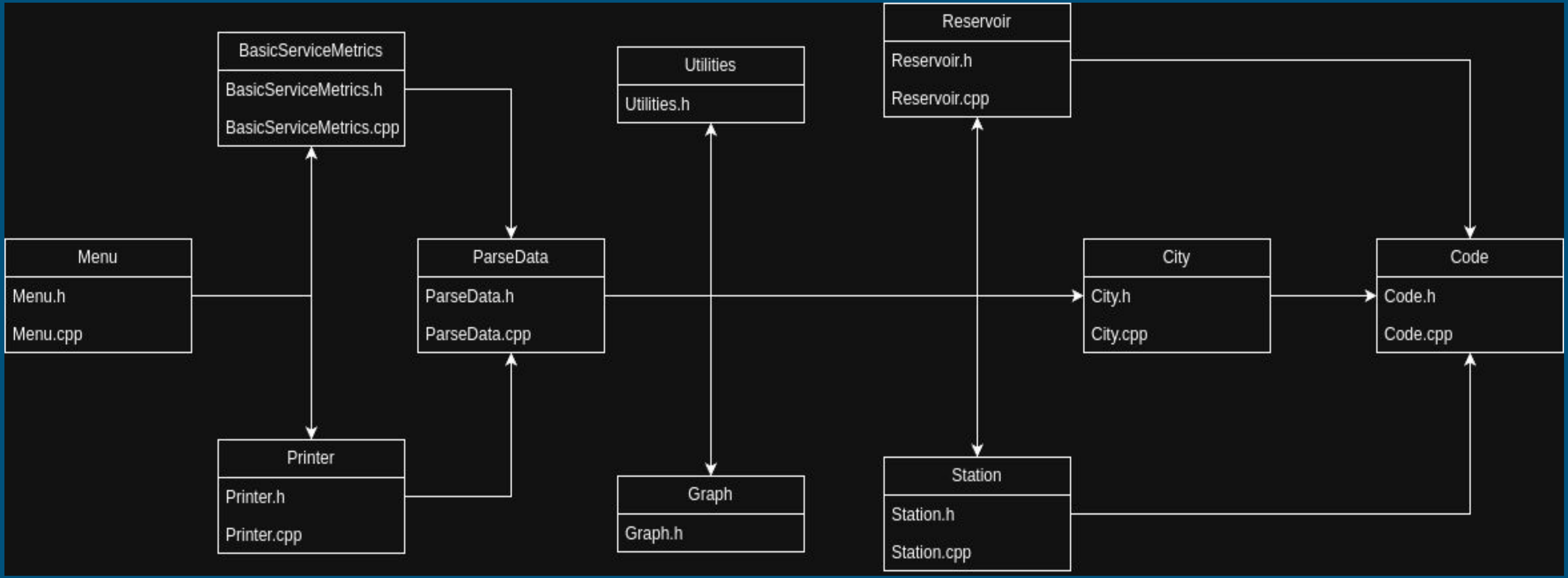
LUSO ANALYTICS

DA PROJECT 1

G07_9: Bruno Huang 202207517
Ricardo Yang 202208465

Class Diagram



Dataset Reading Description

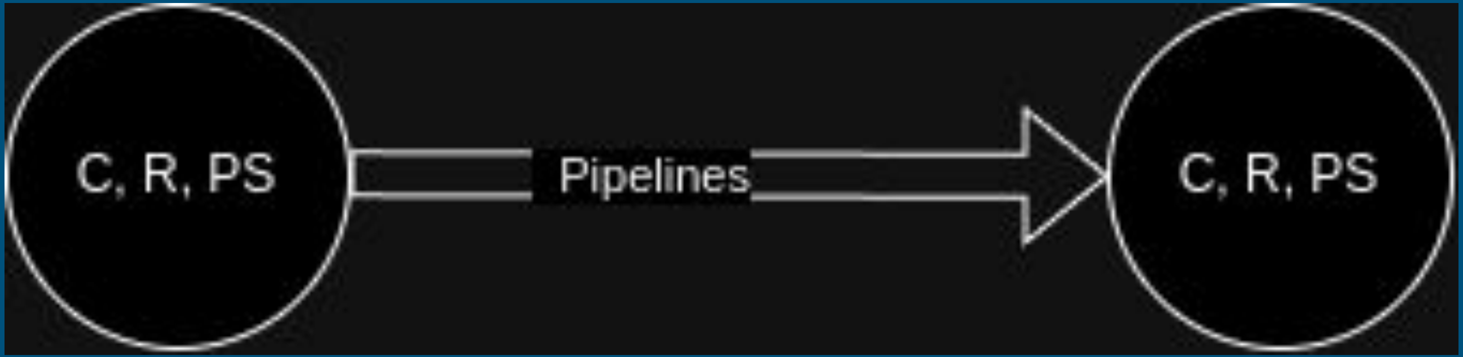
We created a class Code where it had a enum class CodeType to represent the type of a Code (RESERVOIR, STATION, CITY), then we read the csv files and created vertices with Code class and the edge between those vertices representing the pipelines.

The vertices info only stored Code (i.e. C_2, R_6, PS_5), for more info we created a class DataContainer, that had 3 hash tables (unordered maps) for each Code type so we could achieve a constant search time complexity if we needed the whole info about the vertex.

Graph Description

Vertices represent either Cities, Reservoirs or Pumping Stations.

Edges represent the pipelines that connect Cities, Reservoirs or Pumping Stations.



List of Implemented Functions

- `void resetBSMGraph()`

  Algorithm used: Edmonds-Karp

  Time complexity: O(V * E^2)

- `void edmondsKarp()`

  Algorithm used: Edmonds-Karp

  Time complexity: O(V * E^2)

List of Implemented Functions

- `unordered_map<Vertex<Code>*, double> pumpRemainingWaterFromReservoirs()`
- `void distributeExtraFlow(Vertex<Code> *vertex, double extraFlow)`
- `void balanceFlow()`

Algorithm used: DFS

Time complexity:

- O(V * E)
- O(E logE)
- O(V * E logE)

List of Implemented Functions

- `void removeReservoir(const Code& reservoirCode)`

  Algorithm used: Edmonds-Karp

  Time complexity: O(V * E^2)

- `void removePumpingStation(const Code& stationCode)`

  Algorithm used: Edmonds-Karp

  Time complexity: O(V * E^2)

List of Implemented Functions

```
void removePipes(const vector<pair<Code,Code>> pipeCodes)
```

Algorithm used: Edmonds-Karp

Time complexity: O(V * E^2)

```
map<int,double> getCitiesFlow()
```

Time complexity: O(C * V)

User Interface

```
Choose dataset:
1. Portugal Continental (large dataset)
2. Madeira (small dataset)

Enter your choice (1 or 2):
```

```
Portuguese Water Management Tool

[1] Specific City Maximum Flow
[2] Total Maximum Flow
[3] Cities With Water Deficit
[4] Each Pipe Flow Difference And Initial Metrics
[5] Balance Graph
[6] Remove Reservoir
[7] Remove Pumping Station
[8] Remove Pipes
[9] Export All Cities Max Flow
[10] Export Critical Pipes per City
[11] Export Data Container
[12] ⚠ Reset Graph (for balance) ⚠
[0] EXIT

Enter your choice:
```

User Interface

```
[1] Specific City Maximum Flow
[2] Total And Each City Maximum Flow
[3] Cities With Water Deficit
```

[1]

```
Enter your choice: 1

Enter the City ID number (eg. C_9, enter 9): 5
Name                |    Code    |    Max Flow
--------------------|------------|-------------
Santa Cruz          |    C_5     |    295

Press ENTER to continue...
```

[2]

```
Enter your choice: 2

Total max flow: 1643
Name                |    Code    |    Max Fl
--------------------|------------|----------
Porto Moniz         |    C_1     |    18
São Vicente         |    C_2     |    34
Santana             |    C_3     |    46
Machico             |    C_4     |    137
Santa Cruz          |    C_5     |    295
Funchal             |    C_6     |    664
Câmara de Lobos     |    C_7     |    225
Ribeira Brava       |    C_8     |    89
Ponta do Sol        |    C_9     |    59
Calheta             |    C_10    |    76

Press ENTER to continue...
```

[3]

```
Enter your choice: 3
Name                |    Code    |    Deficit    |    Max Flow   |    Demand
--------------------|------------|---------------|---------------|-------------
Funchal             |    C_6     |    76         |    664        |    740

Press ENTER to continue...
```

User Interface

[4] Each Pipe Flow Difference And Initial Metrics

[4]

Enter your choice: 4

Note: * represents the pipes that are bidirectional

Orig	Dest	Flow/Capacity	Difference
R_1	PS_1	100/100	0
R_1	PS_2	400/400	0
R_1	PS_12	25/250	225
R_2	PS_11	138/150	12
R_2	PS_12	45/150	105
R_3	PS_4	250/250	0
R_3	PS_10	300/300	0
R_3	PS_11	0/100	100
R_4	PS_4	135/150	15
R_4	PS_6	50/100	50
R_4	PS_7	50/50	0
R_4	PS_9	100/100	0
R_4	PS_10	50/50	0
PS_1	C_1	18/20	2
PS_1	C_10	26/40	14
PS_1	PS_2	56/400	344
PS_2	C_2	20/20	0
PS_2	PS_3	436/480	44
PS_3	C_2	4/20	16
*PS_5	*PS_6	291/750	459
PS_6	C_3	20/20	0
*PS_6	*PS_7	321/750	429
PS_7	C_4	80/80	0
*PS_7	*PS_8	291/600	309
PS_8	C_4	57/80	23
PS_8	C_5	195/200	5
*PS_8	*PS_9	39/600	561
PS_9	C_5	100/100	0
PS_9	C_6	214/500	286
PS_10	C_6	450/450	0
PS_10	C_7	225/300	75
*PS_10	*PS_9	175/500	325
PS_11	C_8	89/100	11
PS_11	C_9	39/40	1
PS_11	C_2	10/10	0
PS_12	C_9	20/20	0
PS_12	C_10	50/50	0

Max difference: 561

Sum of differences: 4067

Average difference: 4067 / 42 (#pipes) ≈ 96.8333

Variance: 24769

Press ENTER to continue...

User Interface

[5] Balance Graph

[5]

```
Enter your choice: 5

Balance algorithm successfully terminated, please check [4] again to see the differences
⚠ AFTER THIS PLEASE ENSURE THAT YOU RESET THE GRAPH TO AS IT WAS BY CLICKING [12] ⚠
```

Before (Madeira)

```
Max difference: 750
Sum of differences: 8867
Average difference: 8867 / 51 (#pipes) ≈ 173.863
Variance: 25284
```

After (Madeira)

```
Max difference: 750
Sum of differences: 8670
Average difference: 8670 / 51 (#pipes) ≈ 170
Variance: 25712
```

Before (Continental)

```
Max difference: 14000
Sum of differences: 179387
Average difference: 179387 / 208 (#pipes) ≈ 862.438
Variance: 797809
```

After (Continental)

```
Max difference: 5130
Sum of differences: 32877
Average difference: 32877 / 208 (#pipes) ≈ 158.062
Variance: 39775
```

Balance Graph

*PS_45	*PS_51	5071/9000	3929
PS_46	C_18	55/55	0
PS_47	C_18	55/55	0
PS_48	C_18	50/55	5
PS_49	C_18	40/55	15
*PS_49	*PS_58	5550/8000	2450
*PS_49	*PS_59	5500/9000	3500
*PS_50	*PS_49	11030/14000	2970
*PS_50	*PS_57	20/500	480
PS_51	C_16	35/35	0
*PS_51	*PS_45	106/9000	8894
*PS_51	*PS_53	5130/9000	3870

We took in notice that with large data set, after running Edmonds-Karp algorithm some bidirectional pipes would have flow in both directions. So in our balance algorithm we considered the bidirectional pipes as 2 pipes and incremented flow to both of them.

Balance Graph

```cpp
void BasicServiceMetrics::distributeExtraFlow(Vertex<Code> *vertex, double extraFlow) {
    vector<pair<Edge<Code>*, double>> edgeRatios;
    double totalRemCap = 0;

    for (auto& e : vertex->getAdj()) {
        double remCapacity = e->getWeight() - e->getFlow();
        if (e->getDest()->getInfo().getType() != CodeType::CITY) {
            totalRemCap += remCapacity;
            edgeRatios.emplace_back(e, e->getFlow() / e->getWeight());
        }
    }

    sort(edgeRatios.begin(), edgeRatios.end(), [](const pair<Edge<Code>*, d
        return a.second < b.second;
    });

    for (auto& [edge, ratio] : edgeRatios) {
        double remCapacity = edge->getWeight() - edge->getFlow();

        double flowToAdd = min(extraFlow, remCapacity);
        flowToAdd = min(flowToAdd, extraFlow);
        edge->setFlow(edge->getFlow() + flowToAdd);

        // Recursively distribute the remaining extra flow to downstream vertices
        if (flowToAdd > 0) {
            distributeExtraFlow(edge->getDest(), flowToAdd);
        }
    }
}
```

The most simple method we found to balance the flow was to pump the remaining water that wasn't delivered from the reservoirs to the whole graph.

Flow distribution was implemented in a dfs where each vertex would distribute the extra flow that received.

The distribution is based on a flow / weight ratio, where edges are sorted by ratio in an ascending order. Also we excluded the edges that would give flow to cities so we wouldn't change the max flow that could reach each city.

In this algorithm we could further improve the balance and reduce the variance along the graph by updating the ratio everytime we distributed flow, for example:

v1 -> v2; v1 -> v3

- v2 current flow: 2/10
- v3 current flow: 1/10
- v1 receives 3 units of flow, with the existing algorithm it would give 3 units of the flow to v3, resulting in v2 (2/10), v3 (4/10).

If we had track of the ratio when running the algorithm we would know that giving 2 units to v3 and 1 unit to v2 would result in a lower variance and better balance where we had 3/10 of flow in both v2 and v3.

Balance Graph

After pumping remaining water we could:

1. **Identify Overloaded Pipes**
   - Determine which pipes flow is close or even full to its capacity.
   - Keep track of them.
2. **Redistribute Flow**
   - Start with the pipes that have more flow / capacity ratio
   - Redistribute excess flow from those pipes to neighboring pipes while ensuring that the vertex receiving flow outputs the same amount of flow. Also need to ensure the flow that reaches each city remains the same.
   - Update the flow values for each pipe accordingly.
3. **Iteratively Improve**
   - Repeat the redistribution process iteratively.
   - Monitor the graph metrics after each iteration to track improvement.
   - Also monitor the paths that give to max flow, again, to make sure that the max flow to each city remains the same.

We didn't have enough time to find and implement an efficient way to keep track of everything while redistributing the flow so we couldn't implement this 2nd part of our heuristic. The first problem we encountered was that we had to distribute a small quantity of flow every time if we wanted to keep track of the ratio between neighboring pipes. At first we tried to distribute flow 1 by 1 but that wasn't a good option at all when dealing with the large data set. Then we took in notice that sometimes not balancing the first pipes would result in a better balance in pipes more behind and many other problems.

User Interface

```
[6] Remove Reservoir
```

[6]

```
Enter your choice: 6

Enter the Reservoir ID number (eg. R_9, enter 9): 3
New Total Max Flow: 1093
CITIES AFFECTED: "< [code] name (new-flow/demand), old-flow >"
    1. [C_4] Machico (121/137), 137
    2. [C_5] Santa Cruz (100/295), 295
    3. [C_6] Funchal (450/740), 664
    4. [C_7] Câmara de Lobos (100/225), 225

Press ENTER to continue...
```

Remove Reservoir

Proposta de abordagem para lidar com a remoção de um reservatório da rede de água sem ter que recorrer o algoritmo de Edmonds-Karp por completo:

1. **Armazenamento dos Caminhos e Valores de Fluxo:**

- Numa primeira execução do algoritmo de Edmonds-Karp, guardar os caminhos e os valores de fluxo que cada reservatório de água contribui para esses caminhos.

2. **Identificação dos Caminhos Afetados e decrementar o Fluxo:**

- Ao remover um reservatório da rede, é necessário identificar os caminhos que foram influenciados pelo fluxo que sai desse reservatório e decrementar o fluxo nas respectivas arestas.

3. **Verificação de Novos Augmenting Paths:**

- Ao decrementar o fluxo numa aresta afetada pela remoção do reservatório, essa aresta pode potencialmente tornar-se num augmenting path para outro reservatório de água. Portanto é necessário executar no final o ciclo para encontrar caminhos de augmenting entre a super source e a super sink.

Com esta abordagem, evitamos percorrer todos os caminhos para encontrar augmenting paths, pois a influência do reservatório removido é localizada, afetando apenas um subconjunto dos caminhos na rede. Isso reduz significativamente a quantidade de computação necessária para recalcular o fluxo máximo na rede.

User Interface

```
[7] Remove Pumping Station
```

[7]

```
Enter your choice: 7

Enter the Pumping Station ID number (eg. PS_9, enter 9): 5
New Total Max Flow: 1376
CITIES AFFECTED: "< [code] name (new-flow/demand), old-flow >"
    1. [C_3] Santana (20/46), 46
    2. [C_4] Machico (130/137), 137
    3. [C_5] Santa Cruz (100/295), 295
    4. [C_6] Funchal (625/740), 664

Press ENTER to continue...
```

User Interface

[8] Remove Pipes

! can be removed more than 1 pipe

[8]

```
Enter your choice: 8
Pipe from: ⚠[Enter 0 to finish the input]⚠
  1. [R] Reservoir
  2. [PS] Pumping Station
  3. [C] City


Your choice type: 2

Enter the ID number (eg. PS_9, enter 9): 9
Pipe To:
  1. [R] Reservoir
  2. [PS] Pumping Station
  3. [C] City


Your choice type: 2

Enter the ID number (eg. PS_9, enter 9): 10
```

```
Pipe from: ⚠[Enter 0 to finish the input]⚠
  1. [R] Reservoir
  2. [PS] Pumping Station
  3. [C] City


Your choice type: 2

Enter the ID number (eg. PS_9, enter 9): 4
Pipe To:
  1. [R] Reservoir
  2. [PS] Pumping Station
  3. [C] City


Your choice type: 2

Enter the ID number (eg. PS_9, enter 9): 5
Pipe from: ⚠[Enter 0 to finish the input]⚠
  1. [R] Reservoir
  2. [PS] Pumping Station
  3. [C] City


Your choice type: 0
```

User Interface

[8] Remove Pipes

[8]

```
Your choice type: 0
New Total Max Flow: 1551
CITIES AFFECTED: "< [code] name (new-flow/demand), old-flow >"
    1. [C_6] Funchal (572/740), 664

Press ENTER to continue...
```

User Interface

[9]

```
[9] Export All Cities Max Flow

Enter your choice: 9
Successful: Exported to ../output/allCitiesMaxFlow.txt

Press ENTER to continue...
```

allCitiesMaxFlow.txt

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

User Interface

[10] Export Critical Pipes per City

[10]

criticalPipesPerCity.txt

```
CRITICAL PIPES FOR EACH CITY

For each city, determine which pipelines, if ruptured would make it impossible
to deliver the desired amount of water to a given city.

(flow = city's flow if current pipe removed)

1. [C_1] Porto Moniz : (demand: 18)
    1. R_1 -> PS_1 (flow: 0)
    2. PS_1 -> C_1 (flow: 0)

2. [C_2] São Vicente : (demand: 34)
    1. PS_2 -> C_2 (flow: 30)
    2. PS_3 -> C_2 (flow: 30)

3. [C_3] Santana : (demand: 46)
    1. PS_5 -> C_3 (flow: 20)
    2. PS_6 -> C_3 (flow: 40)
```

User Interface

[11]

```
[11] Export Data Container
```

```
Enter your choice: 11
Successful: Exported to ../output/dataContainer_reservoir.txt
Successful: Exported to ../output/dataContainer_station.txt
Successful: Exported to ../output/dataContainer_city.txt
```

reservoir_dataContainer.txt

```
>> RESERVOIR INFORMATION
            Name: Ribeiro Frio
    Municipality: Santana
              ID: 4
            Code: R_4
    Max Delivery: 385

>> RESERVOIR INFORMATION
            Name: Curral das Freiras
    Municipality: Câmara de Lobos
              ID: 3
            Code: R_3
    Max Delivery: 630

>> RESERVOIR INFORMATION
            Name: Serra de Água
    Municipality: Ponta do Sol
              ID: 2
            Code: R_2
    Max Delivery: 300
```

station_dataContainer.txt

```
              ID: 12
            Code: PS_12

>> STATION INFORMATION
              ID: 11
            Code: PS_11

>> STATION INFORMATION
              ID: 10
            Code: PS_10

>> STATION INFORMATION
              ID: 9
            Code: PS_9

>> STATION INFORMATION
              ID: 8
            Code: PS_8

>> STATION INFORMATION
              ID: 7
```

city_dataContainer.txt

```
>> CITY INFORMATION
          City: Calheta
            ID: 10
          Code: C_10
        Demand: 76
    Population: 10915

>> CITY INFORMATION
          City: Ponta do Sol
            ID: 9
          Code: C_9
        Demand: 59
    Population: 8360
```

Highlight Functionalities

In function **`removePipes()`** we were able to let user choose as many pipes as it wanted to remove and get the cities affected.

Our balance algorithm was able to drastically reduce the variance with the large data set, although it increased the variance in the small data set. However it was still able to reduce the average difference of the flow/capacity along the graph.

Main Difficulties and Participation of Each Member

The main difficulty was probably to implement the balance algorithm that we had in mind. We only implemented a part of it.

We tried several implementations of recalculating the maximum flow when removing a reservoir without executing the Edmonds-Karp algorithm from scratch, but we always fell short of the solutions. The general flow decrement was correct, but the affected cities were missing by a small margin.

Effort:

- Bruno Huang - 50%
- Ricardo Yang - 50%