



ALGORITHM DESIGN - LEIC

PROJECT 1

G11_3

Water Supply Management System

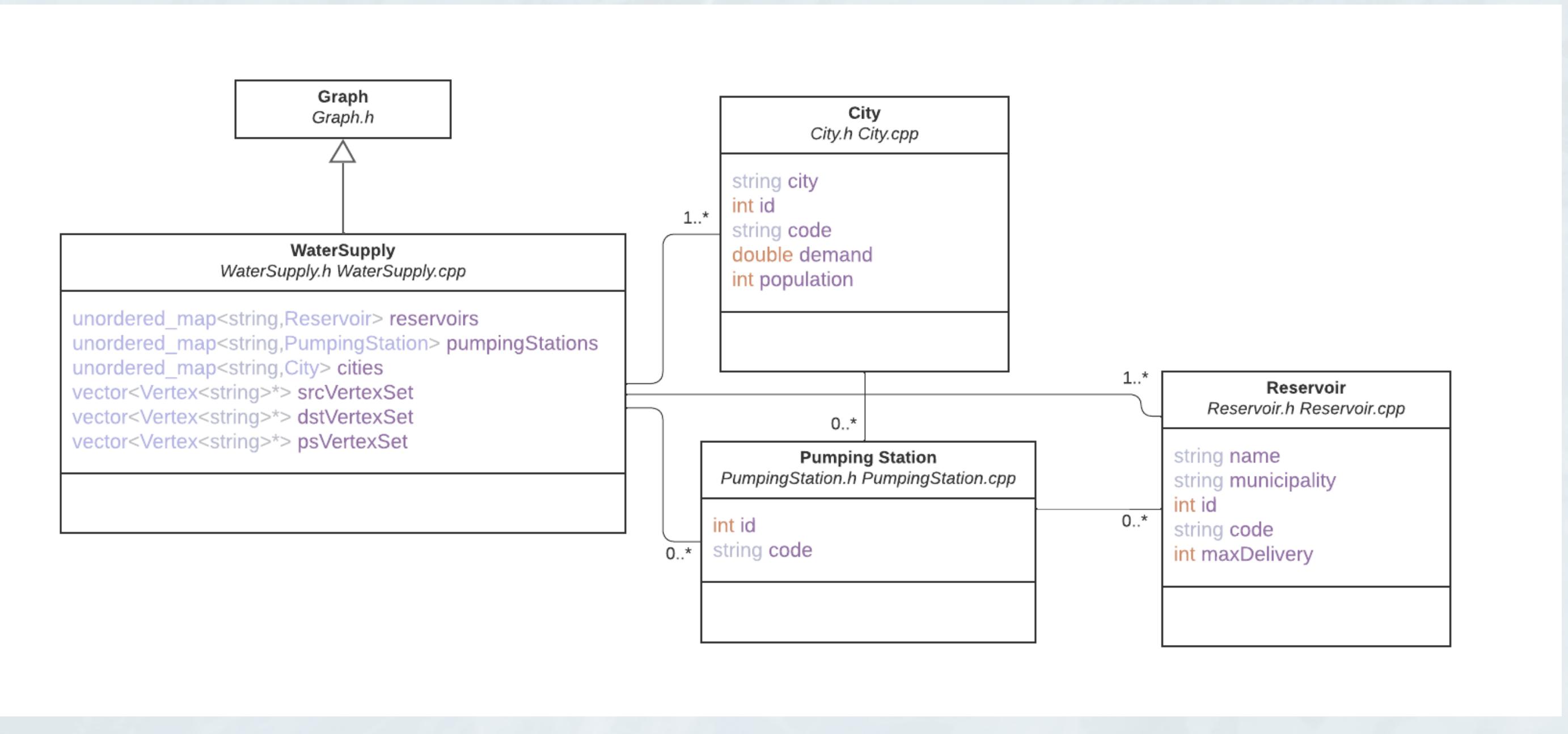
Work

Analysis tool that can support the management of a water supply network in Portugal to make informed decisions about how to best allocate its resources

Group Members

João Parada
Sofía Rodrigo
Tiago Loureiro

Class Diagram

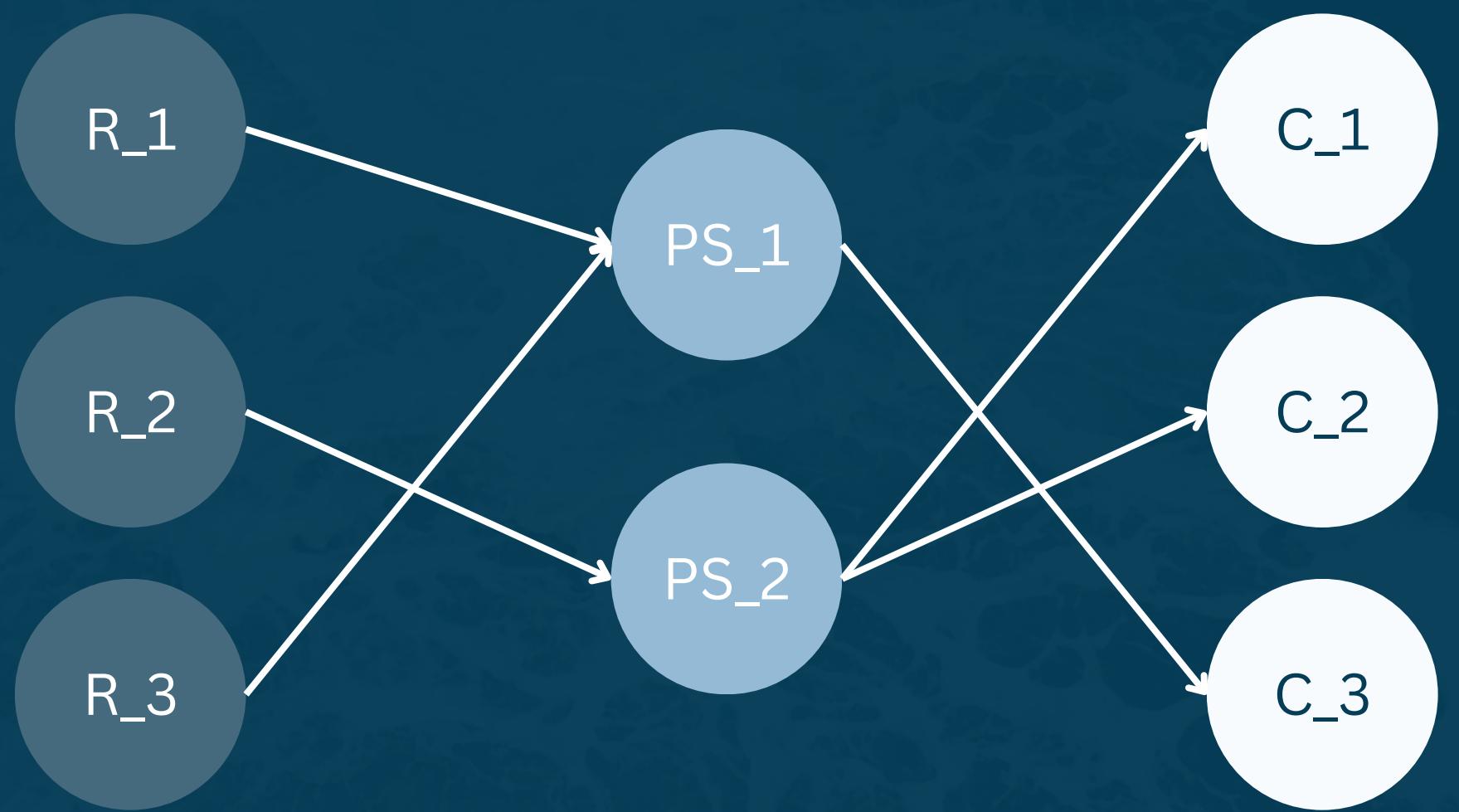


Reading Dataset

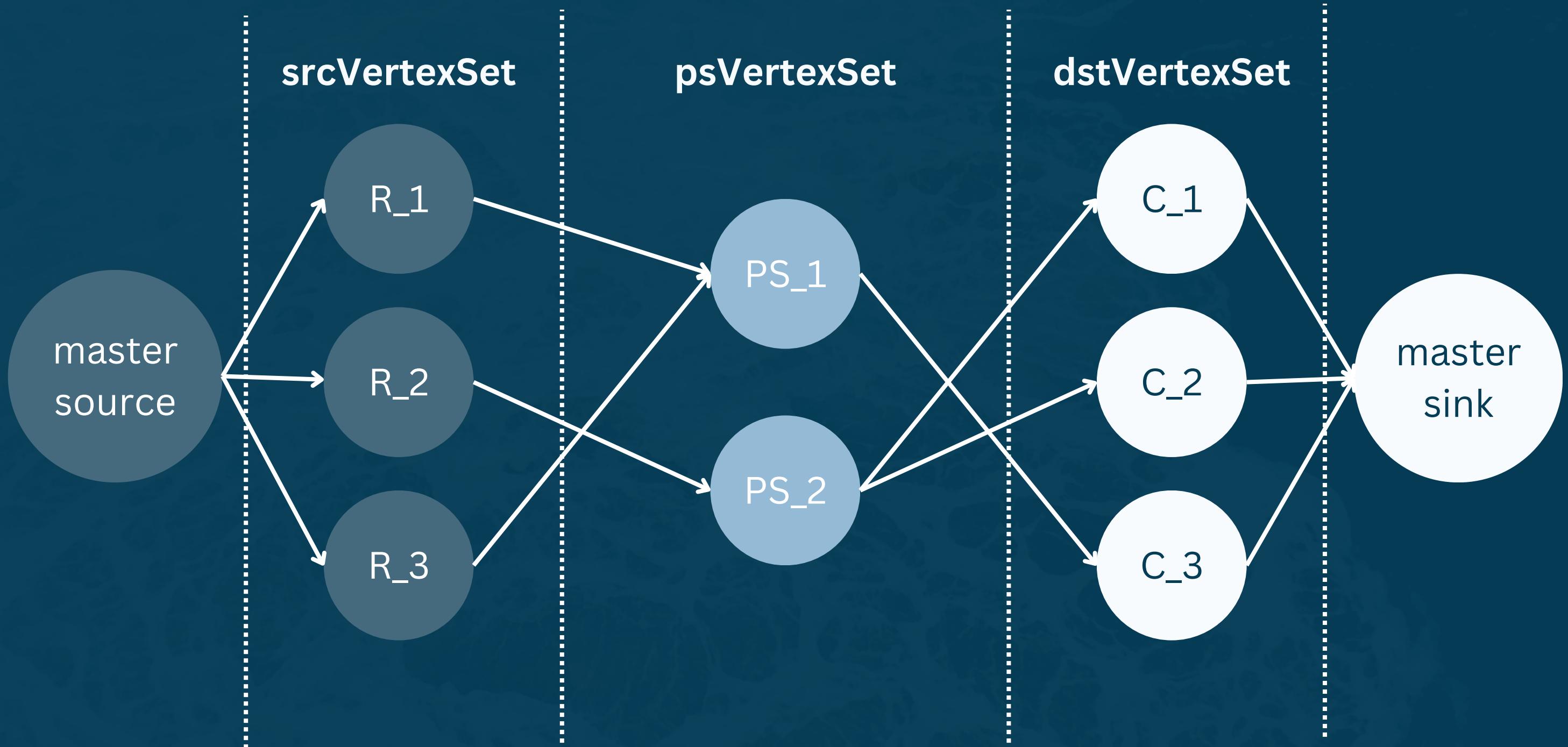
FileReader.cpp

Utilizes **ifstream** to read data from CSV files, constructing instances of Reservoir, PumpingStation, City classes and pipe connections, and then adds them to the WaterSupply network accordingly.

GRAPH



GRAPH



FUNCTIONALITIES I

MAX FLOW CITY

Calculates maximum flow for the specified city using Edmonds-Karp algorithm

$O(S*(V*E^2))$

$S = \#$ Sources (Reservoirs)

MAX FLOW GRAPH

Calculates the maximum flow for the entire graph using Edmonds-Karp algorithm. Optionally, it can run in an inverse order for special cases

$O(V*E^2)$

MAX FLOW GRAPH BALANCED

Balances the maximum flow in the graph by running the algorithm in both normal and inverse orders

$O(V*E^2)$

FUNCTIONALITIES II

REMOVE RESERVOIR AND LIST AFFECTED CITIES

Removes a reservoir from the graph and lists affected cities by recalculating maximum flow

$O(V^*E^2)$

CHECK USELESS PUMPING STATIONS

Checks and lists the impact of removing each pumping station from the graph

$O(P*(V^*E^2))$

P = #Pumping Stations

CHECK CRITICAL PIPES

Checks and lists the impact of removing each pipe from the graph

$O(V^*E^3)$

USER INTERFACE

```
8. Exit
1

The maximum amount of water that can reach the city C_1 is 20 m^3/sec. Maximum incoming capacity = 20.
Demand = 18.

The maximum amount of water that can reach the city C_2 is 50 m^3/sec. Maximum incoming capacity = 50.
Demand = 34.

The maximum amount of water that can reach the city C_3 is 60 m^3/sec. Maximum incoming capacity = 60.
Demand = 46.

The maximum amount of water that can reach the city C_4 is 160 m^3/sec. Maximum incoming capacity = 160.
Demand = 137.

The maximum amount of water that can reach the city C_5 is 300 m^3/sec. Maximum incoming capacity = 300.
Demand = 295.

The maximum amount of water that can reach the city C_6 is 950 m^3/sec. Maximum incoming capacity = 950.
Demand = 740.
```

Highlights

MAX FLOW GRAPH

Main difficulties

Balance Network
Algorithm

Class
Design/Structure

Team
Participation

João: 50
Sofia: 25
Tiago: 25

Copying a graph
object

EXTRA: Tests

✓ DA2324_PRJ1_G11_3	70 ms
✓ FileReader	4 ms
✓ addReservoirsSmallDataSet	0 ms
✓ addReservoirsLargeDataSet	0 ms
✓ addStationsLargeDataSet	1 ms
✓ addCitiesLargeDataSet	0 ms
✓ addPipesLargeDataSet	3 ms
✓ Functionality	66 ms
✓ maxFlowCity	4 ms
✓ maxFlowGraph	13 ms
✓ maxFlowGraphCleansFirst	19 ms
✓ balanceMaxFlowGraph	27 ms
✓ balanceMaxFlowGraphWithMadeiraSample	3 ms

EXTRA: Doxygen

localhost:63342/DA2324_PRJ1_G11_3/html/class_functionality.html

My Project

Main Page Related Pages Classes ▾ Files ▾

Functionality Class Reference

Static Public Member Functions

```
static string maxFlowCity (WaterSupply &graph, string &cityCode)
static vector< string > maxFlowGraph (WaterSupply &graph)
static vector< string > maxFlowGraph (WaterSupply &graph, bool inverseOrder)
static void balanceMaxFlowGraph (WaterSupply &graph)
static void removeReservoirAndListAffectedCities (WaterSupply &graph, const string &reservoirCode)
static void checkUselessPumpingStations (WaterSupply &graph)
static void checkCriticalPipes (WaterSupply &graph)
```

Member Function Documentation

♦ **balanceMaxFlowGraph()**

```
void Functionality::balanceMaxFlowGraph ( WaterSupply & graph )
```

This function generates two alternatives and then attempts to reconcile their differences, trying to pick the best alternative from each "splitting node" and rejecting the worst
Time Complexity = $O(V^2E)$ We need to run EdmondsKarp twice, then do several linear operations

♦ **checkCriticalPipes()**

```
void Functionality::checkCriticalPipes ( WaterSupply & graph )
```

Similarly to the previous two functions, this one calculates maxFlowGraph, then, for each edge of the graph, sets its capacity to 0, recalculates max flow and checks which cities got affected.
Time Complexity = $O(V^2E^3)$ (We run EdmondsKarp for each edge of the graph)

**Thank you
for your
attention**