Relatório de ESINF – Sprint 3

Turma 2DO _ Grupo 151

1200605 _ André Ferreira

1180611 _ Diogo Silva

1180682 _ Francisco Ferreira

1190708 _ João Ferreira

1200625_ Sérgio Lopes

Data: 03/01/2022

Índice

Parte I – Introdução e estrutura do trabalho 3
I.1 Introdução ao Sprint 3 3
I.2 Diagrama de Classes 4
Parte II – Desenvolvimento do Sprint 3 5
II.1.a. Planeamento da 1º User Story 5
II.1.b. Análise de complexidade da 1º User Story 5
II.2.a. Planeamento da 2º User Story 6
II.2.b. Análise de complexidade da 2º User Story 6
II.3.a. Planeamento da 3º User Story 7
II.3.b. Análise de complexidade da 3º User Story 7
Contribuição de cada membro do grupo 9

Parte I – Introdução e estrutura do trabalho

I.1 Introdução ao Sprint 3

No Sprint 3, a área de estudo do grupo foi a criação de **grafos** de modo a armazenar os dados de um ficheiro, ou mais que um, ou de uma base de dados. Um grafo é um par (V,E) onde:

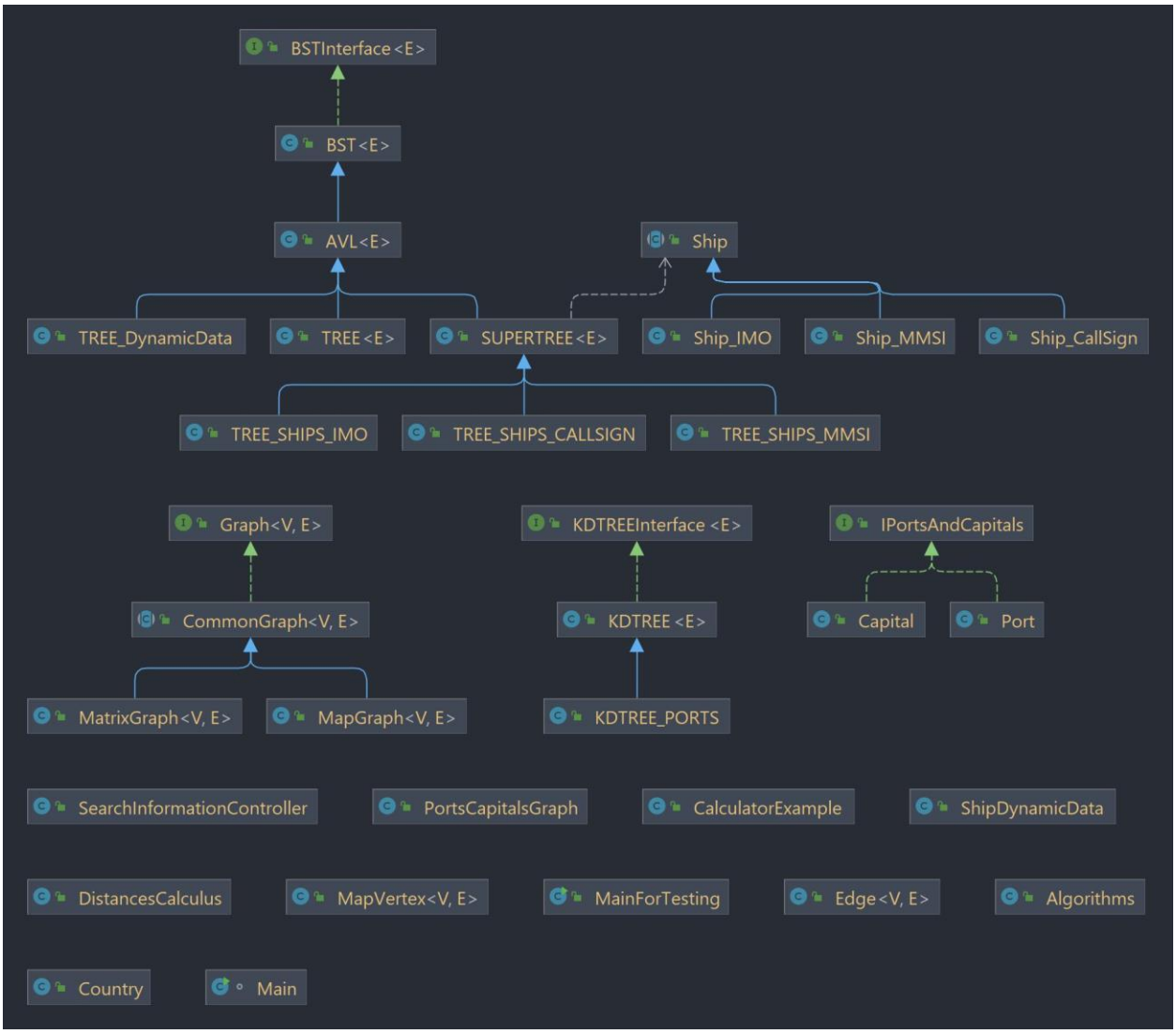
- V é uma coleção de nós (*nodes*), que são os chamados Vértices (*Vertices*);
- E é uma coleção de arestas (*edges*), que são chamados de Arestas (*Edges*).

Além disso, os grafos representam conexões, onde cada nó pode ter mais que um antecessor, sendo que também pode ter mais que um sucessor. Pode também ter mais que um caminho ou até nenhum, de um nó para outro, sendo que adicionalmente tem também ciclos e loops.

Os *Vertices* (nodes) representam objetos, estados, posições ou até place holders. Cada vértice é **único**, sendo que é impossível haver dois vértices a representar o mesmo objeto ou estado.

Os *Edges* podem ser direcionados ou não direcionados, sendo que também podem ser *weighted (or labeled) or unweighted.*

I.2 Diagrama de Classes



Parte II – Desenvolvimento do Sprint 3

II.1.a. Planeamento da 1º User Story

A 1ª User Story pedia para importar a informação que continha os ficheiros fornecidos pelos professores (countries, ports, borders e seadists) para construir uma rede de transportes. Para isso e de modo a conseguir satisfazer a acceptance criteria pedida, a capital dos países deve ter uma ligação direta às capitais dos países que faz fronteira. Os portos além de se conectar a todos os outros portos do mesmo país, o porto mais próximo à capital tem uma conexão à capital.

Além disso, todos os portos de cada país conectam aos n portos mais próximos dos outros países. Era necessário garantir que o cálculo das distâncias entre capitais, portos e capitais era em quilómetros, usando as coordenadas dos mesmos. Por fim, o grafo tem de ser implementado usando a *adjacency matrix* a manipulação indistinta das capitais e portos.

Para isso, foi criado os métodos **insertCapitals(), insertEdgesBetweenCapitals(), insertPortsAsVertex(), insertEdgesBetweenPorts(n),insertEdgesBetweenPortsSameCountry(), insertEdgeBetweenCapitalAndClosestPortWithinCountry(),** que de modo a que todos estes métodos fossem carregados no grafo ao mesmo tempo, temos o método **createPortCapitalGraph(int n)** que engloba todos os outros métodos anteriormente mencionados.

II.1.b. Análise de complexidade da 1º User Story

Relativamente á análise de complexidade do exercício 1, este chama vários métodos que possuem a sua respetiva complexidade, que irão, portanto, influenciar a complexidade final do exercício (método ***createPortCapitalGraph***). O primeiro método ***insertCapitals*** é determinístico tendo uma complexidade de n^2 . O método invocado (***insertPortsAsVertex***) é determinístico e tem uma complexidade de n^3. O mesmo se verifica para o método invocado (**insertEdgesBetweenPorts**) com uma complexidade de n^3. O método (**insertEdgesBetweenCapitals**) é determinístico e tem uma complexidade de n^2. (**insertEdgesBetweenPortsSameCountry**) tem uma complexidade de n^3 e é determinístico. Por fim o método (**insertEdgeBetweenCapitalAndClosestPortWithinCountry**) é também determinístico e tem uma complexidade de nLogn.

Complexidade final: n^2 + n^3 + n^3 + n^2 + n^3 + nlogn = n^3

II.2.a. Planeamento da 2º User Story

Para a 2º user story deste 3º Sprint, era pedido que conseguíssemos colorir o mapa dos países que foram disponibilizados no ficheiro fornecido pelos professores (countries.csv) usando o mínimo de cores possíveis, dependendo do que fosse necessário para colorir. Para isto, teríamos de garantir que os países vizinhos (que partilham fronteiras ex. Portugal e Espanha) não tivessem a mesma cor.

Desta forma, uma cor deveria ser atribuída a todos os países, sem ter países vizinhos com a mesma cor, para isso foi necessário atribuir às capitais dos países do ficheiro que tinham sido anteriormente carregadas para o grafo ***portsCapitalsGraph*** na classe ***PortsCapitalsGraph***, o país a que pertence com o método *getCountryByCapital.* Tornou-se também necessário fazer os métodos getCapitalVertices e getAdjVerticesFromCapital, para obter todos os vértices como capitais e obter os vértices adjacentes a uma capital, respetivamente.

Depois disto, temos o método ***colourTheMap*** que vai percorrendo os vértices das capitais que tem armazenados com as cores dos países todas inicializadas a -1 e vai dando uma cor aos países (começando pelo 0) de modo a colorir todos os países. Se encontrar um país que tenha fronteira com um país que tenha já cor 0, ele irá atribuir 1 a esse país para assim garantir que os países vizinhos não partilhem a mesma cor, fazendo isso por todos os países que se encontram no grafo e garantindo assim também que apenas usamos as cores estritamente necessárias, indo, neste caso com este ficheiro, até à cor 4.

II.2.b. Análise de complexidade da 2º User Story

Depois de analisar a complexidade deste nosso método, concluímos que tem uma complexidade de O(n^6) no pior dos casos e no melhor dos casos uma complexidade de O(n^4), visto que o método getCapitalVertices tem complexidade, no pior dos casos, O(n^2) e no melhor O(n), o método getAdjVerticesFromCapital, no pior dos casos O(n^2) e no melhor O(n) e por último o método getCountryByCapital tem como complexidade n, sendo que é um determinístico.

Sendo que, no pior dos casos: n + n + n^2 + n^3(n^2 + n^3 + n^2+ n) = n^6

E no melhor: n + n + n^2(n + n^2 + n + n) = n^4

II.3.a. Planeamento da 3º User Story

O objetivo desta user story passa por obter os "n" locais (seja um porto ou uma capital) que estejam mais próximos de todos os outros locais relativamente a cada continente existente. Para tal, no *controller* criamos o método "***nClosestlocalsByContinent***" que recebe por parâmetro o "***n***" referente ao número de locais a devolver por continente, este que é inserido pelo utilizador. Este método para além de dar print ao que se pretende, também cria uma lista com todos os continentes existentes no grafo para ir chamando o método "***nClosestLocalsByContinent***" da classe "***PortsCapitalsGraph***" enviando-lhe o "**n**" inserido e um **continente** de cada vez, para que este possa retornar um mapa ordenado com a informação pretendida por continente (distância média a todos os locais e a sua respetiva informação). Este segundo método começa por criar uma lista de portos/capitais onde vai adicionando apenas aqueles que pertençam ao continente inserido. 1- De seguida para todos os locais dessa lista, vai comparando-os com todos os outros locais calculando a soma das distâncias, fazendo posteriormente a sua média e adicionando a um mapa auxiliar esta mesma distância média e a respetiva informação do local em análise. *2- Uma outra forma que inicialmente tínhamos, mas que deixamos apenas para demonstração era para todos os locais dessa lista, vai comparando-os com todos os outros locais para saber os caminhos entre eles. Entre cada par de vértices, através do método "**shortestPath**", adiciona a uma linked list o respetivo caminho, calculando também o seu peso, adicionando a um mapa auxiliar a respetiva distância média e o local em estudo, isto para todos os locais da lista inicial. Infelizmente como alguns países ou conjunto de países são ilhas, acabam por ter poucas ligações no grafo ficando com distâncias médias muito pequenas e aparecendo em primeiro como países mais próximos dos restantes.* Finalmente deste mapa auxiliar apenas se adiciona ao mapa a retornar os "***n***" locais pedidos, estes que serão os n primeiros do mapa auxiliar por terem as distâncias médias aos outros locais mais reduzidas.

II.3.b. Análise de complexidade da 3º User Story

A complexidade deste user story começa no seu controller (***nClosestLocalsByContinent***) que ao preencher a lista com todos os continentes tem uma complexidade de O(n). Ao invocar o método principal do exercício (***nClosestLocalsByContinent***) este é feito num ciclo "for" para cada continente que tem também complexidade O(n) e que possui um "for" para fazer a listagem (O(n)). Relativamente ao método ***nClosestLocalsByContinent*** este começa por preencher a lista de locais (O(n)), de seguida na comparação entre cada dois locais utiliza dois ciclos (O(n^2)) e finalmente adiciona os "n" pretendidos ao mapa a retornar através de um ciclo (O(n)). Assim conclui-se que na resolução deste user story se utiliza um

método determinístico com complexidade no pior caso de n + n (n*(n+ n^2 + n) + n) = n^2(n+ n^2 + n) + n^2 = n^3 + n^4 + n^3 = O(n^4).

*Na resolução da nossa outra solução deste user story é utilizado o método "**shortestPath" que** se auxilia no método **"shortestPathDijkstra"** que possui uma complexidade de n^2 no pior dos casos, assim o método de **shortestPath** terá também a mesma complexidade pois não possui nada que possa alterar este valor.*

*O método onde o **shortestPath é** invocado terá uma complexidade de O(n + n*n*(n^2) + n) + n). Resultando no valor de complexidade de O(n^4 + n^3), simplificando em apenas O(n^4). Este método é chamado dentro do controller, obtendo este user story uma complexidade de O(n + n*(n^4+n)), simplificando em O(n^5+n^2) = O(n^5).*

Contribuição de cada membro do grupo

Tal como o Sprint 1 e 2, 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.