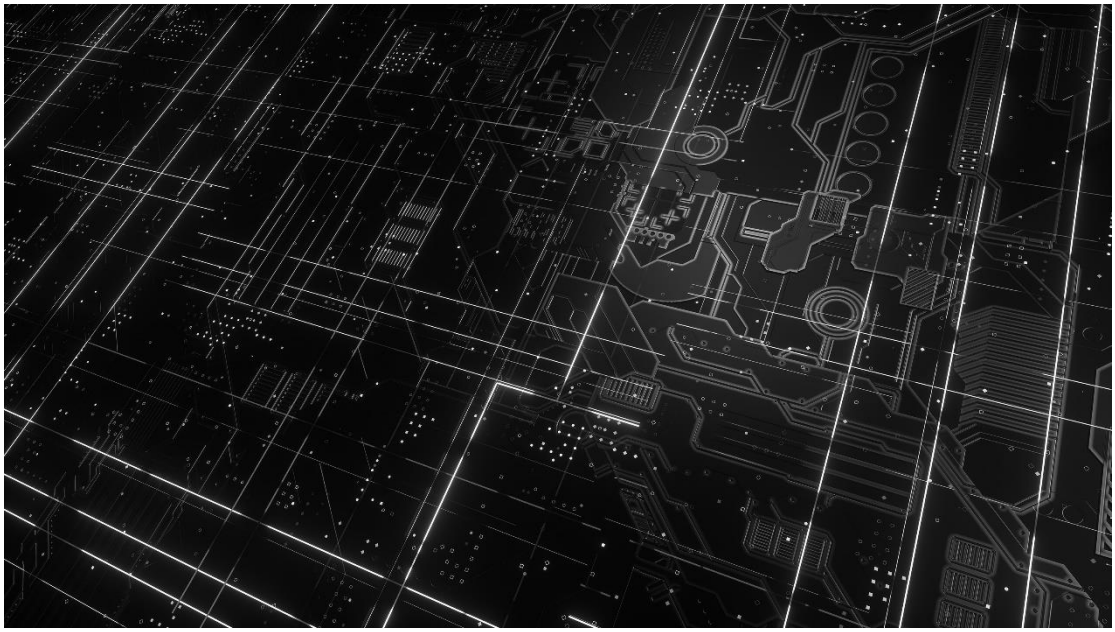
w



Relatório

EXPOSIÇÃO E EXPLICAÇÃO

Diogo Sá Dias – 1161605
Duarte Dias - 1190539
ESINF | 2021-22

Índice

Introdução

Breve explicação do Diagrama de Classes:

Inicialmente, foi pensado fazer vários controllers e cada um dos mesmos iria implementar uma US. Posteriormente optamos por fazer um só controller que designa-mos de MainController, no qual esse fará a função de todos os outros controllers anteriormente mencionados.

Basicamente, é importado através da classe Import, um ficheiro csv, no qual possui todas as informações dos navios. Esses navios que são guardados numa árvore ShipTree que possui os navios. Cada navio tem uma árvore de movimentos que guarda todos os atributo dinâmicos do navio, ou seja, os relacionados com a posição do navio.

Complexidade das US's: varia desde logn até n ao quadrado

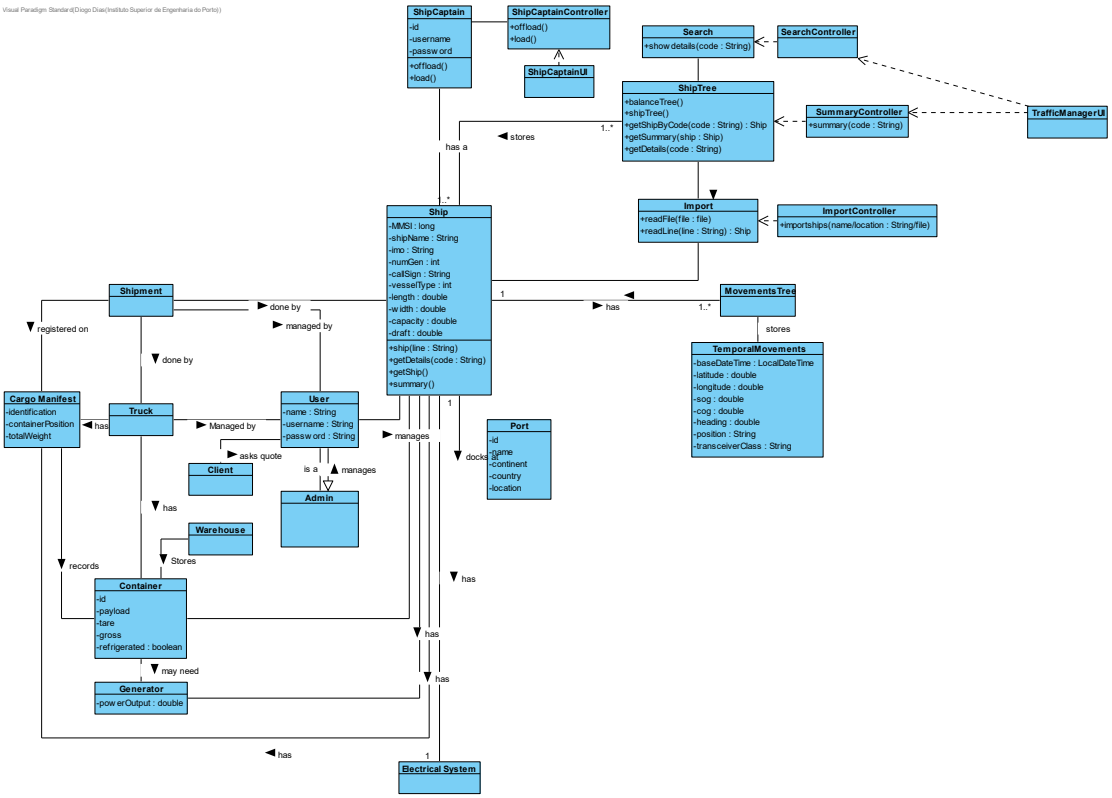
Sprint 1

DIAGRAMA DE CLASSES

ANÁLISE DE COMPLEXIDADE

Sprint 2

DIAGRAMA DE CLASSES



ANÁLISE DE COMPLEXIDADE

[US201] As a Port manager, I which to import ports from a text file and create a 2D-tree with port locations.

De acordo com o pedido no enunciado, o que se pretende nesta funcionalidade é receber informação de um ficheiro e inserir a mesma numa estrutura adequada. Para isso teremos um processo que irá percorrer todas as linhas do documento indicado. Depois iremos a cada uma das linhas construir o objeto Porto e de seguida inserimos na árvore 2D-Tree.

Deste modo, ler a linha e construir o Porto tem complexidade 1 e inserir na 2d-Tree tem complexidade $log\ n$. Como tal a criação desta árvore irá ter no seu pior cenário $n * \log n$ de complexidade temporal.

Diogo Sá Dias - 1161605

[US202] As a Traffic manager, I which to find the closest port of a ship given its CallSign, on a certain DateTime.

De acordo com o enunciado pretende-se encontrar o porto mais próximo usando a 2D-Tree anteriormente criada. Para isso implementei um método da 2D-Tree designado de findNearestNeighbour. Esse método vai calcular a distância entre o ponto dado e um nó da 2d-Tree, começando pela raiz e continuando, recursivamente por ambas os ramos da 2d-Tree. No final guarda a informação do nó mais próximo.

Assim esta funcionalidade vai ter, no melhor caso complexidade $log\ n$ e, caso a dimensão da árvore for alta ou a forma como os nós da árvore for especialmente organizada, a complexidade será n.

Duarte Dias - 1190539

Sprint 3

DIAGRAMA DE CLASSES

ANÁLISE DE COMPLEXIDADE

[US301] As a Traffic manager, I which to import data from countries, ports, borders and sea distances from the database to build a freight network.

O Enunciado pedia para através de uma busca a Base de Dados recolhermos a informação necessária para construir uma Grafo que fosse representativo do modelo de transporte que temos em mãos. Dessa forma decidimos dividir a tarefa de criação e construção da tarefa de recolha dos dados. Inicialmente, e devido a abordagem pretendida neste semestre – TDD – escolhemos começar pela parte da construção do Grafo.

Nesta fase construção e depois de analisarmos a US, onde é pedido que Portos e Países ou Cidades estejam ligados de forma indistinta decidimos criar o grafo e inserir de imediato todos os vértices através de uma lista de Portos e de uma lista de Cidades. Este método tem de **complexidade** $V * V^2 = V^3$ pois **insere** numa estrutura de matriz de Adjacências cada um dos Vértices.

De seguida e com todos os vértices inseridos decidimos ir inserindo os vértices pela ordem que consideramos mais apropriada: Primeiro as fronteiras entre capitais, de seguida a ligação entre porto mais próximo e Capital, fazendo em penúltimo lugar a ligação entre os Portos de cada país e finalizando na ligação que cada porto tem com n Portos de outro país.

Dessa forma primeiramente, verificamos se existem cidades no grafo e de seguida verificamos se o país se encontra na lista de fronteiras. Se verificarmos que essa fronteira existe, inserimos esta aresta. Para isso precisamos de receber uma lista de fronteiras do utilizador tal como as listas que recebemos antes. Esta formato, na verdade, por conveniência do grupo foi o de uma TreeMap que tem no seu keySet os países e no seu valor uma lista de países com quem faz fronteira. Sendo assim a **complexidade deste algoritmo** será da dimensão do keyset do TreeMap vezes a dimensão do tamanho máximo da lista por 1 – valor da inserção de uma aresta na Matriz de Adjacências. Será assim no pior caso de $k * l$.

Para a criação de ligação entre capital e o Porto mais próximo recorremos ao calculo da distancia mais curta entre a cidade e todos os Portos do País até obtermos o mínimo

valor de distância. Desta forma precisamos apenas inicialmente de obter uma lista de todos os portos do país. De seguida percorremos essa lista até encontrar os mínimos e adicionamos esse vértice. Sendo assim teremos de fazer isto para todos os vértices que são cidades, percorrendo todos os vértices que são Portos e pertencem aquele país. A **complexidade** deverá ser $\boldsymbol{V(cidades) * V(portos\ de\ um\ país)}$**.**

Antes de finalizar as últimas conexões fazemos a ligação de todos os portos de um país. Em que para cada porto presente no grafo, verificamos uma lista de portos do país. A não ser que os portos sejam o mesmo criamos a ligação. A **complexidade** é $\boldsymbol{V(portos) * V(portos\ de\ um\ país)}$**.**

Para finalizar adicionamos as n ligações a portos de outros países a cada Porto. Para isso utilizamos o método semelhante ao Floyd-Warshall. Primeiramente criamos uma lista de portos do grafo. De seguida com a distância entre portos e os portos criamos um grafo auxiliar com as distâncias entre portos que não tenham no grafo principal ligação. De seguida com a ajuda deste grafo auxiliar vamos saber quais os portos mais próximos e adicionar um a um aumentando de arestas até n dos dois vértices. Este algoritmo tem **complexidade:**

$$\begin{aligned} &V(portos) + V(portos) * V(portos) + V(portos) + V(portos) * E(porto) \\ &= V(portos) * \big(1 + V(portos) + 1 + E(porto)\big) \\ &= \boldsymbol{V(portos) * (2 + V(portos) + E(porto))} \end{aligned}$$

Sendo assim US tem complexidade de:

$$\begin{aligned} &V * V^2 + k * l + V(cidades) * V(portos\ de\ um\ país) + V(portos) \\ &* V(portos\ de\ um\ país) + V(portos) * \big(2 + V(portos) + E(portos)\big) \\ &= V^3 + k * l + Vc * Pp + Vp * Pp + Vp * (2 + Vp + Ep) \\ &= \boldsymbol{V^3 + Vp(Pp + 2 + Vp + Ep) + Vc * Pp + k * l} \end{aligned}$$

$V - Vértices\ do\ Grafo$
$Vp - Vértices\ que\ são\ portos$
$Pp - Vértices\ que\ são\ portos\ e\ que\ são\ de\ um\ País$
$Ep - Arestas\ de\ um\ porto$
$Vc - Vértices\ que\ são\ cidades$
$k - numero\ de\ país\ que\ fazem\ fronteira\ com\ outros\ países$
$l - numero\ de\ país\ que\ um\ país\ pode\ fazer\ fronteira$

Diogo Sá Dias - 1161605

[US302] As a Traffic manager I wish to colour the map using as few colours as possible.

[US303] As a Traffic manager I wish to know which places (cities or ports) are closest to all other places (closeness places).

O enunciado pedia para, através da utilização do grafo anteriormente construído, obter os locais que eram mais próximos a todos os outros locais, ordenados por continente. Assim, a ideia é construir 5 subgrafos, cada um deles relacionado a um continente, de modo a ser possível dividir todos os locais pelos seus respetivos continentes. Para isso a ideia é usar o algoritmo de Dijkstra que tem como complexidade (E) * log(V) no melhor dos casos. No entanto o algoritmo do shortestPath que encontra o caminho mais curto entre dois vértices usa o algoritmo de Dijkstra V vezes, o que significa que a complexidade passa a ser V * (E) * log(V).

Para saber a média entre os caminhos mais curtos, este algoritmo do shortestPath tem de ser chamado V vezes, o que faz com que a complexidade final seja V*V*E*log(V) no melhor dos casos e da maneira como se pretende fazer a US.

Duarte Dias – 1190539

Sprint 4

[US401] As a Traffic manager I wish to know which ports are more critical (have greater centrality) in this freight network.

De modo a encontrar os Portos com maior centralidade no grafo construído anteriormente optamos por proceder as seguintes operações. Primeiramente aplicamos o algoritmo de Dijktra para todos os vértices do grafo de modo a obtermos a lista de todos os caminhos mais curtos do grafo. De seguida contamos quantas vezes cada Porto aparece nesses caminhos, de modo a saber qual tem a maior centralidade. Percorrendo assim a lista de caminhos e verificando se o vértice é um Porto. Sendo um Porto aumentamos o valor para este em uma unidade. Por último percorremos N vezes – sendo N o valor de Portos Centrais que se pretende saber – a lista de modo a saber qual o maior valor – de caminhos percorridos – e o Porto que o tem.

Desta maneira percebemos que a complexidade do primeiro passo reside no algoritmo de Dijktra que é aplicado a todos os vértices e que resulta numa complexidade de V^3.

Depois temos de percorrer a lista de todos os caminhos curtos o que no pior caso pode tem uma complexidade de V^2.

No último passa vamos diminuindo o tamanho da lista de Portos e apenas temos de percorrer N vezes por isso a complexidade é $V(portos)! - (V(portos) - N)!$.

Sendo assim a função irá ter uma complexidade de:

$$\boldsymbol{V^3 + V^2 + V(portos)! - (V(portos) - N)! = V^3}$$

Diogo Dias - 1161605

[US402] As a Traffic manager I wish to know the shortest path between two locals (city and/or port).

De forma a encontrar o caminho mais curto entre dois locais é preciso, primeiramente, saber o tipo de caminho que se pretende, visto que nesta US havia três tipos de caminhos de acordo com o tipo de parâmetros que se usava. Posteriormente de serem efetuadas as necessárias verificações, através do algoritmo shortestPath, que basicamente retorna a mínima distância possível entre dois locais. É de realçar que este método usa o algoritmo de Dijkstra. Assim, para calcular a complexidade esta será a soma da complexidade do de Dijkstra (V*E+V = V(E+1) =V*E), o shortestPath usa um ciclo for V vezes, fora da invocação do algoritmo de Dijkstra, daí a complexidade total para já ser V*E+V.

No entanto o shortestPath também usa outro método chamado getPath, sendo este recursivo e, no pior dos casos pode ser chamado recursivamente V vezes (caso só haja um caminho entre o primeiro e o último vertex da matriz);

Assim a complexidade total no pior dos cenários é V*E+V+V = V(E+1+1) = V*E

Duarte Dias – 1190539

[US403] As a Traffic manager I wish to know the most efficient circuit that starts from a source location and visits the greatest number of other locations once, returning to the starting location and with the shortest total distance

De modo a encontrar o circuito mais eficiente que passe pelo maior número de localizações com a menor distância, teremos de fazer os seguintes passos. Para cada Vértice do Grafo, mas verifica qual o maior caminho possível que este pode obter. Depois de obtermos este caminho, vamos compará-lo com o caminho obtido anteriormente para verificar se é maior – primeiramente no que toca a localizações e depois, sendo iguais, ao nível da distância.

Sendo assim o primeiro iremos iterar V vezes esta sequência de operações que se seguem. Para encontrar o circuito primeiro iniciamos no vértice indicado e retiramos todos os seus vértices adjacentes – V. De seguida verificamos se estes já foram visitados – V. Antes de começarmos a recursão, colocamos os vértices adjacentes pela ordem de menor distância - V^2. Para a função recursiva temos que assumir que o pior cenário será este passar por todos os vértices do grafo até encontrar um circuito, dessa forma a complexidade deverá ser de V.

Acrescentamos a esta iteração pelos vértices exemplo a comparação de circuitos que tem complexidade – V- no pior caso, pois terá de analisar aresta a aresta qual o circuito com menos comprimento.

Sendo assim a complexidade geral da US será de:

$$V * \Big(\big((V + V + V^2) * V\big) + V\Big) = V * V^3 = \boldsymbol{V^4}$$

Sendo assim no pior dos cenários, todos os vértices terem ligação e o melhor circuito ser o que liga todos os vértices entre si, a complexidade da US seria de $\boldsymbol{V^4}$.

Diogo Dias - 1161605