Sistema de Gestão de Água

Filipe Correia
Gabriela Silva
Gonçalo Nunes
Grupo G12_2

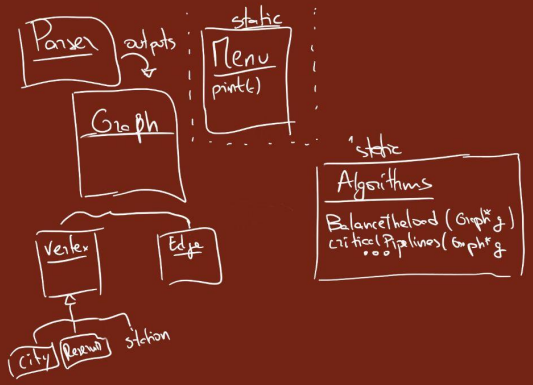
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

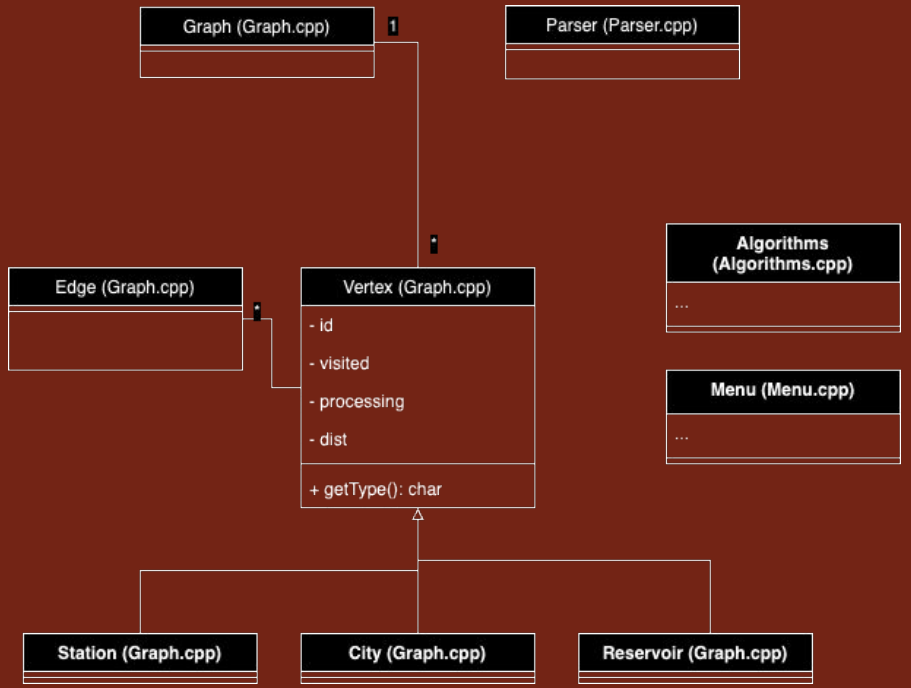
O que implementámos?

- Parser (com possibilidade de trabalhar com os dados de PT ou Madeira)
- Menu Intuitivo
- Máxima Quantidade de água por cidade
- Verificação de Satisfação das necessidades
- Balanceamento da Carga pelas pipes
- Falha de Reservatório, de uma estação de “pumping” ou de uma pipeline
- Pipelines críticas
- Output Opcional para ficheiro

Finalmente, as funções mais relevantes, contêm informação sobre complexidade temporal.

Visão Geral da Estrutura





Inicialização:

class Parser

```
Graph* parse(const std::string& reservoirFilePath, const
std::string& stationFilePath, const std::string&
cityFilePath, const std::string& pipeFilePath);
```

Responsável por percorrer os ficheiros percorridos, separando os dados com auxílio das funções std, como std::getline, que permite definir o carácter de terminação, facilitando a obtenção de segmentos separados por vírgulas, std::stoi, std::stod, std::stol, e as classes std::ifstream e std::istringstream.

class Graph

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

class Graph

```
class Graph{
protected:
    std::vector<Vertex*>
vertexSet;
}
```

```
class Edge{
protected:
    Vertex * dest;
    double capacity=0;
    bool selected = false;
    Vertex *orig;
    Edge *reverse = nullptr;
    double flow = 0;
}
```

```
class Vertex{
protected:
    std::vector<Edge *> adj;
    int id;
    bool visited = false;
    bool processing = false;
    double dist = 0;
    Edge *path = nullptr;
    std::vector<Edge *>
incoming;
    std::string code;
}
```

```
class Station : public Vertex{
public:
    char getType() override;
}
```

```
class Reservoir : public Vertex{
private:
    std::string name;
    std::string municipality;
    int delivery;
    int actualDelivery = 0;
}
```

```
class City : public Vertex{
private:
    std::string name;
    int id;
    int demand;
    int population;
    double totalWaterIn;
}
```


U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Algoritmos

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

$O(V E^2)$

Max-flow

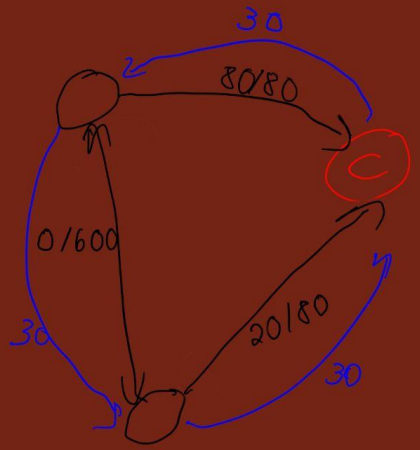
Começamos por adicionar uma SuperSource e um SuperSink ao grafo.

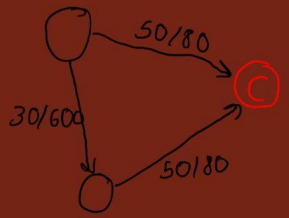
Algoritmo de Edmonds-Karp simples entre a SuperSource e o SuperSink.

Worst case - O((V + E)²)

Balance The Load - 1ª versão

1. Percorrer todos os vértices do grafo e calcular a média de espaço disponível nas incoming edges desse vértice.
2. Para todas as edges que tenham espaço disponível abaixo da média, tentamos redirecionar a água a mais para outro augmentation path, através de um Edmonds-Karp que começa e acaba naquele vértice.



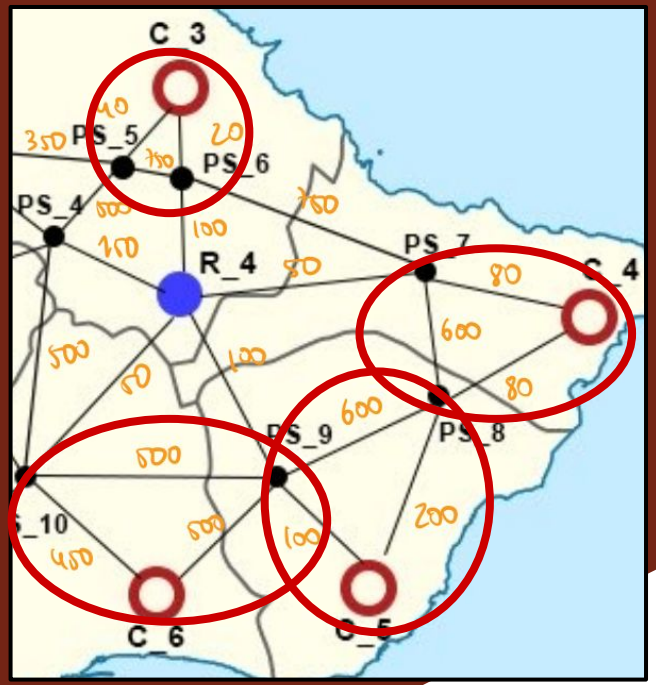


U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Worst case - O((V + E)²)

Balance The Load - 1ª versão - Nota

Este algoritmo funciona bem com grafos do estilo do mapa da Madeira, isto é, em que na maioria das cidades é possível chegar de um incoming edge para outro (como está na imagem de exemplo) mas quando na maioria das cidades as incoming edges vêm de sítios muito diferentes, como é o caso do mapa de Portugal, este algoritmo não vai ter o mesmo nível de resultados.



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Worst case - $O((V + E)^3 * E)$

Balance The Load - 2ª versão

1. Percorrer todos os vértices do grafo e calcular a média de espaço disponível nas incoming edges desse vértice.
2. Para cada edge que tenha espaço disponível abaixo da média, procuramos um path com flow que venha desde um reservatório até essa edge.
3. Depois para cada edge que tenha espaço disponível acima da média, procuramos um augmentation path que termine nesse edge, se existir adicionamos (no máximo o que falta para chegar à média) e retiramos o mesmo valor do edge com espaço abaixo da média.

Worst case - $O((V + E)^3 * E)$

Balance The Load - 2ª versão - Nota

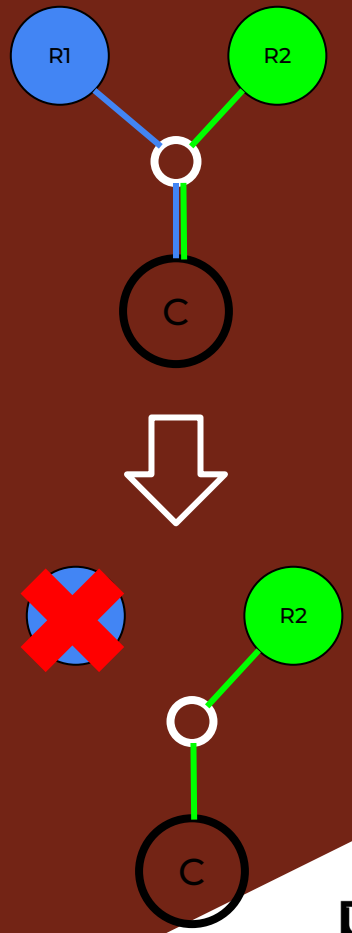
Este algoritmo, apesar de ser bem mais custoso, por estar a realizar mais pesquisas com o algoritmo base de Edmonds-Karp, vai ter melhores resultados no tipo de grafos em que a 1ª versão não conseguia melhorar muito, mas acaba por ter resultados semelhantes no mapa da Madeira.
Logo, a escolha da versão a usar depende do tipo de grafo, pensamos que para o mapa de Portugal esta 2ª versão seria uma melhor solução que a 1ª.

Reservoir Failure

A ideia: Gerar um algoritmo que calcule o fluxo que chega a cada cidade, ignorando o que provém do reservatório que “falha”.

O problema: Como optimizar tal a que, dado um grafo com os fluxos já definidos, se evite correr de novo o algoritmo Edmonds-Karp?

A solução: Tentar remover da rede apenas o fluxo originário do reservatório, e procurar augmenting paths que possam ter surgido.



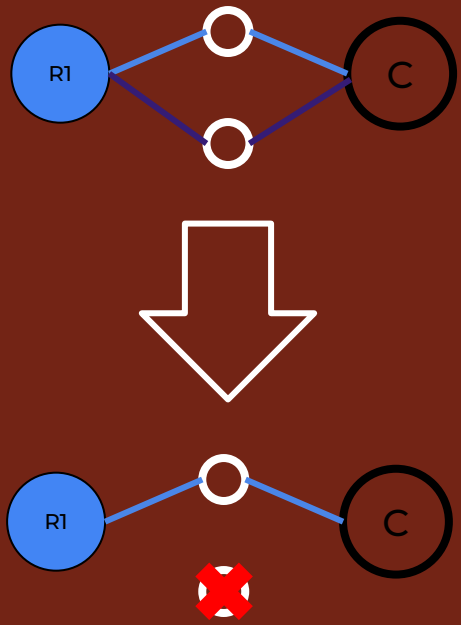
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Pumping Station Failure

A ideia: Gerar um algoritmo que calcule o fluxo que chega a cada cidade, ignorando o que atravessa a estação que "falha".

O problema: Como optimizar tal a que, dado um grafo com os fluxos já definidos, se evite correr de novo o algoritmo Edmonds-Karp?

A solução: Tentar remover da rede apenas o fluxo que atravessa a estação, e procurar augmenting paths que possam ter surgido.



O(V+ E)

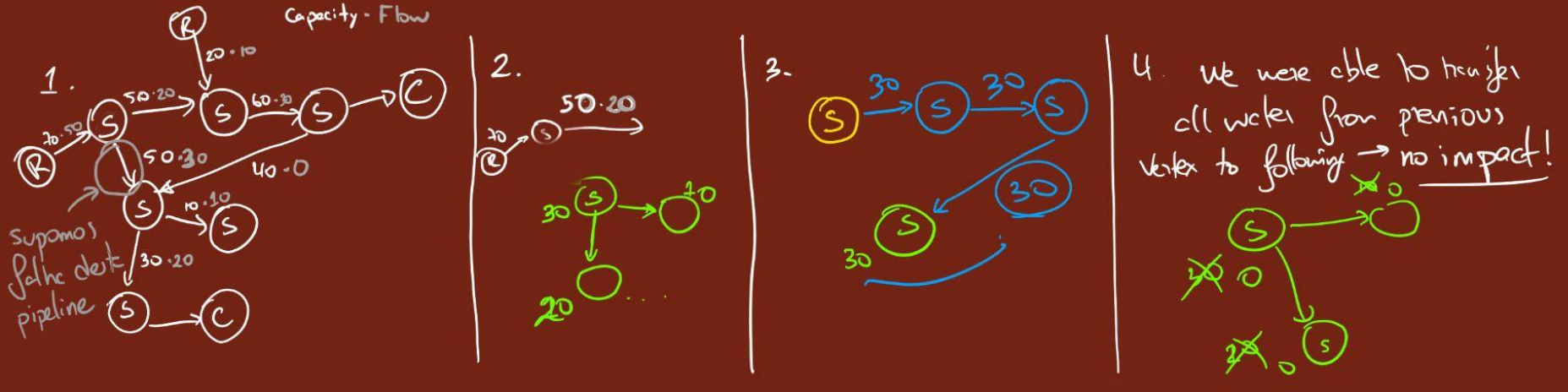
Pipeline Failure

Com uma pipeline em mente, a ideia é verificar todos os *Vertexes* que podem ser alcançados a partir do *Vertex* de destino (A) desta pipeline.

Associamos a cada Vertex encontrado a capacidade de água que ele consegue receber a partir do Vertex A. (Ver exemplo)

Posteriormente, iremos encontrar que caminhos a partir do(s) vértice(s) anterior(es) ainda conseguem transferir água para os vertexes posteriores. Depois de encontrado, atualizamos a capacidade de água que estes Vertexes (posteriores) conseguem receber e atualizamos os Vertexes que são alcançados por edges vindo do vertex A.

Pipeline Failure

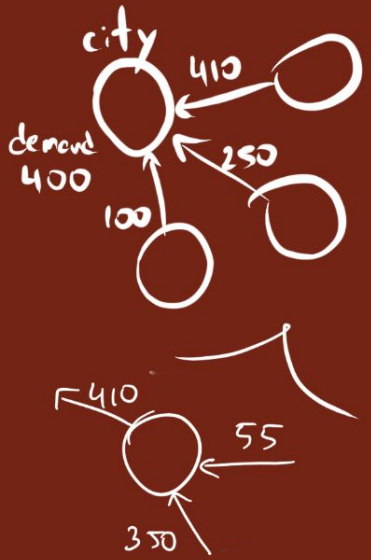


O(V * (V+ E))

Critical Pipelines (by city)

Começamos por iterar por cada Cidade Y e verificamos se algum dos incoming edges pode ser eliminado. Ao analisar uma edge X, caso a capacidade dos restantes edges (dada por sumothers) não seja suficiente, então esta edge X é crítica. Posteriormente, corremos a mesma lógica para o vertex X->getOrig(), com necessidade Y->getDemand() - sumothers.

Critical Pipelines (by city)



When checking 410:

- the sum of others: 350!
  ↓ < demand

so, 410 is critical

by 410 - 350 = 60

User Interface:

```
Choose model to work on. MADEIRA: 0; PT: 1  1
Welcome to the Water Management Program.
Options:
0 : Exit Program
1 : Repeat Instructions
2 : Max Amount of Water by city (Execute Edmonds Karp)
3 : Are needs met?
4 : Balance the load
5 : [RELIABILITY] One water reservoir is out
6 : [RELIABILITY] Which pumping stations are essential
7 : [RELIABILITY] Which pipelines are essential by city?
8 : [RELIABILITY] What cities are affected by mal-functioning of a pipeline.
9 : [INFO] Print Info about edges
Your option:
```

Documentação

Doxygen

```
//4 ------------------------------------------------------------
/*** @brief This function calculates the change in water reaching the cities w
 * Output: Cities that have lost water are printed, alongside the amount of wa
 * Complexity: O(V * E^2), because, even though it runs the optimized algorith
 * @param graph The graph on which to run this operation.
*/
static void shutDownReservoir(Graph* graph);

/** @brief This function calculates the change in water reaching the cities wh
* Output: If no errors occur, nothing will be printed \n
* Complexity: O(V * E^2)
* @param graph The graph on which to run this operation
* @param reservoirCode The reservoir which we are attempting to remove
*/
static std::vector<CityWaterLoss> CanShutDownReservoir(Graph* graph, const std
/** @brief This function calculates the change in water reaching the cities wh
* Output:  If no errors occur, nothing will be printed \n
* Complexity: In the worst case, it could be the same as running the Edmonds-K
* @param graph The graph on which to run this operation
* @param reservoirCode The reservoir which we are attempting to remove
*/
static std::vector<CityWaterLoss> CanShutDownReservoirOptimized(Graph* graph,

//5) ------------------------------------------------------------
/** @brief This function calculates the change in water reaching the cities wh
* Output: Cities that have lost water are printed, alongside the amount of wat
* Complexity: O(V * E^2), because, even though it runs the optimized algorithm
* @param graph The graph on which to run this operation.
*/
static void deletePumpingStation(Graph* g);
/** @brief This function calculates the change in water reaching the cities wh
```

Reflexões:

- No nosso trabalho, procuramos aprofundar cada questão apresentada, de modo a produzir algoritmos otimizados e reutilizáveis em problemas semelhantes. Exemplo: as funções de retirar água do grafo, e a variante do algoritmo de Edmonds-Karp que ignora um determinado vértice.
- O facto de utilizarmos uma classe para interação com utilizador (input/ output) (Menu.cpp), permite-nos ter maior flexibilidade e possibilitar funcionalidades de impressão diferentes.

Finalmente, este trabalho permitiu-nos compreender como os algoritmos estudados podem ser aplicados a situações reais.

Participação

Acreditamos que todos os membros trabalharam de uma forma equilibrada.

Sistema de Gestão de Água

Filipe Correia
Gabriela Silva
Gonçalo Nunes
Grupo G12_2

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO