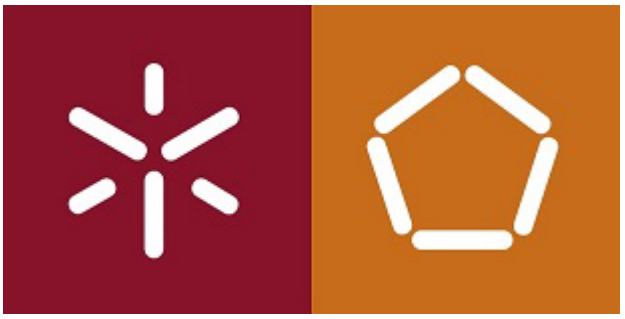


Universidade do Minho
Departamento de Informática

Sistema Estatístico de Viagens

Laboratórios De Informática III
Grupo 06

João Paulo Campelo Gomes, a100747
Rodrigo Viana Ramos Casal Novo, a100534
Mariana Antunes Silva, a100702

Índice

Introdução

O presente relatório tem como objetivo apresentar o projeto realizado no âmbito da unidade curricular de Laboratórios de Informática III, ao longo do primeiro semestre, do segundo ano, da Licenciatura em Engenharia Informática da Universidade do Minho. Este consiste na criação de um programa capaz de ler , armazenar, gerir toda a informação válida contida em ficheiros *csv*(s) tal como o modo interativo.

Modularidade

No que toca a modularidade decidimos dividir o drivers, o users, o rides, a utilities e as queries todas em módulos (*.c) separados para conseguirmos reutilizar o máximo de código possível e assim evitar ter de reescrever código já definido noutros módulos.

Encapsulamento

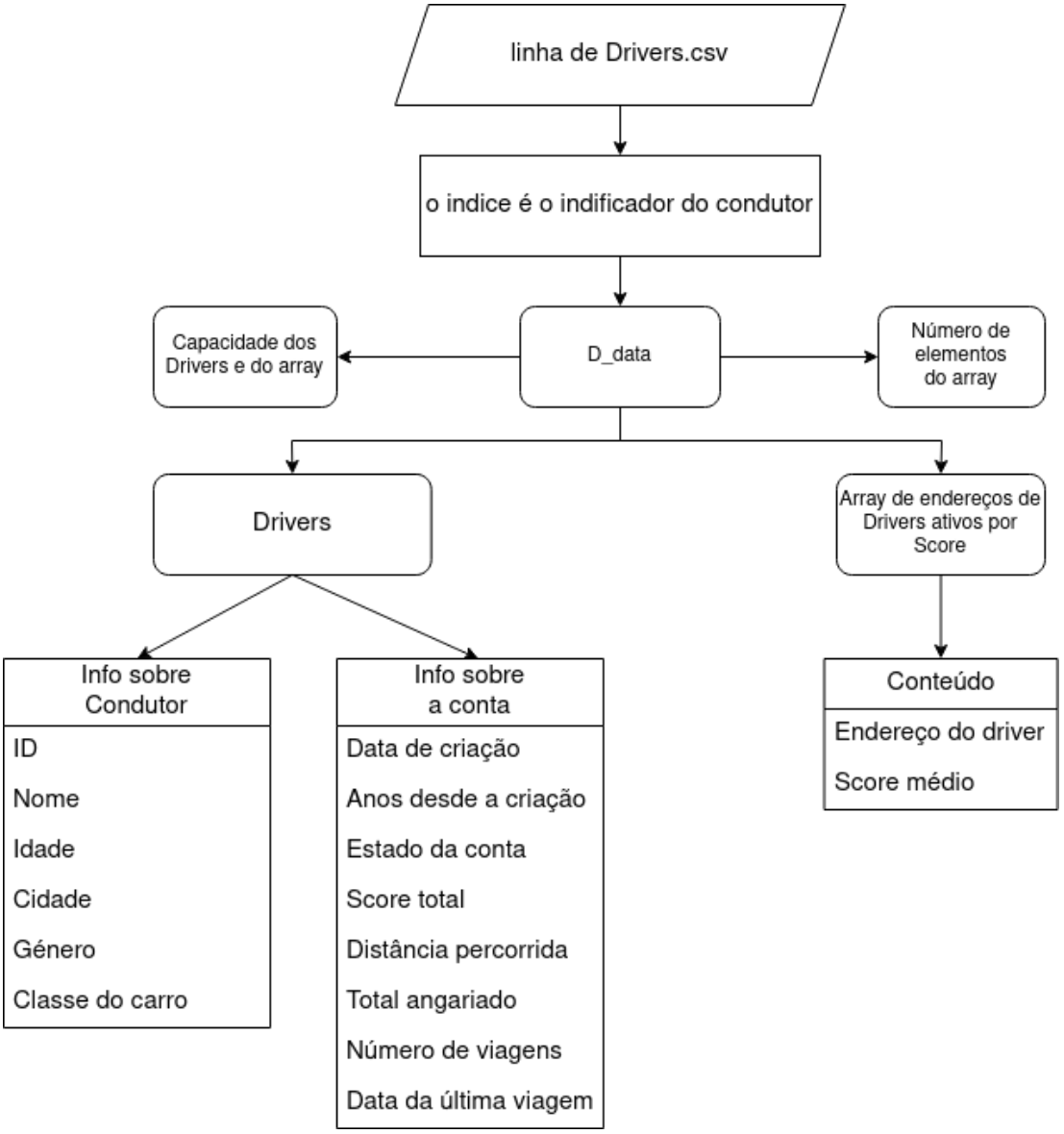
Com o objetivo de visar a questão do encapsulamento decidimos tornar todas as estruturas opacas a outros ficheiros e utilizar funções de get para aceder a cópias dos valores que se encontram no drivers.c, user.c e rides.c.

Módulos

A finalidade deste capítulo é expor detalhadamente, e com a devida fundamentação, a nossa abordagem para o armazenamento, gestão de dados e resolução das queries. Esta componente constitui toda a parte lógica do nosso programa.

2.1 Drivers

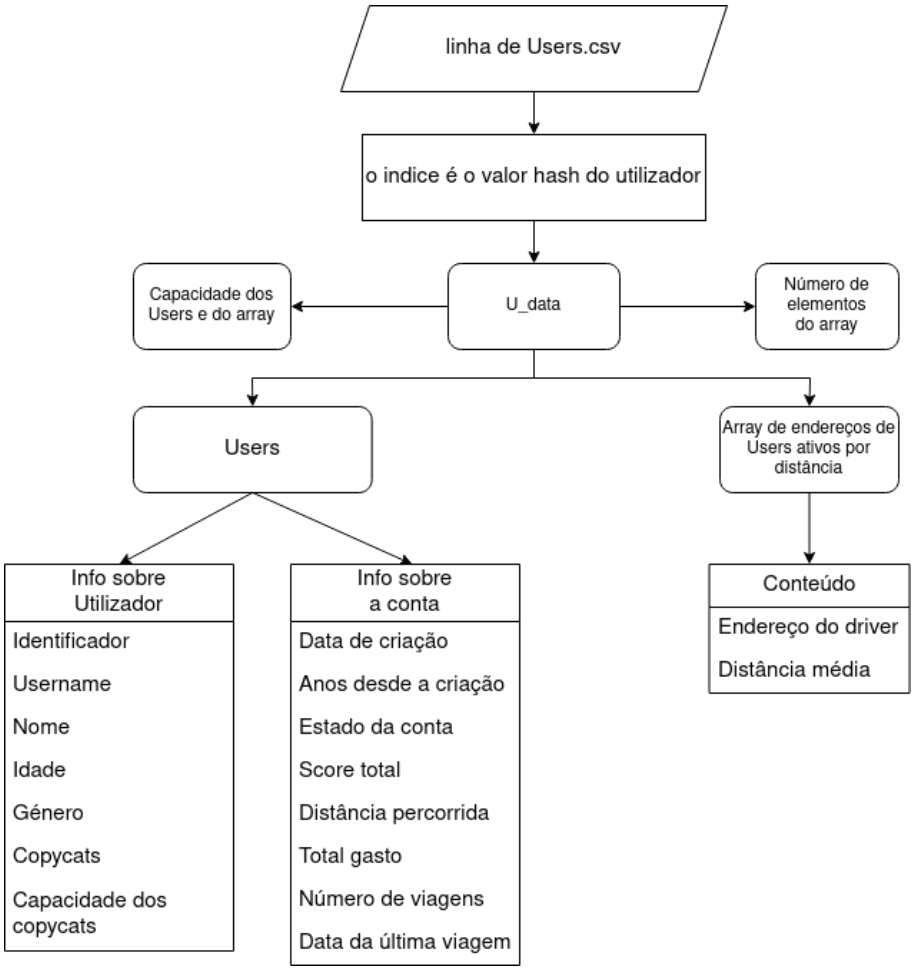
O drivers foi alterado ligeiramente nesta versão do trabalho, pois como pretendíamos ordenar todos os condutores de uma vez para preparação do modo interativo criámos uma nova estrutura chamada D_data, que contém a hash table dos condutores (que usa id - 1 como função de hash), cujo funcionamento se mantém igual ao da primeira fase , e uma outra estrutura nova que guarda os endereços dos condutores ativos e uma variável para facilitar a ordenação para a query 2.



2.2 Users

O users foi alterado ligeiramente nesta versão do trabalho, pois como pretendíamos ordenar todos os usuários de uma vez para preparação do modo interativo criámos uma nova estrutura chamada U_data, que contém a hash table dos usuários(que usa como função de hash a função djb2), cujo funcionamento se

mantém igual ao da primeira fase , e uma outra estrutura nova que guarda os endereços dos usuários ativos e uma variável para facilitar a ordenação para a query 3.

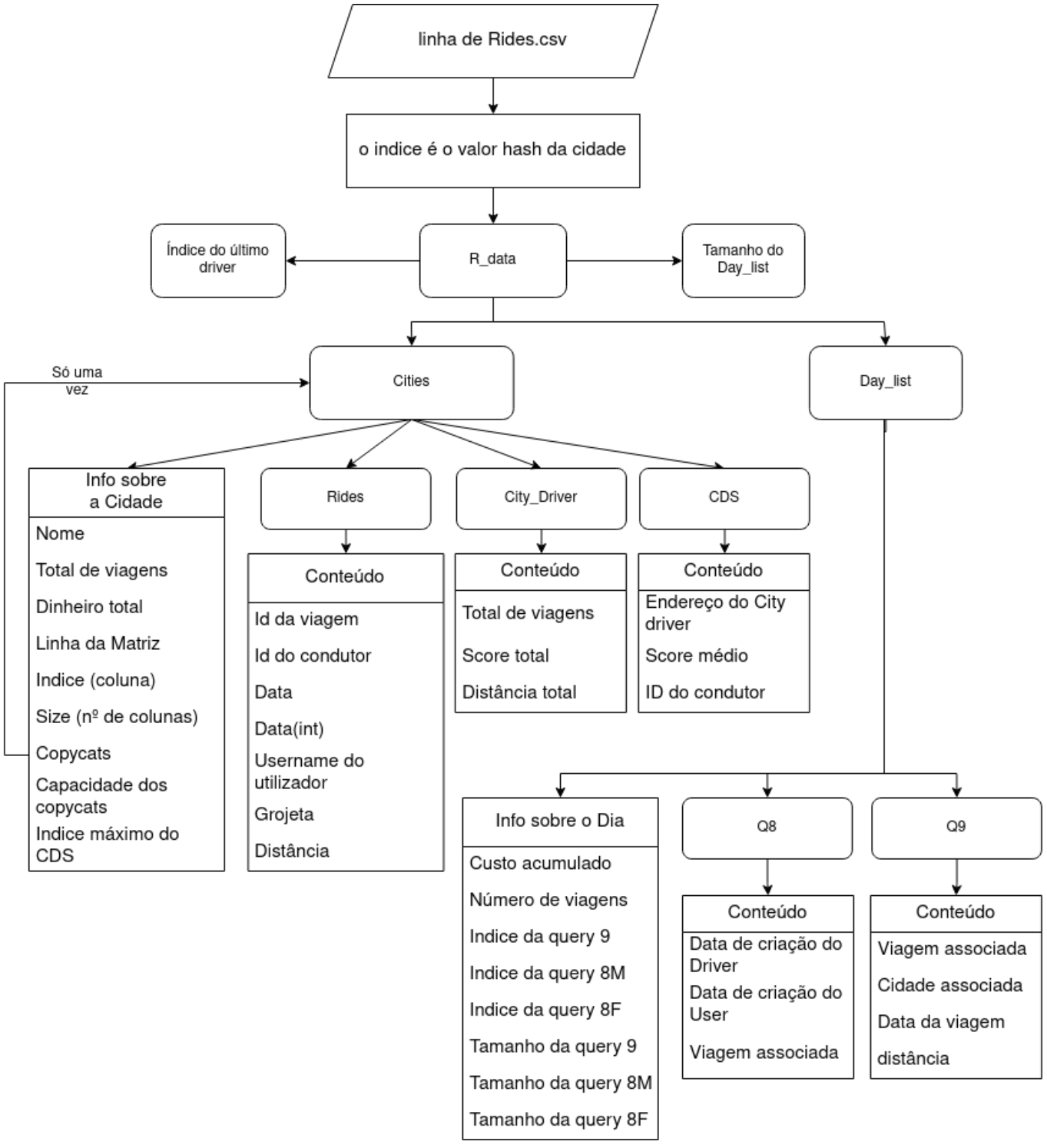


2.3 Rides

A função rides sofreu uma alteração enorme comparada à primeira parte, porque durante o desenvolvimento da parte interativa observamos que não havia diferenças de tempo significativas entre o método que estávamos a usar antes de guardarmos todas as viagens e o interativo. Por isso descartamos a ideia de termos funções diferentes para o modo batch e para o modo interativo.

Devido ao grande volume de informação presente nas rides e procurando sempre o melhor tempo de execução possível a uma larga escala, decidimos guardar fragmentos das viagens em estruturas pensadas em facilitar as queries 4 a 9, uma nota importante é que como priorizamos o tempo de execução sobre a memória ocupada, há dados iguais em estruturas diferentes, isto é feito sem quebrar o encapsulamento pois apesar de os dois dados terem a mesma origem, uma vez guardados são independentes.

A estrutura principal da rides é a R_data, estrutura esta que contém um array de uma estrutura chamada Cities (que é uma hashtable para as cidades que funciona de forma muito semelhante ao users, mas com uma função de hash mais simples), esta estrutura possui por sua vez uma estrutura chamada c_driver que é uma hashtable dos condutores (semelhante ao drivers_table), e um array com o endereço de todos os c_drivers com pelo menos uma viagem e valores úteis à execução da query 7. Para além disto a estrutura R_data, ainda possui um array de uma estrutura chamada Day_list que é usado para guardar todas as informações relativas a querys relacionadas com tempo.



2.4 Interactive

No caso de o utilizador não executar o programa com os endereços necessários para o modo bash, é necessário pedir ao utilizador essa informação, para isso criamos o módulo interactive. Com o auxílio da biblioteca *ncurses.h,* é pedido ao utilizador o caminho para a pasta com os ficheiros CSV, caso esse caminho não seja válido é pedido de novo. Depois é apresentado ao utilizador as várias opções de queries disponíveis e este pode escolher de forma facilitada o que pretende fazer.

Para queries onde o número de linhas pode superar o tamanho de apresentação criamos um método de paginação para possibilitar ao utilizador a visibilidade de todos os dados.

Nós decidimos que o melhor era criar o ficheiro tal como no modo bash é feito para o pedido ao invés de ficar apenas no terminal, uma vez que assim conseguiria ser utilizado, posteriormente, pelo utilizador. No futuro poderia ser uma melhoria perguntar ao utilizador se ele pretende guardar informação ou não, possivelmente poupando tempo.

Caso o utilizador decida alterar os ficheiros CSV utilizados, permitimos que ao longo da execução volte ao início do modo e insira de novo um caminho para os ficheiros CSV desejados.

2.5 Queries

2.5.1 Query 1

Esta query é relativamente simples uma vez que estamos a usar hashtables como estruturas de dados, pelo que apenas precisamos de determinar em que tabela temos de procurar. Para isso percorremos a string que recebemos do input até encontrarmos um caractere que não seja um dígito, se encontrarmos sabemos que recebemos um username, caso contrário é um id.

2.5.2 Query 2

Na primeira fase do trabalho tínhamos optado por usar listas ligadas para ordenar os condutores por avaliação, no entanto comparando o tempo entre ordenar tudo com quicksort e o método anterior não houve diferenças significativas para os inputs pedidos, no entanto a versão da primeira fase era substancialmente mais lenta para o modo interativo então optamos por usar o quicksort para ambos por uma questão de simplicidade e para haver menos oscilações de tempo para diferentes inputs.

Como nós organizamos todos os drivers por avaliação, quando nos é pedido para executar essa query, apenas temos de escrever os primeiros n elementos (onde n é o valor que nos foi dado como argumento da query, ou todos os drivers existentes caso o argumento seja maior que o número de condutores) sendo por isso a query linear.
Nota: Nós organizamos um array com os endereços dos condutores e a respetiva avaliação média e não a hashtable em si, para a preservar.

2.5.3 Query 3

O funcionamento da query 3 é exatamente igual ao da query 2, mas ao invés de organizarmos condutores por avaliação média, organizamos usuários por distância percorrida.

2.5.4 Query 4

Para a execução desta query aproveitamos a estrutura *R_data*, nesta estrutura estão todas as cidades e informações relacionadas com as mesmas, como tal, para o resultado, localizamos a cidade na estrutura através do seu identificador e através das funções *get_City_money* e *get_City_totalrides*, calculamos a média da respetiva cidade. Se a cidade não existir na estrutura é porque não existe nenhuma viagem relacionada com a mesma, nessa situação, não é escrito nada no ficheiro output. Tendo em conta esta aproximação a execução desta query é em tempo constante.

2.5.5 Query 5

Visto que neste pedido é necessário percorrer as datas adjacentes às recebidas rapidamente percebemos que o melhor seria criar uma estrutura nova (guardada dentro de *R_data*) chamada *Day_list* esta estrutura é preenchida de forma a que a informação de um dia está armazenada no índice correspondente ao número de dias que passaram desde a viagem até ao “hoje”(data de referência), onde hoje é a origem do array o que permite reallocs caso sejam necessários. Para processar o pedido iteramos, na função *média,* esse array desde a posição respetiva à data final até à posição da data inicial, guardando o somatório em parâmetros. A média é calculada e retornada para a função que processa a query respeitando o encapsulamento. caso não exista viagens no intervalo o ficheiro será vazio.

2.5.6 Query 6

Para a sexta query segue um procedimento parecido ao da quarta query no que toca a encontrar as cidades na estrutura, difere no entanto na média, uma vez que nem todas as viagens devem ser incluídas. Decidimos então guardar por ordem de chegada as viagens respectivas a essa cidade. através de vários testes percebemos que organizar as viagens para as aceder mais tarde demoraria mais do que percorrer simplesmente de forma linear. Então, na função *getDistMed,* percorremos as viagens da cidade e, caso respeite o intervalo dado pelo pedido, é adicionado ao total da distância e incrementado o número de viagens. Retorna então, à semelhança da query 5 para ser impresso.

2.5.7 Query 7

Como nós organizamos todos os condutores por avaliação nas respetivas cidades, quando praticamos a query apenas temos de escrever os top n pedidos.
Nós apenas organizamos o array com os endereços dos condutores, tal como a avaliação e o id do condutor a partir do quicksort.

2.5.8 Query 8

A query 8 é diferente das restantes, porque para além de utilizar apenas os dados que estejam dentro de um certo intervalo de tempo, ainda é preciso que estes sejam do gênero pedido e depois têm de ser ordenados. Devido a esta série de restrições decidimos criar uma estrutura no R_data que facilitasse aceder a estes dados. Para isso decidimos colocar a estrutura da query 8 dentro da Day_list uma vez que esta foi criada para segregar as viagens por data, no entanto como esta query funciona à base de anos e não de dias, seria pouco eficiente guardar os dados no dia exato, pois isso só levaria a mais cache misses e mais instruções, pelo que convertemos o dia da criação de conta em anos desde a sua criação, permitindo-nos assim fazer um único acesso ao Day_list para termos acesso a todas as viagens que respeitam os critérios desta query com conta à X anos.
Para executar a query são percorridas as posições do Day_list desde a idade de perfil passada como argumento até à posição 22 (escolhemos este número porque não há nenhuma conta com mais de 12 anos, mas queríamos que o trabalho funcionasse para outros datasets, por isso limitamos a pesquisa até ao ano 2000, ano onde a internet foi disponibilizada em Portugal) copiando todos os dados para um array que posteriormente será ordenado com quick sort. Nesta query optamos por fazer uma cópia dos dados porque como é relacionada com tempo as viagens disponíveis mudam de acordo com o argumento que nos foi passado, pelo que uma organização geral não é ideal.

Nota: Dentro da estrutura Day_list existem 2 arrays da estrutura Q8, porque existem 2 gêneros e assim cada estrutura tem os dados relativos apenas ao respectivo gênero o que acelera a execução da query.

2.5.9 Query 9

Para ordenar todas as viagens entre as datas pedidas por distância, optamos por criar um array, apenas quando realizamos esta query, com os endereços da Q9, que nos dá acesso às viagens e às cidades, tal como a respetiva distância e organizar a partir do quicksort.

2.6 Query Handler

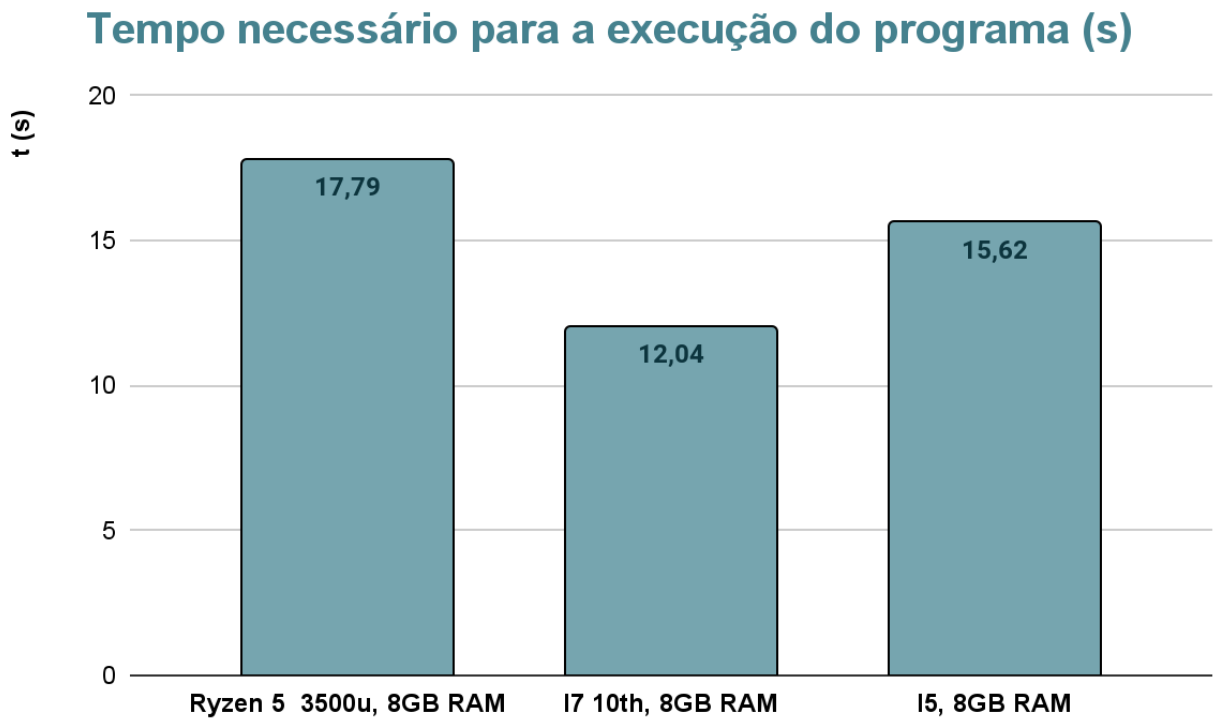
É neste módulo que identificamos quais as queries pedidas e os referentes argumentos tal como as funções que vão executar cada uma destas de forma quase instantânea.

Estatísticas de Execução para os ficheiros Data-Large

Para a obtenção destes resultados usamos os ficheiros **Data-Large** sem entradas inválidas e os nossos testes funcionais e de desempenho.

Tempo médio necessário para processar e executar cada query (*μs*)

	Ryzen 5 3500u, 8GB RAM	I7 10th, 8GB RAM	I5, 8GB RAM
Query 1	0.083	0.207	1,180
Query 2	189.700	129.700	173.9
Query 3	2.600	1.100	2.000
Query 4	0.360	0.240	1.440
Query 5	0.260	0.280	0.440
Query 6	83.18	78.740	113.9
Query 7	0.433	0.400	0.767
Query 8	1765.200	1866.000	1994.800
Query 9	4.667	4.733	8.467



A query 8 é a query mais lenta do nosso projeto e a única que demora mais que um milisegundo e isto deve-se ao facto de ser uma query que, por ser relacionada com intervalos de tempo obriga a que os dados relativos a esta sejam processados durante a sua execução ao contrário da maioria das outras onde os dados são processados anteriormente à query.
A query 2 nestes testes aparece como sendo a segunda mais lenta, o que se deve a um simples infortúnio nos testes realizados, uma vez que a query 2 e 3 são idênticas a única diferença de tempo entre elas deve-se ao tamanho do top pedido nos inputs.
A query 6 e a query 9 são as duas relacionadas com intervalos temporais, o que ,tal como na query 8, obriga a que os dados relativos a estas sejam processados durante a execução das mesmas, no entanto estas queries são mais rápidas que a 8, porque como estas são intervalos entre dias e a 8 é entre anos, estas queries geralmente processam um volume menor de dados.
As restantes queries são todas bastante rápidas pois elas apenas requerem aceder à estrutura de dados e escrever num ficheiro o valor lido, o que é feito em tempo constante devido à natureza das nossas estruturas.