U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

L.EIC Water Supply

Project 1

Design of Algorithms

April 2024

Afonso Domingues up202207313

Jorge Mesquita up202108614

Tatiana Lin up202206371

Table of Contents

01 Task

02 Classes

03 Functionalities

Descrição das funções mais relevantes

04 Test Cases

Doxygen

01

Task

Task description

Management of a Water Supply Network

1. Read and Parse the Input Data;

2. Basic Service Metrics:
    a. Maximum amount of water that can reach each or a specific city;
    b. Check if it meets the water needs of its customers;
    c. Balance the network;

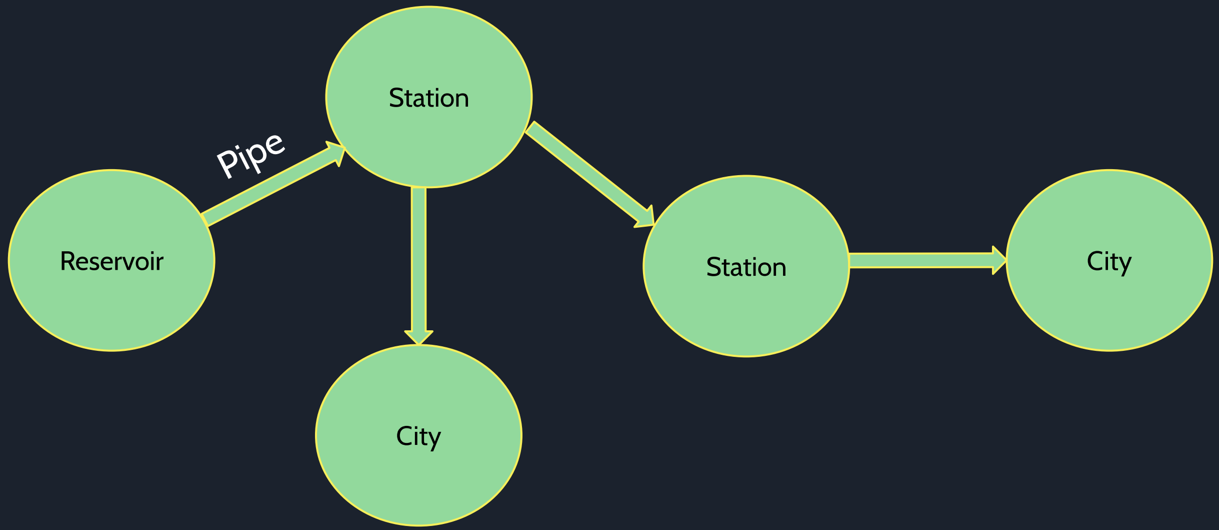
3. Reliability and Sensitivity to Failures;
    a. Analysis of the effect of the removal of a pumping station, reservoir or a pipe;

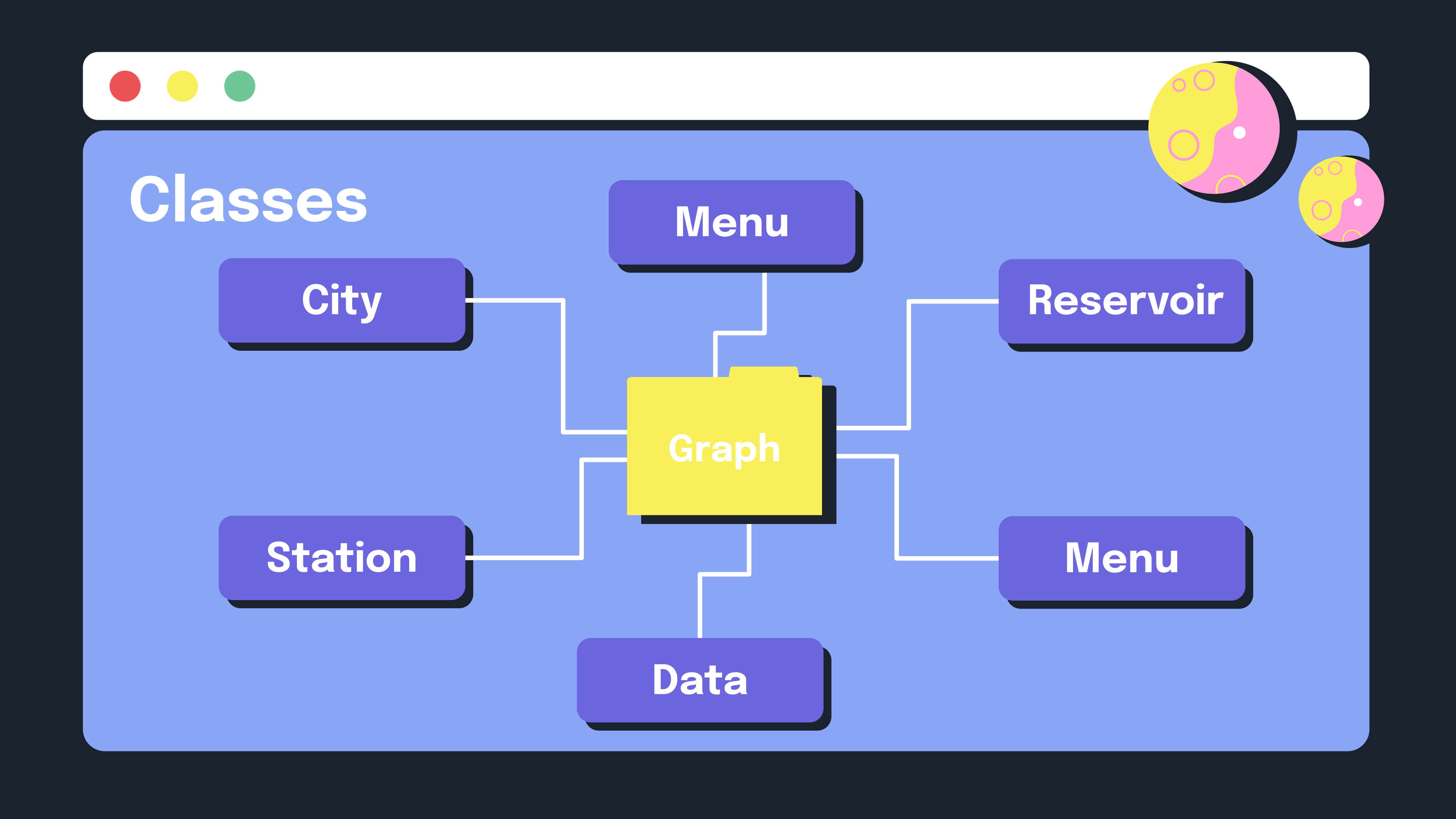
02

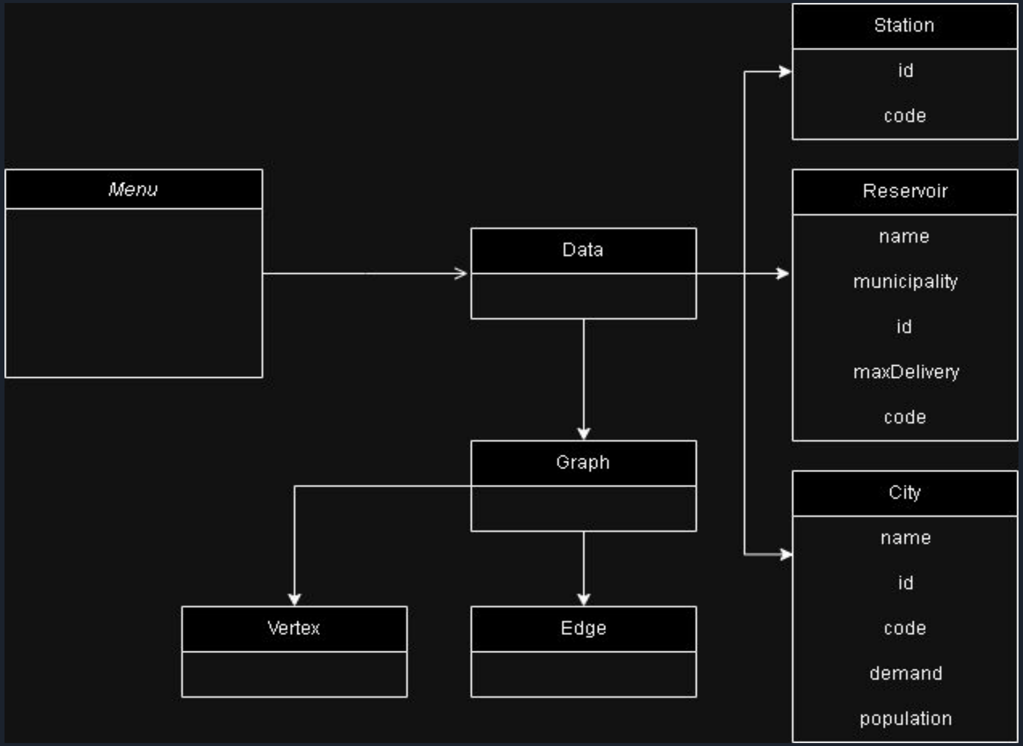
Classes

Class representation

Description of the Graph




Classes
Menu
City
Reservoir
Graph
Station
Menu
Data



Structure of the Project

03

Functionalities

Read and Parse the Input Data

```
void Data::parseReservoir() {
    ifstream reservoirs( s: "../dataset/Reservoir.csv");
    string line;
    getline( &: reservoirs, &: line); //read and ignore first line
     while (getline( &: reservoirs,  &: line)) {
         string name, municipality, id, maxDelivery, code;
         istringstream iss( str: line);
         getline( &: iss,  &: name,  delim: ',');
         getline( &: iss,  &: municipality,  delim: ',');
         getline( &: iss,  &: id,  delim: ',');
         getline( &: iss,  &: code,  delim: ',');
         getline( &: iss,  &: maxDelivery);
         Reservoir r = Reservoir(name,municipality,id,  maxDelivery: stod( str: maxDelivery),code);
         reservoirs_[code] = r;
         supply.addVertex( in: code);
     }

}
```

Complexity: O(n) (Same for City and Station!)

Read and Parse the Input Data

```
void Data::parsePipes() {
    ifstream pipes( s: "../dataset/Pipes.csv");
    string line;
    getline( &: pipes, &: line); //read and ignore first line
    while (getline( &: pipes,  &: line)) {
        string source, target, capacity, direction;
        istringstream iss( str: line);
        getline( &: iss,  &: source,  delim: ',');
        getline( &: iss,  &: target,  delim: ',');
        getline( &: iss,  &: capacity,  delim: ',');
        getline( &: iss,  &: direction);

        if(direction == "0"){
            supply.addBidirectionalEdge( sourc: source, dest: target,  w: stod( str: capacity));
        }
        else supply.addEdge( sourc: source, dest: target,  w: stod( str: capacity));
    }
}
```

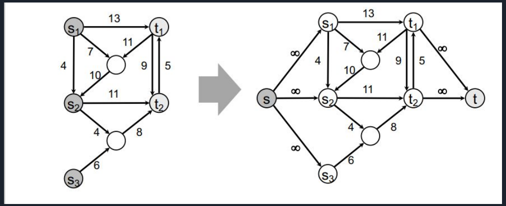
Complexity: O(n)

Maximum amount of water that can reach each or a specific city

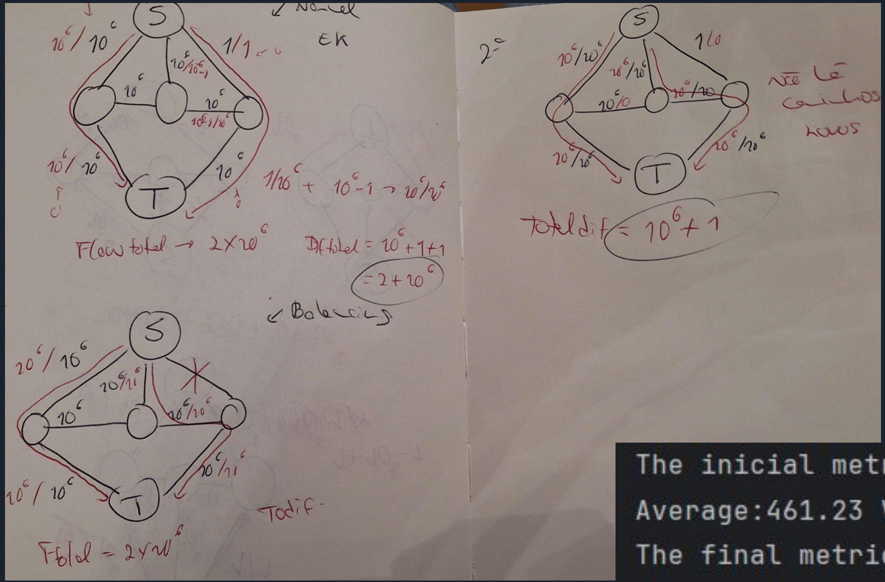
```cpp
list<pair<City,double>> Menu::edmondsKarp(Graph<string> g) {
    list<pair<City,double>> r;
    string super_source = "SS";
    string super_target = "ST";
    g.addVertex(super_source);
    g.addVertex(super_target);
    for(Vertex<string>* v : g.getVertexSet()){...}
    //create s super sink
    for(Vertex<string>* v : g.getVertexSet()){
        if(v->getInfo()[0] == 'C') g.addEdge(v->getInfo(), super_target, d.getCities()[v->getInfo()].getDemand());
    }
    Vertex<string>* s = g.findVertex(super_source);
    Vertex<string>* t = g.findVertex(super_target);


    for (auto v : g.getVertexSet()) {
        for (auto e : v->getAdj()) {
            e->setFlow(0);
        }
    }
    while( findAugmentingPath(&g, s, t) ) {
        double f = findMinResidualAlongPath(s, t);
        augmentFlowAlongPath(s, t, f);
    }
```

```cpp
    for(Vertex<string>* v : g.getVertexSet()){
        if(v->getInfo()[0] == 'C'){
            double value = 0.0;
            for(auto e : v->getIncoming()){
                value += e->getFlow();
            }
            City temp = d.getCities()[v->getInfo()];
            r.push_back(make_pair(temp, value));

        }
    }

    for(Vertex<string>* v : g.getVertexSet()){
        if(v->getInfo()[0] == 'R')
            g.removeEdge(super_source, v->getInfo());
    }
    //create s super sink
    for(Vertex<string>* v : g.getVertexSet()){
        if(v->getInfo()[0] == 'C') g.removeEdge(v->getInfo(), super_target);
    }
    g.removeVertex(super_source);
    g.removeVertex(super_target);

    return r;
```

Complexity: O(VE^2)



Balance the network



```
while(delta >= 1) {
    for(auto v : g.getVertexSet()){
        for(auto e : v->getAdj()){
            if(e->getWeight() < delta){
                restore_weights[e] = e->getWeight();
                e->setWeight(0);
            }
        }
    }
    while (findAugmentingPath(&g, s, t)) {
        double f = findMinResidualAlongPath(s, t);
        augmentFlowAlongPath(s, t, f);
    }

    for(auto p : restore_weights){
        p.first->setWeight(p.second);
    }

    delta /= 2;
}
```

```
The inicial metrics are:
Average:461.23 Variance:918508.06 Max-Difference:4000.00
The final metrics are:
Average:403.91 Variance:541759.91 Max-Difference:4000.00
```

Complexity: `O((V + E) * log(delta))`

Analysis of the effect of the removal of a pumping station, reservoir or a pipe

```
1.Remove the element from the graph (Pumping station, reservoir or pipe);

2.Apply the max flow algorithm;

3.Analyze the metrics;

4.Display the affected cities.
```

Complexity: O(VE^2 + f(n) (overall)
O(VE(VE^2+f(n))) for the pipes of each city!

Analysis of the effect of the removal of reservoir

```
bool Menu::Remove_Water_Reservoir(std::string reservoir_code,Graph<string> s) {
    if(d.getReservoirs().find( x: reservoir_code) == d.getReservoirs().end()) return false;
    list<pair<City,double>> l = Meet_Costumer_needs( a: s);
    set<string> cities_affected;
    for(auto p : pair<City, double>  : l) cities_affected.insert( x: p.first.getCodeCity());
    unordered_map<string,double> temp;
    for(auto p : pair<City, double>  : l) temp[p.first.getCodeCity()] = (p.first.getDemand() - p.second);
    unordered_map<Edge<string>*,double> restore_weights;
    Vertex<string>* v = s.findVertex( in: reservoir_code);
    for(auto e : Edge<string> * : v->getIncoming()){
        restore_weights[e] = e->getWeight();
        e->setWeight(0.0);
    }

    for(auto e : Edge<string> * : v->getAdj()){
        restore_weights[e] = e->getWeight();
        e->setWeight(0.0);
    }

    list<pair<City,double>> r = edmondsKarp( g: s);
    //restore weights
```

```
for(auto e : Edge<string> * : v->getIncoming()){
    e->setWeight(restore_weights[e]);
}

for (auto e : Edge<string> * : v->getAdj()) {
    e->setWeight(restore_weights[e]);
}
bool flag = false;
for(auto p : pair<City, double>  : r){
    if(p.second < p.first.getDemand()) {
        if ((cities_affected.find( x: p.first.getCodeCity()) == cities_affected.end()) || (temp[p.first.getCodeCity()] > p.second)) {
            flag = true;
            break;
        }
    }
}
if (!flag) {
    cout << "None of the cities were affected by the removal!\n";
    return true;
}
```

Complexity: O(VE^2 + f(n)) (overall)
O(VE(VE^2+f(n))) for the pipes of each city!

```
cout << "The affected cities by the removal of the Reservoi are:\n";
for(auto p : pair<City, double>  : r){
    if(p.second < p.first.getDemand()) {
        if ((cities_affected.find( x: p.first.getCodeCity()) == cities_affected.end()) || (temp[p.first.getCodeCity()] > p.second))
            cout << p.first.getNameCity() << ' ' << (p.first.getDemand() - p.second) << " m^3 of water in deficit!" << '\n';
    }
}
return true;
```

Analysis of all the critical pipes for each and a specific city

```
1.Iterate over all the pipes and remove each one of them;

2.Apply the max flow algorithm;

3.Analyze the metrics;

4.For each affected city, associate the pipe with it.
```

Complexity: complexity $O(VE(VE^2 + f(n)))$ for both algorithms

Analysis of all the critical pipes for each and a specific city

```cpp
void Menu::Critical_Pipe_allCities(Graph<std::string> s) {
    map<string, vector<string>> all_cities;
    unordered_map<string,double> temp;
    list<pair<City,double>> l = Meet_Costumer_needs(s);
    set<string> cities_affected;

    for(auto p : l) cities_affected.insert(p.first.getCodeCity());
    for(auto p : l) temp[p.first.getCodeCity()] = (p.first.getDemand() - p.second);

    for(auto v : s.getVertexSet()){
        for(auto e : v->getAdj()){
            if(!e->isSelected()) {
                bool bidirectional = false;
                if (e->getReverse() != nullptr) bidirectional = true;
                unordered_map < Edge < string > *, double > restore_weights;
                if (bidirectional) {

                    restore_weights[e] = e->getWeight();
                    e->setWeight(0.0);


                    restore_weights[e->getReverse()] = e->getReverse()->getWeight();
                    e->getReverse()->setWeight(0.0);
```

```cpp
                        } else if (temp[p.first.getCodeCity()] > p.second) {
                            all_cities[p.first.getNameCity()].push_back(v->getInfo() + "-->" + e->getDest()->getInfo());

                        }
                    }
                }

            }
        }
    }
    for(auto v : s.getVertexSet()){
        for(auto e : v->getAdj()){
            e->setSelected(false);
        }
    }
    cout << "The critical pipes for each city are: " << endl;
    for (auto it : all_cities) {
        cout << it.first << ":" << endl;
        for (auto it2 : it.second) {
            cout << it2 << endl;
        }
        cout << endl;
    }
}
```

```cpp
                } else {
                    restore_weights[e] = e->getWeight();
                    e->setWeight(0.0);
                }
                //Solve for EdmondKarp
                list <pair<City, double>> r = edmondsKarp(s);
                //Evalute the context

                if (bidirectional) {
                    e->setWeight(restore_weights[e]);
                    e->getReverse()->setWeight(restore_weights[e->getReverse()]);


                    e->setSelected(true);
                    e->getReverse()->setSelected(true);
                } else {
                    e->setWeight(restore_weights[e]);
                    e->setSelected(true);
                }
                //Print result
                for (auto p : r) {
                    if (p.second < p.first.getDemand()) {
                        if (cities_affected.find(p.first.getCodeCity()) == cities_affected.end()) {
                            all_cities[p.first.getNameCity()].push_back(v->getInfo() + "-->" + e->getDest()->getInfo());
```

Complexity: complexity O(VE(VE^2 + f(n)))

Is it possible to apply the max flow algorithm again without applying it from the scratch?

```
function quickMaxflow(Graph g, string reservoir):
      s = createSubGraph(g, reservoir); //Subgraph by removing vertex and edges
      edmondKarp(s);
      evaluate the affected cities;
      end;
```

This algorithm is more efficient because it doesn't apply the max flow algorithm from scratch again, it applies the edmondKarp algorithm only to a portion(subgraph) of the original graph!

User interface

```
 ------------------------------------------
|      WATER SUPPLY MANAGEMENT SYSTEM      |
|                                          |
| [1]  Maximum flow                        |
| [2]  Costumer water needs                |
| [3]  Balance Load across network         |
| [4]  Remove Water Reservoir              |
| [5]  Pumping Station Maintenance         |
| [6]  Station Maintenance - no effect     |
| [7]  Remove Pipe                         |
| [8]  Key pipes for each city             |
|__________________________________________|
Please enter your choice:
```

Highlight

1. Balance Load;

2. Remove multiple pipes;

```
bool Menu::Remove_Pipe2(Graph<string> s,set<pair<string,string>> t) {
    unordered_map<string,double> temp;
    list<pair<City,double>> l = Meet_Costumer_needs( a: s);
    for(auto p : pair<City, double> : l) temp[p.first.getCodeCity()] = (p.first.getDemand() - p.second);
    unordered_map<Edge<string> *, double> restore_weights;
    set<string> cities_affected;
    for (auto p : pair<City, double> : l) cities_affected.insert( x: p.first.getCodeCity());
    for(auto final : pair<string, string> : t) {
        bool bidirectional = false;
        auto v_source : Vertex<string> * = s.findVertex( in: final.first);
        auto v_target : Vertex<string> * = s.findVertex( in: final.second);
        bool exits = false;
        if (v_source == nullptr || v_source == v_target || v_target == nullptr) {
            return false;
        }

        for (auto e : Edge<string> * : v_source->getAdj()) {
            if (e->getDest()->getInfo() == final.second) {
                exits = true;
                break;
            }
        }
```

complexity O(N(VE^2) + f(n))

Highlight

Remove Multiple pipes

```cpp
    for (auto e : v_target->getAdj()) {
        if (e->getDest()->getInfo() == final.first) {
            bidirectional = true;
            break;
        }
    }
    if (!exits) return false;
    //Check for already existent problems with supply
    //Check for invalid source or target
    if (bidirectional) {
        for (auto e : v_source->getAdj()) {
            if (e->getDest()->getInfo() == final.second) {
                restore_weights[e] = e->getWeight();
                e->setWeight(0.0);
                break;
            }
        }
        for (auto e : v_source->getIncoming()) {
            if (e->getOrig()->getInfo() == final.second) {
                restore_weights[e] = e->getWeight();
                e->setWeight(0.0);
                break;
            }
        }
```

```cpp
    } else {
        for (auto e : v_source->getAdj()) {
            if (e->getDest()->getInfo() == final.second) {
                restore_weights[e] = e->getWeight();
                e->setWeight(0.0);
                break;
            }
        }
    }
    //Solve for EdmondKarp
}
list<pair<City, double>> r = edmondsKarp(s);
//Evalute the context
restore_capacities(s,restore_weights);
bool flag = false;
for(auto p : r){
    if(p.second < p.first.getDemand()) {
        if ((cities_affected.find(p.first.getCodeCity()) == cities_affected.end()) || (temp[p.first.getCodeCity()] > p.second)) {
            flag = true;
            break;
        }
    }
}
```

```cpp
if (!flag) {
    cout << "None of the cities were affected by the removal!\n";
    return true;
}
//Print result
cout << "The affected cities by the removal of the Pipe are:\n";
for(auto p : r){
    if(p.second < p.first.getDemand()) {
        if ((cities_affected.find(p.first.getCodeCity()) == cities_affected.end()) || (temp[p.first.getCodeCity()] > p.second))
            cout << p.first.getNameCity() << ' ' << (p.first.getDemand() - p.second) << " m^3 of water in deficit!" << '\n';
    }
}
return true;
```

04

Test Cases

Test Cases

```
Please choose the desired option:1
Porto Moniz - C_1 - 18 m^3 of water supplied!
S├úo Vicente - C_2 - 34 m^3 of water supplied!
Santana - C_3 - 46 m^3 of water supplied!
Machico - C_4 - 137 m^3 of water supplied!
Santa Cruz - C_5 - 295 m^3 of water supplied!
Funchal - C_6 - 664 m^3 of water supplied!
C├ómara de Lobos - C_7 - 225 m^3 of water supplied!
Ribeira Brava - C_8 - 89 m^3 of water supplied!
Ponta do Sol - C_9 - 59 m^3 of water supplied!
Calheta - C_10 - 76 m^3 of water supplied!
The maxflow for the virtual super sink is: 1643
```

T2.1 - Maximum Flow

Expected total flow: 1643

By city:

C_1-Porto Moniz 18

C_2-São Vicente 34

C_3-Santana 46

C_4-Machico 137

C_5-Santa Cruz 295

C_6-Funchal 664

C_7-Câmara de Lobos 225

C_8-Ribeira Brava 89

C_9-Ponta do Sol 59

C_10-Calheta 76

Test Cases

```
Please enter your choice:2
Funchal - C_6 - 76 m^3 of water in deficit!
```

T2.2 - Water Demand vs Actual Flow

C_6-Funchal

- Demand: 740
- Actual Flow: 664
- Deficit: 76

Test Cases

```
Please enter your choice:4
Please insert a valid Reservoi code:R_4
The affected cities by the removal of the Reservoi are:
Machico 1 m^3 of water in deficit!
Santa Cruz 195 m^3 of water in deficit!
Funchal 265 m^3 of water in deficit!
```

T3.1 - Reliability and Sensitivity to Failures (Reservoir)

Case: Reservoir R_4: Ribeiro Frio is removed (it had a maximum delivery of 385 m3/sec)

City	Old Flow	New Flow
C_4: Machico	137	136
C_5: Santa Cruz	295	100
C_6: Funchal	664	475

Test Cases

```
Please enter your choice:5
Please insert a valid Station code:PS_1
The affected cities by the removal of the Station are:
Porto Moniz 18 m^3 of water in deficit!
Santa Cruz 17 m^3 of water in deficit!
Funchal 115 m^3 of water in deficit!
Calheta 26 m^3 of water in deficit!
```

T3.2 - Reliability and Sensitivity to Failures (Pumping Stations)

Case: Pumping Station PS_1 is removed These cities are affected:

City	Old Flow	New Flow
C_5: Santa Cruz	295	278
C_6: Funchal	664	625
C_1: Porto Moniz	18	0
C_10: Calheta	76	50

Test Cases

```
Please choose the desired option:2
Please insert a valid Source code (Station or Reservoir):
PS_9
Please insert a valid Target code (Station or City):
PS_10
To continue removing pipes type C else type any key
C
Please insert a valid Source code (Station or Reservoir):
PS_4
Please insert a valid Target code (Station or City):
PS_5
To continue removing pipes type C else type any key

L
The affected cities by the removal of the Pipe are:
Funchal 168 m^3 of water in deficit!
```

T3.3 - Reliability and Sensitivity to Failures (Pipelines)

Case 1: Only PS_9 - PS_10 is removed. No city if affected.
Case 2: Only PS_4 - PS_5 is removed. No city if affected.
Case 3: Both PS_9 - PS_10 and PS_4 - PS_5 are removed. City affected:

City	Old Flow	New Flow
C_6: Funchal	664	572

Doxygen


My Project
Main Page
Classes
Files
Search
Class List
Here are the classes, structs, unions and interfaces with brief descriptions:
City Identifies the City class by its name, id, code, demand(rate of demand or need of delivery of water) and population (total population)
Data Represents the data that was parsed from the csv files
Edge For this specific project, an edge represents a Pipe that can be either unidirectional or bidirectional that connects to structures
Graph The main class, where the structure used for all the project resides
Menu Represents the user-friendly interface for the water supply management system. It shows available functionalities and their corresponding outputs in a clear, organized, and logical manner, facilitating seamless program utilization and straightforward result interpretation
MutablePriorityQueue
Reservoir
Station Identifies the Station class by its id and code
Vertex For this specific project, a vertex's info is the code of any of the multiple structures that composes the water supply network(Pumping Stations, Cities or Reservoirs)
Generated by doxygen 1.9.8

Main Difficulty

In our project, the primary challenge we encountered revolved around the implementation of the balancing function. The balancing function played a crucial role in ensuring the stability and efficiency of our system. Its responsibility was to distribute the flow evenly across the system , ensuring optimal performance while preventing bottlenecks.

Work Distribution percentages:

- Afonso Domingues 70%
- Jorge Mesquita 25%
- Tatiana Lin 5%

Fim!