Introdução

Se no primeiro relatório explicámos com detalhe cada uma das diferentes estruturas de dados implementas e apresentámos as estratégias que adotámos para responder o mais rapidamente possível às análises que eram pretendidas, neste relatório iremos focar a nossa apreciação num ponto de vista mais ao nível do *hardware* e tempos de execução, algo que certamente até os menos entendidos na área da informática serão capazes de compreender.

Deste modo, pretendemos apresentar e justificar os diversos tempos de resposta que o nosso programa detém, quer na transferência dos dados para a memória, quer na análise de cada uma das *queries*. Além disso, como a quantidade de recursos requeridos por um programa parece-nos ser um parâmetro de extrema relevância, decidimos por bem referir alguns dados relativos à memória.

Por fim, como naturalmente algumas *queries* levam significativamente mais tempo do que outras a serem executadas, procurámos justificar tais resultados a partir de uma análise assimptótica que levava em conta diversos parâmetros, pois desta forma julgamos estar a ser o mais rigorosos possível.

Testes de Desempenho

O tempo é um dos parâmetros mais relevantes no que à execução de programas diz respeito, posto isto, decidimos analisar o tempo de execução do nosso projeto não só no global, mas também em diferentes etapas da execução em si, entre elas a recolha dos dados e o tempo que cada *query* leva a ser analisada.

Deste modo, para realizar uma análise da forma mais correta possível, executámos o dito programa em computadores distintos, pois assim sempre podemos ter uma ideia de qual o *hardware* que melhor se adequa ao código gerado.

Tratamento estatístico

- Realização de cinco execuções em cada um dos computadores (ligados à corrente).
- Eliminação de *outliers* resultantes de execuções nas quais o sistema operativo estava livre/sobrelotado.
- Cada um dos tempos resulta de uma média ponderada sem a consideração dos *outliers*.

Ambientes de Execução

Computador A		
Processador	**Memória**	**Sistema Operativo**
Intel i5-10300H	**8GB - 2600MHz**	**Ubuntu**

Computador B		
Processador	**Memória**	**Sistema Operativo**
Intel i5-1135G7	**16GB - 2700MHz**	**Ubuntu**

Desempenho das Queries - Dataset Regular

Query	Computador A (s)	Computador B (s)
1 SaCruz110	0.000073	0.000072
1 ClarPacheco48	0.000053	0.000021
1 GabriJesus	0.000013	0.000005
1 000000004780	0.000041	0.000018
1 000000007141	0.000027	0.000018
1 000000003123	0.000028	0.000005
2 10	0.177266	0.207539
2 50	0.180220	0.205983
3 10	0.236767	0.264780
4 Braga	0.013727	0.011439
4 Porto	0.011629	0.010332
4 Lisboa	0.011562	0.010240
5 01/01/2020 01/01/2021	0.054139	0.053338
5 01/01/2021 01/01/2022	0.058550	0.058892
6 Braga 01/01/2020 01/01/2021	0.017424	0.014922
6 Porto 01/01/2021 01/02/2021	0.017406	0.014893
7 10 Braga	0.038273	0.037780
8 M 12	0.013602	0.015746
8 F 12	0.014444	0.016966
9 24/12/2021 25/12/2021	0.042344	0.038273
9 9 01/01/2012 01/01/2013	0.041159	0.036926

Desempenho das Queries - Dataset Estendido

Query	Computador A (s)	Computador B (s)
1 EmanSimões9	0.000079	0.000084
1 GuilAraújo529	0.000018	0.000031
1 MateuPereira	0.000018	0.000039
1 000000063182	0.000023	0.000025
1 000000030891	0.000022	0.000024
1 000000095244	0.000020	0.000027
2 10	2.024240	2.364776
2 50	2.027261	2.420070
3 10	2.708971	3.009026
4 Braga	0.234393	0.205141
4 Porto	0.234568	0.198575
4 Lisboa	0.234013	0.204607
5 01/01/2020 01/01/2021	0.781794	0.799246
5 01/01/2021 01/01/2022	1.005023	1.070379
6 Faro 01/01/2020 01/01/2021	0.153627	0.139516
6 Braga 01/01/2021 01/02/2021	0.157949	0.137627
7 10 Porto	0.588885	0.571663
8 M 12	0.176378	0.212583
8 F 12	0.184072	0.217654
9 24/12/2021 25/12/2021	0.454373	0.405002
9 9 01/01/2012 01/01/2013	0.420670	0.383568

Leitura e Validação dos Dataset

Dataset Regular (s)		
Utilizadores	Condutores	Viagens/Cidades
0.074244	0.008443	0.882962

Dataset Estendido (s)		
Utilizadores	Condutores	Viagens/Cidades
0.708699	0.083938	11.151530

Tal como a tabela evidencia, a recolha dos dados relativos aos utilizadores e condutores parece ser feita em tempo linear, ao passo que a recolha das viagens/cidades é um pouco mais complexa, sendo muito possivelmente polinomial, com um grau pouco superior a um, algo que se deve ao facto de os dados estarem a ser armazenados em diversas estruturas simultaneamente.

Gestão de Memória

Dataset Regular (input)		
Memória Alocada	Leak	Peak
345.2 MB	0 MB	226.5 MB

Dataset Estendido (input)		
Memória Alocada	Leak	Peak
3240.4 MB	0 MB	2126.2 MB

A partir destes dados percebemos que a quantidade de memória exigida pelo programa cresce de forma linear, algo que faz todo o sentido, uma vez que a quantidade de dados que devem ser guardados também segue essa tendência, além disso, notamos que o programa revela um determinado nível de segurança relativamente à gestão de memória, dado que não foram detetados quaisquer *memory leaks*.

Análise de complexidade

Para melhor compreendermos o porquê de o tempo requerido em algumas *queries* ser tão distinto do de outras, pensamos que a melhor forma de explicar tais valores seja recorrer a uma análise assimptótica, na qual verificamos a complexidade de cada umas das *queries* tendo em conta diversos parâmetros ao invés de fixarmos um valor em especifico, como por exemplo o número de viagens realizadas.

Query 1

Esta *query* possui uma complexidade de $N \in \theta(N)$, sendo que N representa o número de viagens realizadas por um utilizador/condutor, deste modo, poderíamos pensar que a análise de um condutor levaria mais tempo, já que normalmente este possui mais viagens, contudo como os utilizadores estão contidos numa tabela de *hash* (onde cada *bucket* é na verdade uma lista ligada), daí resulta que o acesso a estes é bastante custoso e moroso, visto que estão dispersos pela memória.

Assim sendo, e tal como verificamos pelas múltiplas execuções desta *query*, um condutor necessita de menos tempo que um utilizador para ser analisado.

Query 2

Nesta análise é pedida uma listagem dos N condutores com maior avaliação média, portanto, para resolver este problema, tivemos de identificar cada um dos condutores e analisar as suas respetivas viagens, pelo que se pode dizer que no fundo percorremos o *array* das viagens de cima a baixo.

Depois disso, foi ainda necessário ordenar a estrutura de dados resultante da recolha inicial, operação que foi feita a partir do famoso algoritmo de ordenação *quick sort*.

Posto isto, a complexidade desta *query* corresponde a $N + P + P \log P \in \theta(N)$, sendo que N corresponde ao número de viagens e P ao número de condutores, pelo que se percebe que o tempo de execução não depende do parâmetro *top n* passado com argumento.

Query 3

Esta *query* é muito semelhante à anterior, muda o facto de se pretender listar os N utilizadores com maior distância viajada, assim sendo, se na análise anterior foi necessário percorrer o *array* dos condutores por completo, neste caso teremos de percorrer a tabela de *hash* onde os utilizadores estão presentes, o que não é tão eficiente dado que esta estrutura não beneficia da mesma localidade espacial que os *arrays*.

Deste modo, a complexidade equivale a $N + S + S \log S \in \theta(N)$, onde S corresponde ao número de utilizadores, o que é bastante credível, uma vez que o tempo de execução é bastante próximo do da análise feita anteriormente.

Query 4

Neste problema é pedido para calcular o preço médio das viagens numa determinada cidade, como tal, a melhor forma de abordar esta questão é tirar partido da estrutura de dados das viagens e das cidades, de modo a obter todos os índices cujas viagens foram realizadas na cidade em questão.

Depois de obter todos os índices, basta percorrer essas mesmas posições e ir acumulado o preço de cada uma das viagens, de modo a fazer uma média ponderada no final. Assim, a complexidade desta *query* corresponde a $N/C \in \theta(N/C)$, sendo que C representa o número de cidades existentes, isto assumindo que as viagens estão igualmente distribuídas pelas cidades.

Query 5

Esta *query* é uma das poucas em que não fomos capazes que criar uma estrutura auxiliar de modo a obter uma execução mais rápida, como tal, fomos obrigados a percorrer o *array* das viagens sem que no fundo houvesse essa necessidade, visto que tivemos de analisar viagens que nem sequer respeitavam os parâmetros pedidos pela *query*.

Assim, a complexidade desta *query* equivale a $N \in \theta(N)$, o que não reflete lá muito bem os tempos obtidos, dado que esta análise beneficia bastante da localidade espacial.

Query 6

A exemplo do problema anterior, também neste caso não fomos capazes de realizar qualquer otimização referente às datas, portanto, apenas diminuímos o número de iterações realizadas graças ao parâmetro *cidade*, dado que a partir do *array* das cidades sabemos *a priori* quais os índices das viagens em que a cidade em questão ocorre.

Deste modo, como temos de analisar todas as viagens realizadas numa determinada cidade, concluímos que a complexidade desta *query* é idêntica à da *query* 4, ou seja $N/C \in \theta(N/C)$, o que faz sentido, umas vez que os resultados obtidos em ambas as análise são bastante idênticos.

Query 7

Mais uma vez, o parâmetro *cidade* é um dos pontos chave no que à execução de *queries* diz respeito, assim sendo, apenas necessitamos de analisar determinadas posições da estrutura de dados das viagens e ir adicionando os condutores numa árvore binária de procura de modo a calcular a avaliação média de todos os condutores.

Depois de calcular todos os valores necessários, basta copiar os elementos presentes na árvore binária para um *array*, para que assim possamos fazer uma ordenação a fim de obter o *top n* mais facilmente.

Posto isto, a complexidade desta *query* corresponde a $\theta(N/C \log N/C)$, sendo que N equivale ao número de viagens e C ao número total de cidades identificadas, pelo que se percebe o porquê de o tempo de execução ser ligeiramente superior.

Query 8

Também neste problema foi necessário percorrer o *array* das viagens, de modo a obter todos os dados necessários à realização de uma ordenação o mais correta possível, consequentemente obtemos uma complexidade de $N + A \log A \in \theta(N)$, sendo que A equivale ao número de viagens consideradas válidas segundo os parâmetros apresentados na *query*, e N ao número total de viagens realizadas.

Assim, observamos que o tempo de execução desta *query* é similar ao de outras *queries* que executam em tempo linear, o que a princípio pode parecer um pouco estranho, todavia faz todo o sentido, uma vez que o fator N é, normalmente, muito superior que o fator A, portanto como N é linear, consequentemente a execução aproxima-se do tempo linear.

Não obstante, se o valor de A fosse próximo de N, veríamos o tempo de execução crescer significativamente.

Query 9

Por fim, como esta análise possui parâmetros relativos a datas, tivemos de voltar a percorrer o *array* das viagens e recolher todos os dados necessários a fim de fazer uma ordenação final, deste modo, a complexidade desta *query* corresponde a $N + N \log N \in \theta(N \log N)$.

Complexidade das Queries

Query	Melhor Caso	Caso Médio	Pior Caso
1	$\theta(1)$	$\theta(U)$	$\theta(U)$
2	$\theta(N)$	$\theta(N)$	$\theta(N \log N)$
3	$\theta(N)$	$\theta(N)$	$\theta(N \log N)$
4	$\theta(1)$	$\theta(N/C)$	$\theta(N)$
5	$\theta(N)$	$\theta(N)$	$\theta(N)$
6	$\theta(1)$	$\theta(N/C)$	$\theta(N)$
7	$\theta(1)$	$\theta(N/C)$	$\theta(N/C \log N/C)$
8	$\theta(N)$	$\theta(N)$	$\theta(N \log N)$
9	$\theta(N)$	$\theta(N \log N)$	$\theta(N \log N)$

Lengenda

1. $U \rightarrow$ número de viagens realizadas por um utilizador/condutor.
2. $N \rightarrow$ número total de viagens consideradas válidas.
3. $C \rightarrow$ número total de cidades existentes