Relatório de ESINF – Sprint 4

Turma 2DO _ Grupo 151

1200605 _ André Ferreira

1180611 _ Diogo Silva

1180682 _ Francisco Ferreira

1190708 _ João Ferreira

1200625_ Sérgio Lopes

Data: 23/01/2022

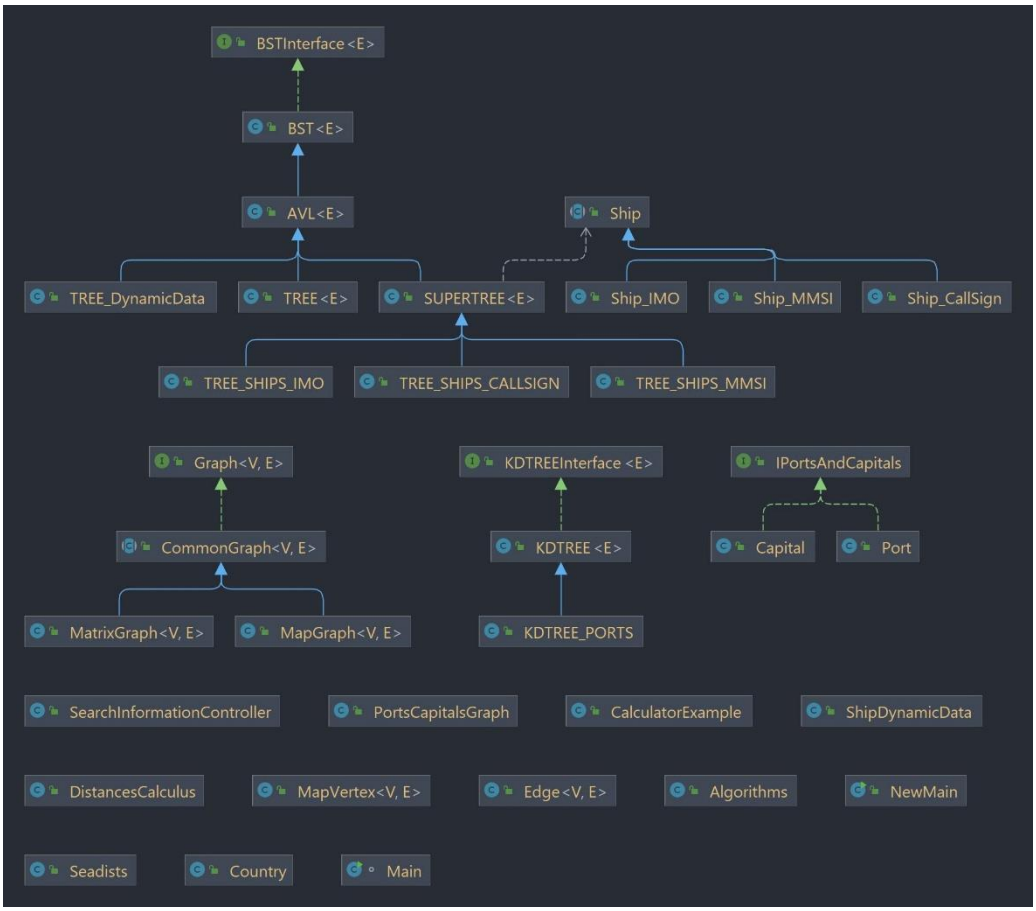
Índice

Parte I – Introdução e estrutura do trabalho

I.1 Introdução ao Sprint 4

Neste Sprint, a área de estudo manteve-se nos grafos e tudo o que ela abrange, sendo que nos permitiu que consolidássemos o nosso conhecimento nesta área. Além disso, permitiu uma melhor compreensão de algumas heurísticas de algoritmos para permitir resoluções o mais simples possíveis de problemas/perguntas complexas e também a utilização do shortestPath em todos as User Stories deste Sprint.

I.2 Diagrama de Classes



Parte II – Desenvolvimento do Sprint 4

II.1.a. Planeamento da 1º User Story (US401)

O objetivo deste user story passa por permitir ao utilizador ficar a saber quais os "***n***" portos que são mais críticos, ou seja, que tenham uma maior centralidade. O "***n***" é então inserido pelo utilizador e a centralidade de um porto corresponderá ao número de "***shortest paths"*** no qual este se insere. Assim, para a resolução deste US criamos um método "***fillPortsCentrality***" na classe "***PortsCapitalsGraph***" que tem como objetivo calcular o valor da centralidade para cada porto e retornar quer o porto quer a sua centralidade num mapa. Este método percorre todos os vértices do grafo e para cada um chama o método do "***shortestPaths***" que preenche duas listas, 1 de distâncias e outra com todos os respetivos shortest Paths para esse porto. De seguida, ainda para cada porto, percorrem-se as respetivas listas devolvidas pelo método do "***shortestPaths***" e para cada índice desses caminhos, verifica se não é uma capital, e adiciona ao mapa o porto e aumenta a centralidade (nº de ocorrências). Se o porto até então ainda não estiver no mapa, é adicionado com valor de centralidade a 0. O método "***getNPortsGreaterCentrality***" é o principal responsável pela resolução da US começando por chamar o método anteriormente referido preenchendo uma lista com os portos e a sua centralidade. De seguida organiza a lista através de um "***comparator***" por ordem decrescente de centralidade. Finalmente adiciona apenas os "n" primeiros portos desta lista já organizada ao mapa a retornar.

II.1.b. Análise de complexidade da 1º User Story (US401)

Relativamente á complexidade deste US, no método "***getNPortsGreaterCentrality***" este organiza a lista através de um "***comparator***" (O(n)), percorre a lista para retornar apenas os n portos (O(n)) e chama o método "***fillPortsCentrality***". Este método começa por percorrer todos os vértices do grafo (O(n)), invoca o método "***shortestPaths***" que terá uma complexidade de O(n^2) devido ao método "***shortestPathDijkstra***" e acaba percorrendo uma lista em função de outra para ir incrementando a centralidade (O(n^2)). Assim concluímos que a complexidade deste US será O(n^3).

- "***shortestPaths***" --> O(n^2+n+n) = O(n^2);
- "***fillPortsCentrality***" --> O(n (n^2 + n (n))) = O(n^3);
- "***getNPortsGreaterCentrality***" --> O(n^3+n+n) = O(n^3);

II.2.a. Planeamento da 2º User Story (US402)

Para esta UC tinham de ser implementadas as seguintes funcionalidades:

- Encontrar o caminho mais próximo terrestre entre dois pontos (pode começar ou acabar em portos ou capitais).
- Encontrar o caminho mais próximo marítimo entre dois pontos (só pode incluir portos).
- Encontrar o caminho mais próximo entre dois pontos (sem restrições).
- Encontrar o caminho mais próximo entre dois pontos, passando em N.

II.2.b. Análise de complexidade da 2º User Story (US402)

Para a funcionalidade de encontrar o caminho mais próximo terrestre, utilizamos o método "**shortestPathLand**".Neste método criámos uma cópia do grafo da classe PortsAndCapitalsGraph, com a diferença de não possuir **edges** entre vértices que são ambos portos. Para isso percorremos os vértices do grafo da classe e adicionamos ao novo grafo, tendo esta operação complexidade O(n(vertices)), de seguida adicionamos todas as **edges** que não ligam dois portos, esta operação tem complexidade de O(n(edges)). Por último fazemos o shortestPath entre os dois pontos passados por parâmetros, sendo o O(n^2).

-"shortestPathLand" --> O(n+n+n^2) = O(n^2).

Para a funcionalidade de encontrar o caminho mais próximo sem restrições, utilizamos o criámos "**shortestPathLandOrSeaPath**". Neste método apenas chamamos o shortestPath, uma vez que este método não tinha nenhuma restrição logo. Temos que a complexidade deste método é O(n^2).

-" shortestPathLandOrSeaPath" --> O(n^2) = O(n^2).

Para a funcionalidade de encontrar o caminho mais próximo marítimo, criámos o método "**shortestMaritimePath**". Neste método adicionamos todos os vértices do grafo da classe, que são portos, a um novo grafo criado dentro do método. De seguida adicionamos todas as edges entre exclusivamente portos. Ambas estas operações têm complexidade O(n).
Por último usamos o algoritmo **shortestPath** para encontrar o caminho. Esta operação tem complexidade O(n^2).
-" shortestMaritimePath" --> O(n+n+n^2) = O(n^2).

Para a funcionalidade de encontrar o caminho mais próximo passando por N locais, criámos o método "**shortestPathPassingOnN**". Neste método utilizamos o algoritmo de FloydWashall para criar um grafo cujas edges são shortestPaths entre pontos. Esta operação tem complexidade O(n^3).
De seguida, chamamos um método que irá comparar todas as iterações do caminho e comparar de modo a retornar o shortestPath de todas as iterações do percurso. Este método tem complexidade O(n^2).

-" shortestMaritimePath" --> O(n^3+n^2) = O(n^3).

II.3.a. Planeamento da 3º User Story (US403)

A 3ª user story deste Sprint tinha como objetivo obter o circuito mais eficiente começando de um local de partida e que visite o número máximo de outros locais apenas uma vez, sendo que o final desse circuito teria de voltar ao local de partida com a distância total mais curta possível.

Para que conseguíssemos solucionar este problema, teríamos de pensar numa heurística para que seja possível resolver este problema complexo com a solução/resposta o melhor possível. Tendo este propósito em mente, pesquisámos por heurísticas para conseguir resolver circuitos como os pedidos na User Story, sendo que nenhuma das encontradas foram exatamente de encontro ao que era necessário, usamos um algoritmo usado para encontrar ciclos num grafo chamado Hamiltonian circuit e modificamos o algoritmo de forma de forma a ir de encontro ao que era pedido. O que fizemos foi partir de um local inicial e avançar sempre para o local mais próximo ainda não visitado, quando não fosse possível avançar mais teria de retroceder até um local onde consiga fechar este circuito, voltando assim ao local de partida. De notar que este pensamento é uma heurística e como tal não tem soluções perfeitas, sendo que apenas é uma ótima e possível.

II.3.b. Análise de complexidade da 3º User Story (US403)

O método **isSafe()** tem uma complexidade de n, porque percorre um for loop, o método **HamCycleUtil()** tem uma complexidade de n^3 porque tem um for loop com complexidade n, que chama o metodo **isSafe()** com uma complexidade de n e faz chamadas recursivas para ele próprio, então a complexidade é de n^3, o método **hamCycle()** tem uma complexidade de n^4 porque tem um while loop que chama o método **HamCycleUtil()** com complexidadde de n^3 , então a complexidade é de n^4 (n*n^3), a **printSolution()** tem uma complexidade de n porque itera sobre o path retornado e imprime o resultado.

A complexidade final é de O(n^4), pois:
total time complexity = n + n^3 + n^4 + n = O(n^4) e é determinístico.

Contribuição de cada membro do grupo

Tal como em todos os Sprints até agora, todo o trabalho (estrutura geral, user stories e testes) contou com a cooperação de toda a equipa para a resolução do mesmo, de modo que todos ajudaram para a resolução das três user stories, tanto como dos testes e da análise da complexidade.