Water Supply Management Analysis Tool

Grupo 5 2LEIC19

- *Afonso Castro* *up202208026*
- *Pedro Santos* *up202205900*
- *Rodrigo de Sousa* *up202205751*

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Graph

- Graphs are invaluable tools for illustrating interconnected systems.
- Here, we employ a graphical representation to delineate the intricate network of Water Reservoirs, Pumping Stations, and Cities within Portugal and Madeira, each delineated with its unique characteristics.
- We use the same Data Structure for this graph as the one that was provided in class, with no changes made. We created a class Node (mentioned later) to implement the changes needed.

Read Functions

- vector<Node> **readReservoirs** (string const &pathname);
- vector<Node> **readCities** (unordered_map<string,string> &cityDict, string const &pathname);
- vector<Node> **readStations** (string const &pathname);
- void **readPipes** (Graph<Node>& graph, string const &pathname);

```
vector<Node> ReadFunctions::readReservoirs(string const &pathname) {
    vector<Node> nodesReservoirs;

    ifstream file(pathname);
    if (!file.is_open()) {
        cerr << "Error (204): Wrong Path";
        return nodesReservoirs;
    }
    string fLine;
    getline(file, fLine);

    string line;
    while (getline(file, line)) {
        istringstream ss(line);
        string reservoir, municipality, code, id, maximumDelivery;


        if (getline(ss, reservoir, ',') && getline(ss, municipality, ',')
                    && getline(ss, id, ',') && getline(ss, code, ',') && getline(ss, maximumDelivery, ',')) {
            nodesReservoirs.push_back(Node('s', reservoir, stoi(id), code, municipality, stoi(maximumDelivery), 0));
        } else {
            cerr << "Error (302): Reservoir Loading Error";
        }
    }

    file.close();
    return nodesReservoirs;
}
  vector<Node> ReadFunctions::readCities(unordered_map<string, string> &cityDict, string const &pathname) {
      vector<Node> nodesCities;

      ifstream file(pathname);
      if (!file.is_open()) {
          cerr << "Error (205): Wrong Path";
          return nodesCities;
      }
      string fLine;
      getline(file, fLine);

      string line;

      while (getline(file, line)) {
          istringstream ss(line);
          string city, code, id, population, demand;

          if (getline(ss, city, ',') && getline(ss, id, ',') && getline(ss, code, ',')
                      && getline(ss, demand, ',') && getline(ss, population, ',')) {
              nodesCities.push_back(Node( 't', Node::removeCarriageReturn(city), stoi(id), code,"", stoi(population), stod(demand)));
              cityDict[Node::removeCarriageReturn(city)] = Node::removeCarriageReturn(code);
          } else {
              cerr << "Error (301): City Loading Error";
          }
      }
      if (cityDict.empty()) {
          cerr << "Error (701): Map Empty!";
      }

      file.close();
      return nodesCities;
  }
```

Read Functions

- vector<Node> **readReservoirs** (string const &pathname);
- vector<Node> **readCities** (unordered_map<string,string> &cityDict, string const &pathname);
- vector<Node> **readStations** (string const &pathname);
- void **readPipes** (Graph<Node>& graph, string const &pathname);

```
vector<Node> ReadFunctions::readStations(string const &pathname) {
    vector<Node> nodesStations;

    ifstream file(pathname);
    if (!file.is_open()) {
        cerr << "Error (206): Wrong Path";
        return nodesStations;
    }
    string fLine;
    getline(file, fLine);

    string line;
    while (getline(file, line)) {
        istringstream ss(line);
        string id, code;
        if (getline(ss, id, ',') && getline(ss, code, ',')) {
            nodesStations.push_back(Node('u', "", stoi(id), code, "", 0, 0));
        } else {
            cerr << "Error (303): Station Loading Error";
        }
    }
    file.close();
    return nodesStations;
}
void ReadFunctions::readPipes(Graph<Node> &graph, std::string const &pathname) {
    ifstream file(pathname);
    if (!file.is_open()) {
        cerr << "Error (207): Wrong Path";
        return;
    }
    string fLine;
    getline(file, fLine);

    string line;
    while (getline(file, line)) {
        istringstream ss(line);
        string start, end, capacity, direction;
        if (getline(ss, start, ',') && getline(ss, end, ',') && getline(ss, capacity, ',') && getline(ss, direction, ',')) {
            Node a('d', "", 0, start, "", 0, 0);
            Node b('d', "", 0, end, "", 0, 0);

            auto vertexA = graph.findVertex(a);
            auto vertexB = graph.findVertex(b);

            if (vertexA && vertexB) {
                graph.addEdge(vertexA->getInfo(), vertexB->getInfo(), stoi(capacity));

                int d = stoi(direction);
                if (d == 0) {
                    graph.addEdge(vertexB->getInfo(), vertexA->getInfo(), stoi(capacity));
                }
            } else {
                cerr << "Error (501): Vertex not found in the graph" << endl;
            }
        } else {
            cerr << "Error (304): Pipe Loading Error";
        }
    }
    file.close();
}
```

Operation Functions

- Graph<Node> **maxFlow**(Graph<Node>& graph);
- double **maxFlowOfCity**(Graph<Node>& graph, Node const &a);
- double **maxFlowPerCity**(Graph<Node>& graph);
- vector<pair<Node, double>> **supplyAndDemand**(Graph<Node>& graph);
- pair<double,double> **averageAndMaxOfDifferenceOfCapAndFlow**(Graph<Node>& graph);
- double **variance**(Graph<Node>& graph);
- void **deactivation**(Graph<Node>& graph, Node a);
- void **criticalPipesOfCity**(Graph<Node>& graph, Node city);
- void **citiesOfCriticalPipe**(Graph<Node>& graph, Node start, Node end);

Data Selection Menu

```
------------------------------------------------------------------------
Welcome to the Water Supply Management Analysis Tool!
------------------------------------------------------------------------
Select your data folder:
1 - Madeira.
2 - Mainland Portugal.
3 - Custom.
4 - Exit.
------------------------------------------------------------------------
Please select the task you wish to perform by inputting its number:
```

Operations Menu

```
------------------------------------------------------------------------
Select your operation:
------------------------------------------------------------------------
1 - Maximum amount of reachable water of a given city.
2 - List of maximum amount of reachable water per city.
3 - Water reservoirs supply and demand status.
4 - Analytics of Capacity and Flow.
5 - Water reservoirs deactivation analysis.
6 - Pumping stations deactivation analysis.
7 - Affected cities by given pipe analysis.
8 - Critical pipes by given city analysis.
9 - Credits.
0 - Exit.
------------------------------------------------------------------------
Please select the task you wish to perform by inputting its number:
------------------------------------------------------------------------
```

Node

Reservoirs, Stations and Cities

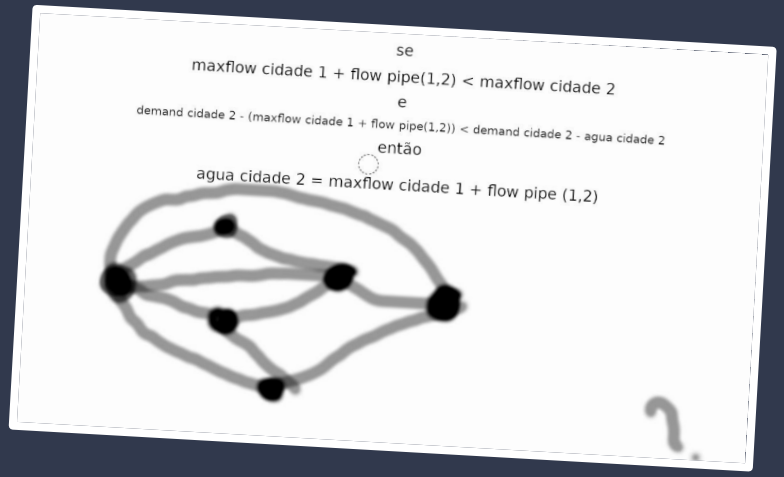
```
class Node {
    private:
        char type_; // s (source); t (sink); u (station)

        string name_; // reservoir name for s; city name for s; blank for u
        int id_; // common attribute
        string code_; // R_* for s; C_* for t; PS_* for u
        string municipality_; // municipality for s; blank for t and u
        int quantity_; // max delivery for s; population for t; blank for u
        double demand_; // demand for t; 0 for s and u
```

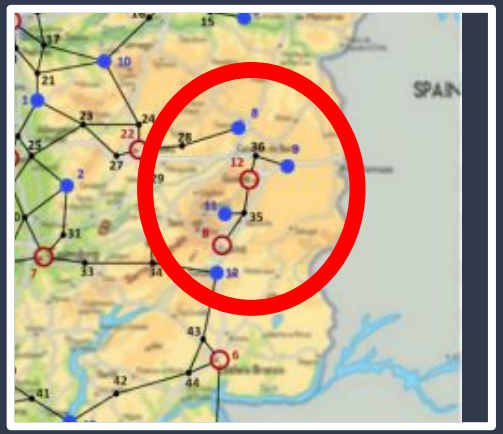
Time Complexity

The Time Complexity for **all Functions** is equivalent to the number of **Edges Squared**, **multiplied** by the number of **Vertices Squared.**

T2.3 E T3.1



Using an adapted version of the Johnsons Algorithm??



Identify SCC, use the algorithm on the specific SCC

Highlights and Difficulties

- Firstly, we had to make a choice on how to develop the main class, Node. We had the option of making many different classes, making a superclass or a single class for all the cities, reservoirs and pumping stations. We opted for the latter as we found it the best option for this scenario. All nodes are created equal, their attributes are what differentiates them from each other. Those differences are well explained in the code as comments.
- Secondly, we had to make a simple and intuitive menu, that allows the user to access all the info they may want. It has only 8 simple options, and the credits. All of the points from the worksheet are represented as options (except the balancing issue, mentioned later), to allow for a simple correction of our work. We had to make a choice of where to develop the functions for each menu option and we chose to call each function through wrappers.
- Lastly, even though we didn't manage to do the balancing of pipes, we have a calculator of the average, variance and max difference of flow and capacity, and we separated the critical pipes operation in two (Critical Pipes of City, and Affected Cities by a given Pipe) for a more organized information exposition.

FIM

Equal participation from all members of the group.