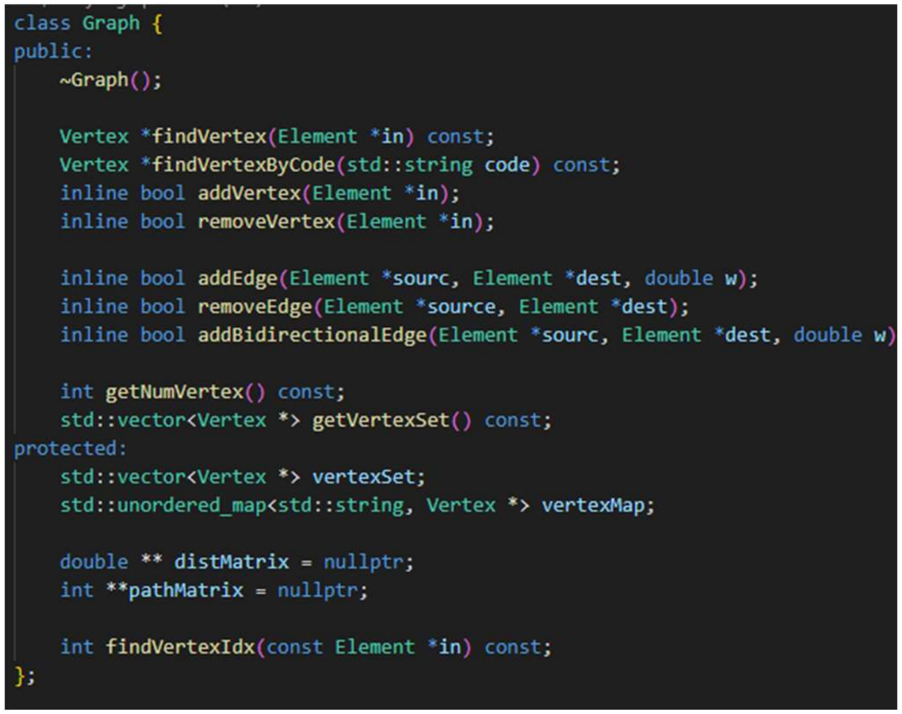


```
if (direction == "0")
    network.addBidirectionalEdge(allElements[source], allElements[destination], stoi(capacity));
else
    network.addEdge(allElements[source], allElements[destination], stoi(capacity));
```

Graph
Representation

Our graph class is almost the same as used in the AED/DA practical lessons, but with some alterations. The most notable changes are:

- Remove templates, given our use case, to only use a pointer to our base abstracted class Element.
- Add an unordered map in addition to the vector of vertexes to have fast access (basically O(1) in most cases) to any element in the network by its code (as they are unique).
- We also removed some functions and variables not required for any algorithm in this project.



```
class Graph {
public:
    ~Graph();

    Vertex *findVertex(Element *in) const;
    Vertex *findVertexByCode(std::string code) const;
    inline bool addVertex(Element *in);
    inline bool removeVertex(Element *in);

    inline bool addEdge(Element *sourc, Element *dest, double w);
    inline bool removeEdge(Element *source, Element *dest);
    inline bool addBidirectionalEdge(Element *sourc, Element *dest, double w)

    int getNumVertex() const;
    std::vector<Vertex *> getVertexSet() const;
protected:
    std::vector<Vertex *> vertexSet;
    std::unordered_map<std::string, Vertex *> vertexMap;

    double ** distMatrix = nullptr;
    int **pathMatrix = nullptr;

    int findVertexIdx(const Element *in) const;
};
```

Graph

Edges and vertexes

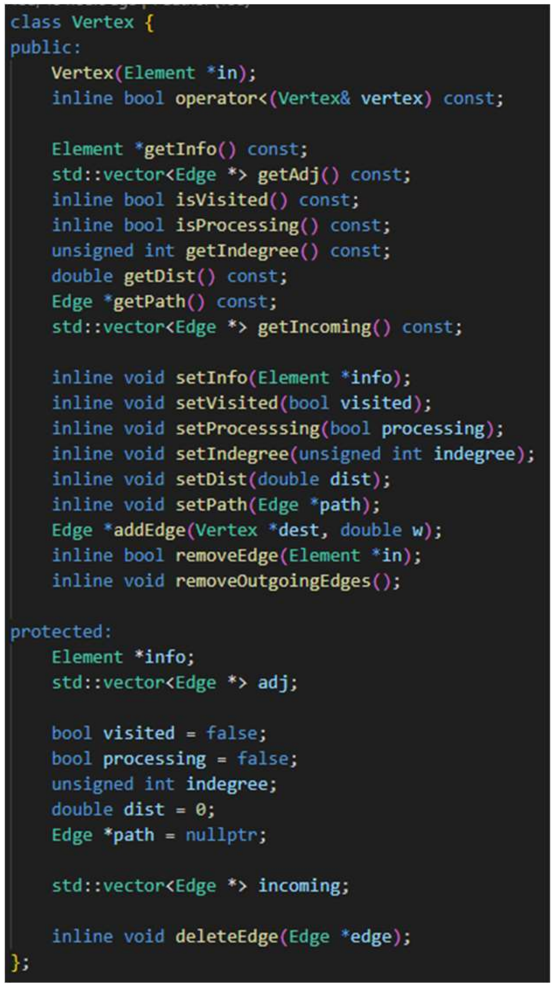
Edges and Vertexes are mostly unaltered, apart from the addition of the variables required to run the Edmonds-Karp algorithm, those being:

Edge:

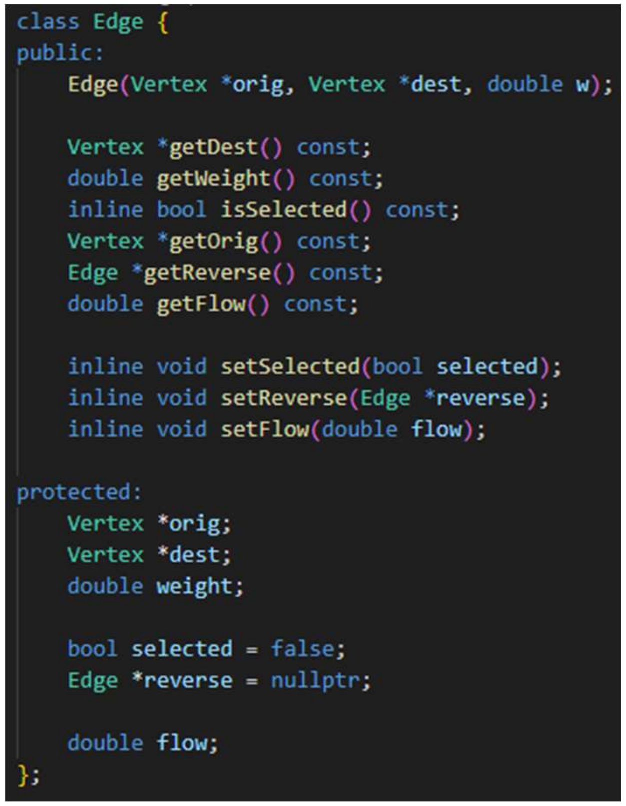
- flow of an edge
- reverse of the edge

Vertex:

- vector of incoming edges.



```cpp
class Vertex {
public:
    Vertex(Element *in);
    inline bool operator<(Vertex& vertex) const;

    Element *getInfo() const;
    std::vector<Edge *> getAdj() const;
    inline bool isVisited() const;
    inline bool isProcessing() const;
    unsigned int getIndegree() const;
    double getDist() const;
    Edge *getPath() const;
    std::vector<Edge *> getIncoming() const;

    inline void setInfo(Element *info);
    inline void setVisited(bool visited);
    inline void setProcesssing(bool processing);
    inline void setIndegree(unsigned int indegree);
    inline void setDist(double dist);
    inline void setPath(Edge *path);
    Edge *addEdge(Vertex *dest, double w);
    inline bool removeEdge(Element *in);
    inline void removeOutgoingEdges();

protected:
    Element *info;
    std::vector<Edge *> adj;

    bool visited = false;
    bool processing = false;
    unsigned int indegree;
    double dist = 0;
    Edge *path = nullptr;

    std::vector<Edge *> incoming;

    inline void deleteEdge(Edge *edge);
};
```



```cpp
class Edge {
public:
    Edge(Vertex *orig, Vertex *dest, double w);

    Vertex *getDest() const;
    double getWeight() const;
    inline bool isSelected() const;
    Vertex *getOrig() const;
    Edge *getReverse() const;
    double getFlow() const;

    inline void setSelected(bool selected);
    inline void setReverse(Edge *reverse);
    inline void setFlow(double flow);

protected:
    Vertex *orig;
    Vertex *dest;
    double weight;

    bool selected = false;
    Edge *reverse = nullptr;

    double flow;
};
```

Implemented functionalities

Max flow and cities in deficit

Max Flow

The max flow functionality pretends to show the max flow of each city or a specific city and the general max flow of the network. To do that, we applied the Edmonds Karp algorithm to our network and calculated the sum of the flow of the incoming edges that is the flow of the city.

Complexity
O(VE^2)

Cities in Deficit

The cities in deficit functionality pretends to show the deficit of flow of the cities. It uses the flow calculated in the previous function and compare it with the necessary demand of the city.

Complexity
O(n)

Implemented functionalities

Balancing the network

Our goal with this function was to create an algorithm capable of minimizing the empty space in the network while redirecting the water to pipes less full. To achieve this, we thought about using the Ford Fulkerson algorithm but always sending the flow through the biggest augmenting path (the one which increased the flow in more pipes). This way, the same amount of water sent would fill more pipes and occupy more space in the network.

The logic behind choosing the augmenting path can very easily be slightly changed in this algorithm, allowing for small adjustments that could further improve its result. All this while being able to maintain a constant value for max-flow.

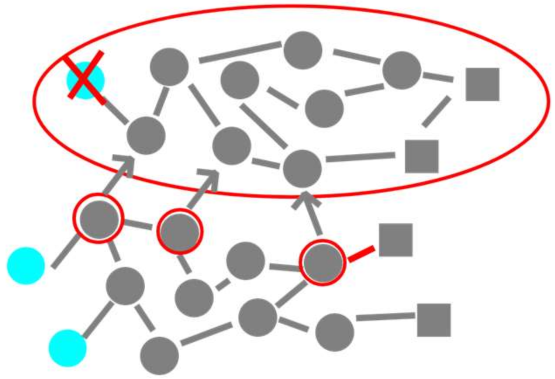
Complexity
O(NVE^2)

Implemented functionalities
Removing a reservoir

For this algorithm, the idea was to first find which vertices (cities, reservoirs and pumping stations) could potentially be affected by the removal of the selected reservoir.

This way, it would be sufficient rerunning the max-flow algorithm for only the affected area of the graph, sending the flow from the vertices connecting the two parts of the graph, as shown in the picture. The vertices considered as being affected are the ones which could have water from the removed reservoir or can have water redirected to them as a response to the increased free space.

Complexity
O(VE^2)



Implemented functionalities

Removing a pumping station

To determine how removing a pumping station could affect the entire network, the algorithm creates a copy of the network and removes the station from this copy. Then the max flow algorithm is run in this modified copy.

Finally, the flow values are compared with the initially obtained in the max flow calculation for the original graph. If the flow is less, we can consider the city affected by the removal.

Complexity
O(NVE^2)

Removing a pipeline

The same logic applies when removing pipes, but this time, we are removing pipes one by one. After the removal, the max flow is calculated again for the copy.

Finally, the results are saved in a map inside the manager.

Complexity
O(NV^2E^3)

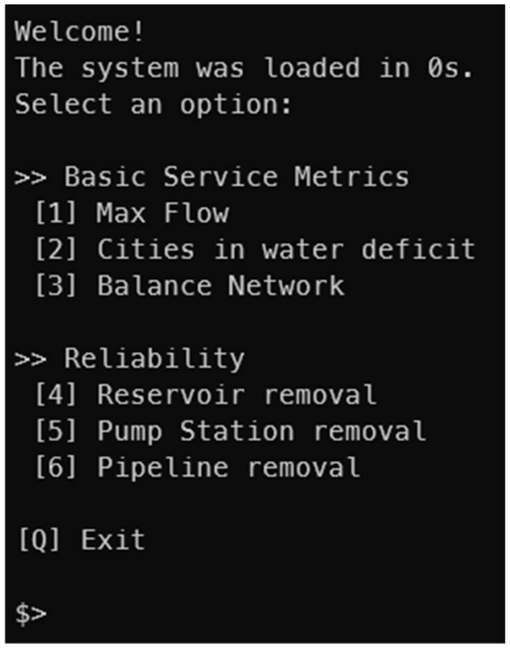
Interface
Basic Usage

Our program is an CLI and takes inspiration from the terminal, **waiting for the user to input a command**. If the command is invalid, a message will be displayed. All available commands are always displayed above user input.

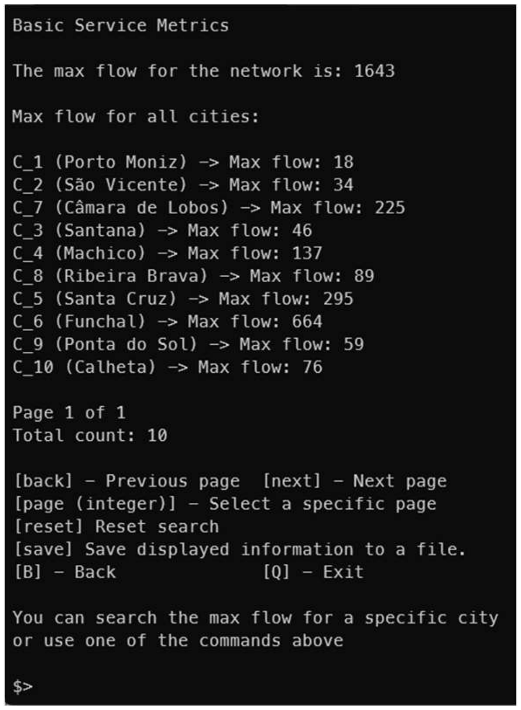
Both small and big datasets can be used with the program. The **big dataset** will be loaded by **default**. To load the **small dataset**, add **any argument** when calling the executable.

By our choice and to allow seamless navigation, most algorithms are run at start-up instead of when accessing their respective menus. Consequently, when using the big dataset, there is a ~6s load time.

Main Menu
The "landing page" for the application. From here, all algorithms and their output can be accessed.



```
Welcome!
The system was loaded in 0s.
Select an option:

>> Basic Service Metrics
 [1] Max Flow
 [2] Cities in water deficit
 [3] Balance Network

>> Reliability
 [4] Reservoir removal
 [5] Pump Station removal
 [6] Pipeline removal

[Q] Exit

$>
```



```
Basic Service Metrics

The max flow for the network is: 1643

Max flow for all cities:

C_1 (Porto Moniz) -> Max flow: 18
C_2 (São Vicente) -> Max flow: 34
C_7 (Câmara de Lobos) -> Max flow: 225
C_3 (Santana) -> Max flow: 46
C_4 (Machico) -> Max flow: 137
C_8 (Ribeira Brava) -> Max flow: 89
C_5 (Santa Cruz) -> Max flow: 295
C_6 (Funchal) -> Max flow: 664
C_9 (Ponta do Sol) -> Max flow: 59
C_10 (Calheta) -> Max flow: 76

Page 1 of 1
Total count: 10

[back] - Previous page  [next] - Next page
[page (integer)] - Select a specific page
[reset] Reset search
[save] Save displayed information to a file.
[B] - Back              [Q] - Exit

You can search the max flow for a specific city
or use one of the commands above

$>
```

Search Menu Example
In most menus where a lot of results are shown, there is a search functionality. Just type the query and a list of results will be shown.

Interface

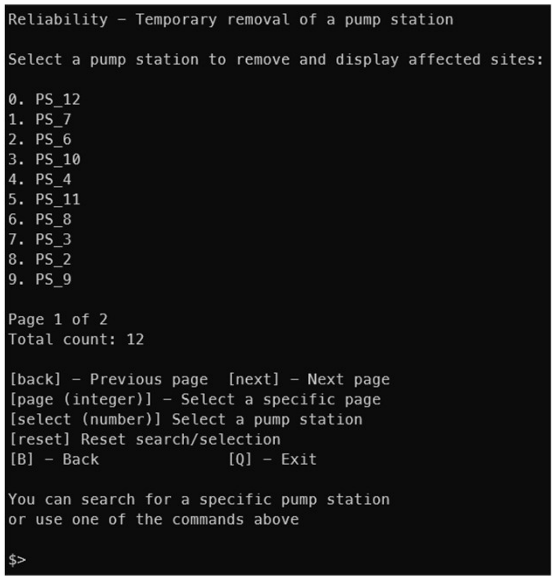
Advanced Usage and Notes

While most sections of the program directly show the output resulting from a specific algorithm, some of these algorithms need user input.

In these cases, a page with all available selection options is presented to the user and he must choose one using "select (number)". This number is shown before the entry.

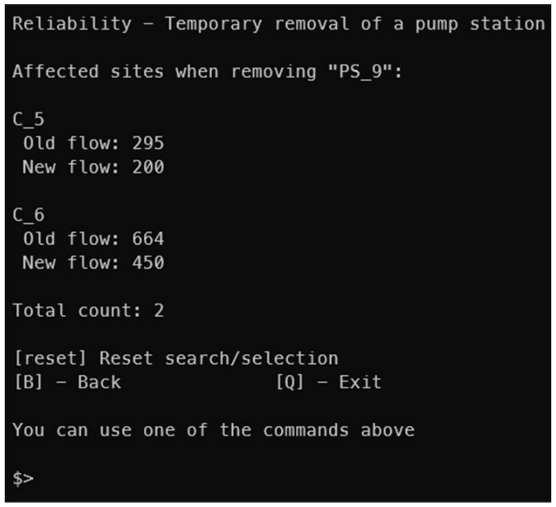
After selection, the results from the algorithm chosen with respect to the selection will be shown.

Important note: Some terminals do not support "cls" or "clean" system calls (e.g. CLion integrated terminal). Consequently, output won't be as visible in such environments.



```
Reliability - Temporary removal of a pump station

Select a pump station to remove and display affected sites:

0. PS_12
1. PS_7
2. PS_6
3. PS_10
4. PS_4
5. PS_11
6. PS_8
7. PS_3
8. PS_2
9. PS_9

Page 1 of 2
Total count: 12

[back] - Previous page  [next] - Next page
[page (integer)] - Select a specific page
[select (number)] Select a pump station
[reset] Reset search/selection
[B] - Back              [Q] - Exit

You can search for a specific pump station
or use one of the commands above

$>
```

Search Menu.

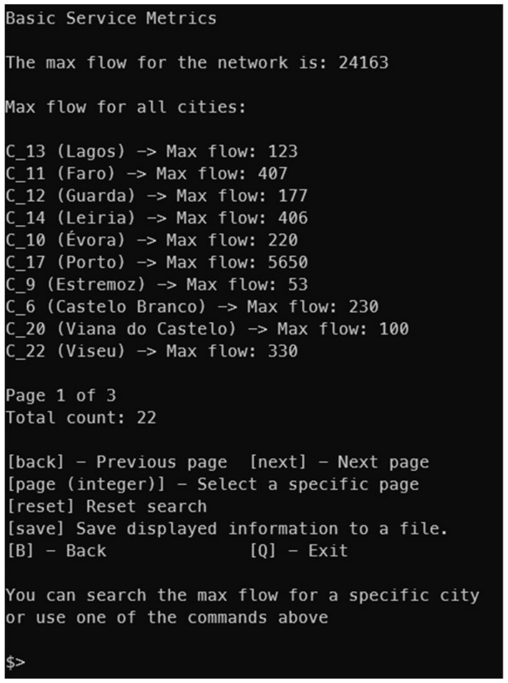


```
Reliability - Temporary removal of a pump station

Affected sites when removing "PS_9":

C_5
 Old flow: 295
 New flow: 200

C_6
 Old flow: 664
 New flow: 450

Total count: 2

[reset] Reset search/selection
[B] - Back              [Q] - Exit

You can use one of the commands above

$>
```

Result Menu.

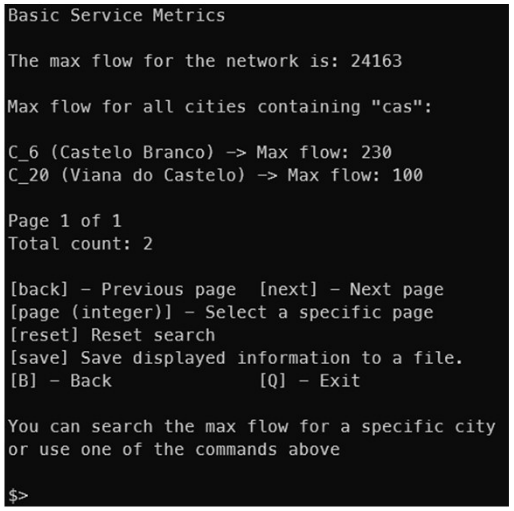
Work highlights

- Calculation of Max Flow and its presentation



```
Basic Service Metrics

The max flow for the network is: 24163

Max flow for all cities:

C_13 (Lagos) -> Max flow: 123
C_11 (Faro) -> Max flow: 407
C_12 (Guarda) -> Max flow: 177
C_14 (Leiria) -> Max flow: 406
C_10 (Évora) -> Max flow: 220
C_17 (Porto) -> Max flow: 5650
C_9 (Estremoz) -> Max flow: 53
C_6 (Castelo Branco) -> Max flow: 230
C_20 (Viana do Castelo) -> Max flow: 100
C_22 (Viseu) -> Max flow: 330

Page 1 of 3
Total count: 22

[back] - Previous page  [next] - Next page
[page (integer)] - Select a specific page
[reset] Reset search
[save] Save displayed information to a file.
[B] - Back               [Q] - Exit

You can search the max flow for a specific city
or use one of the commands above

$>
```

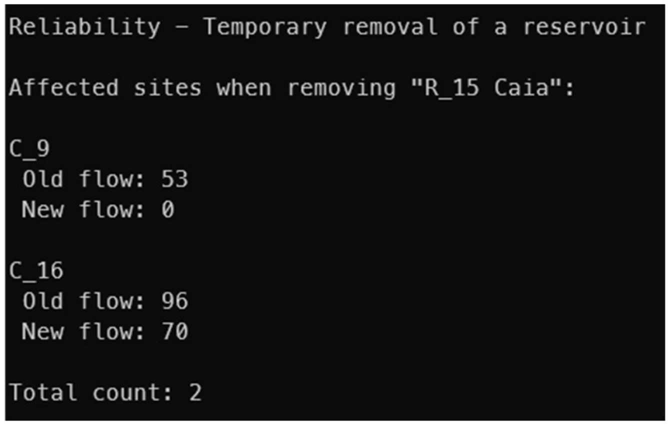
All results



```
Basic Service Metrics

The max flow for the network is: 24163

Max flow for all cities containing "cas":

C_6 (Castelo Branco) -> Max flow: 230
C_20 (Viana do Castelo) -> Max flow: 100

Page 1 of 1
Total count: 2

[back] - Previous page  [next] - Next page
[page (integer)] - Select a specific page
[reset] Reset search
[save] Save displayed information to a file.
[B] - Back               [Q] - Exit

You can search the max flow for a specific city
or use one of the commands above

$>
```

Results containing “cas”

- Calculation of affected cities are affected by the removal of X element.



```
Reliability - Temporary removal of a reservoir

Affected sites when removing "R_15 Caia":

C_9
 Old flow: 53
 New flow: 0

C_16
 Old flow: 96
 New flow: 70

Total count: 2
```

Main difficulties

The main difficulties for us in this project were the algorithms that affect the graph’s edges, that is, functionalities T3.1 to T3.3.

Most problems we had arisen from not being able to correctly temporarily remove elements from the graph and afterwards reinstating them to the Graphs original state.

Group Participation

All the group members had a similar participation in the project. That is around ~33%, give or take.

END