Universidade do Minho

Licenciatura em Engenharia Informática

LABORATÓRIOS DE INFORMÁTICA III

Ano Letivo 2022/2023

Projeto – Fase II

Fábio Jorge Almeida Cunha, 105089
Gabriel Araújo, 102023
Tiago Ferreira Soares, 97381

Introdução

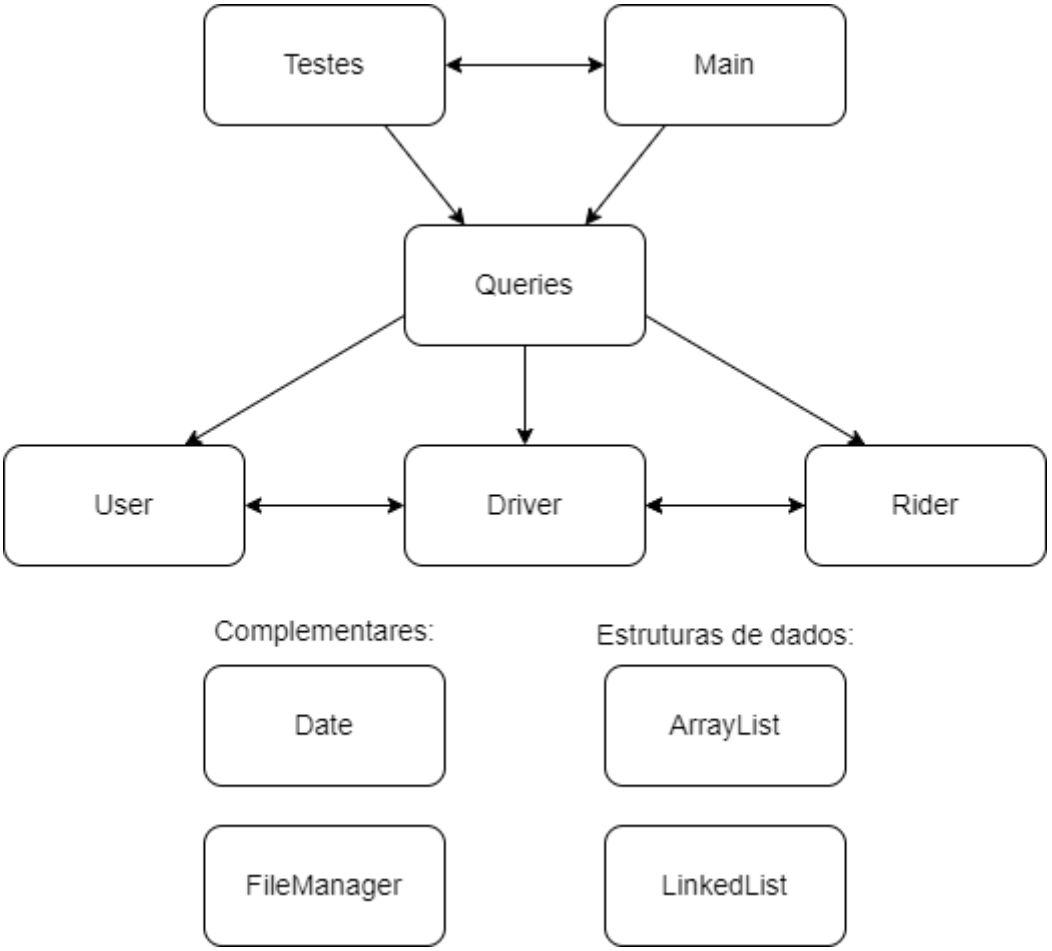
Este projeto consiste na criação de um programa em linguagem C, que realiza queries de consulta de informação sobre um conjunto de dados (Rides, Users e Drivers).

O programa tem os seguintes modos de execução:

- Modo Batch: Neste modo, o utilizador invoca o programa na linha de comandos juntamente com o caminho da pasta onde se encontram os ficheiros de dados e o caminho para um ficheiro de texto com as queries que pretende realizar como argumentos.
- Modo Interativo: Neste modo, o utilizador invoca o programa sem argumentos. Em seguida, é-lhe pedido a query que quer realizar e quais os seus argumentos.
- Modo Testes: Este modo é invocado como o modo Batch juntamente com o caminho para a pasta de testes como argumento adicional.

Arquitetura do programa

De modo a fazer com que os dados fiquem o mais escondidos possível, optamos pela seguinte hierarquia:



Os únicos módulos que vão trabalhar com os dados e que podem trocar dados entre si, são os módulos de nível mais baixo da hierarquia. Os outros níveis não terão acesso aos dados, obtendo assim, o encapsulamento desejado.

Os módulos complementares e de estruturas de dados, apenas vão trabalhar com cópias dos dados, nunca podendo alterar os originais.

Modularização

Para a realização deste projeto, decidimos utilizar os seguintes módulos:

ArrayList, Date, Driver, FileManager, LinkedList, Main, Queries, Ride, Tests e ***User.***

Em seguida, passamos a explicar o que cada um faz:

- ***Driver***, ***Ride*** e ***User:*** Estes três módulos estão no fundo da hierarquia. São responsáveis por carregar os dados do seu respetivo ficheiro para memória. Além disso, é aqui que se realizam todas as operações de consulta de informação e cálculo de estatísticas pedidas pelas *queries*. As funções aqui criadas devolvem apenas valores ou cópias dos dados que estão nos ficheiros.

- ***Date***: Módulo onde transformamos a *string* de data contida nos ficheiros, separando-a em inteiros e agrupando-os numa estrutura do tipo Date. Também realizamos aqui, algumas operações sobre datas, como, por exemplo, calcular a idade de uma pessoa.

- ***Queries***: Módulo que lê um ficheiro com comandos de consulta de informação de dados e os executa em seguida, chamando as funções que calculam o resultado respetivo a cada um deles.

- ***FileManager***: Módulo que faz o tratamento de ficheiros. Lê os ficheiros de dados para depois serem carregados para memória e escreve os resultados de cada comando no seu respetivo ficheiro.

- ***Main***: Módulo que decide o modo de uso e envoca as suas respetivas funções de carregamento, execução e limpeza.

- ***Tests***: Módulo que compara os outputs obtidos da execução do programa com os outputs desejados. Também fornece estatísticas da execução, como o tempo total de execução, o número de vezes que cada query é executada e quantos outputs são considerados corretos e a média do tempo de execução de cada query.

- ***ArrayList*** e ***LinkedList***: Módulos de estruturas de dados. O primeiro usa um *array* de tamanho estático que guarda em memória todos os dados contidos nos ficheiros. O segundo são listas ligadas que são maioritariamente usadas juntamente com *Hashtables* do *glib* para categorizar os dados do módulo ***Ride***.

Encapsulamento de dados

O encapsulamento de dados é uma técnica de programação que permite controlar o acesso aos dados e às informações armazenadas no programa. Isto ajuda a evitar que os dados base sejam alterados ou vistos por outros módulos do programa.

Para concretizar isto, nós seguimos a hierarquia que temos no ponto acima, em que cada módulo é responsável pelos seus próprios dados que nunca são diretamente visíveis para outros, tal como a maneira como os dados são descritos não é disponível fora do seu módulo. Também criamos duas estruturas de dados, Array List e Linked List, que armazenam e operam sobre dados de uma maneira genérica e opaca.

Testes de desempenho

Estes testes de desempenho foram desenvolvidos para medir e comparar quanto tempo o programa demora a executar as queries sobre diferentes datasets, sempre com mesmo input. Os testes foram executados pelos 3 membros do grupo na sua respetiva máquina.

Especificações dos ambientes de teste

Fábio:

- Processador: 12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
- Disco: G-Technology G-Drive Mobile SSD R-Series 500GB USB 3.1 Type C
- Memória RAM: 16GB
- Sistema Operativo: Ubuntu 22.04.1 LTS

Gabriel:

- Processador: AMD Ryzen 5 5500U with Radeon Graphics
- Disco: 512 GB SSD Micron MTFDHBA512QFD
- Memória RAM: 6GB
- Sistema Operativo: Ubuntu 22.04.1 LTS

Tiago:

- Processador: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
- Disco: SAMSUNG MZVLB512HAJQ-00000 (EXA7301Q) 512GB
- Memória RAM: 8GB
- Sistema Operativo: Ubuntu 22.04.1 LTS

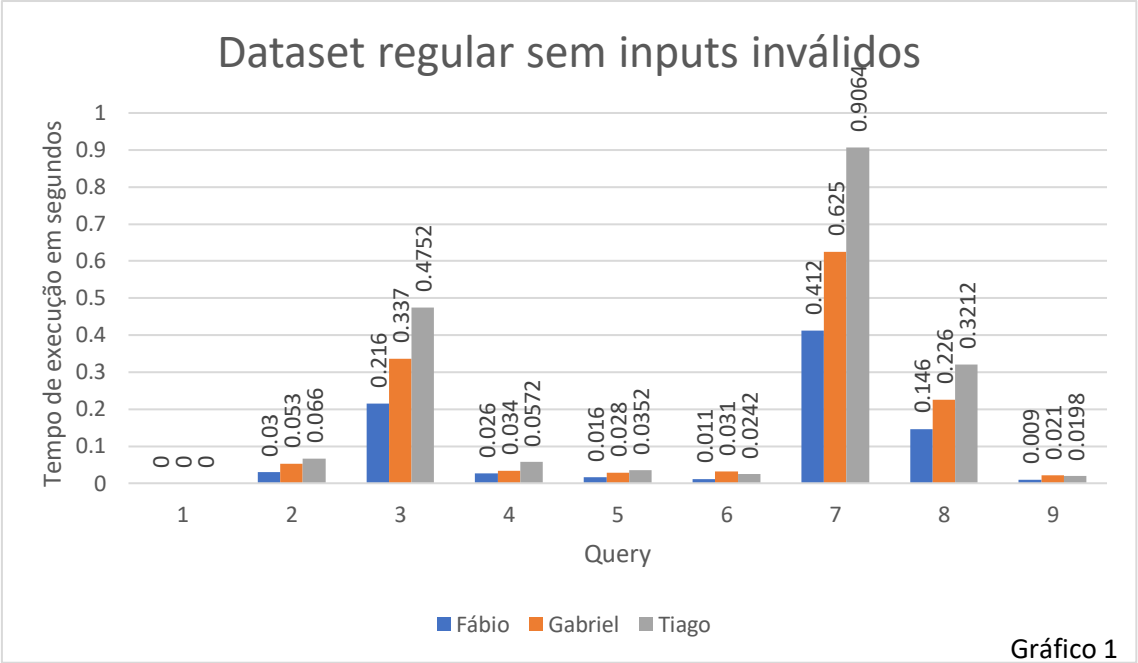


Gráfico 1

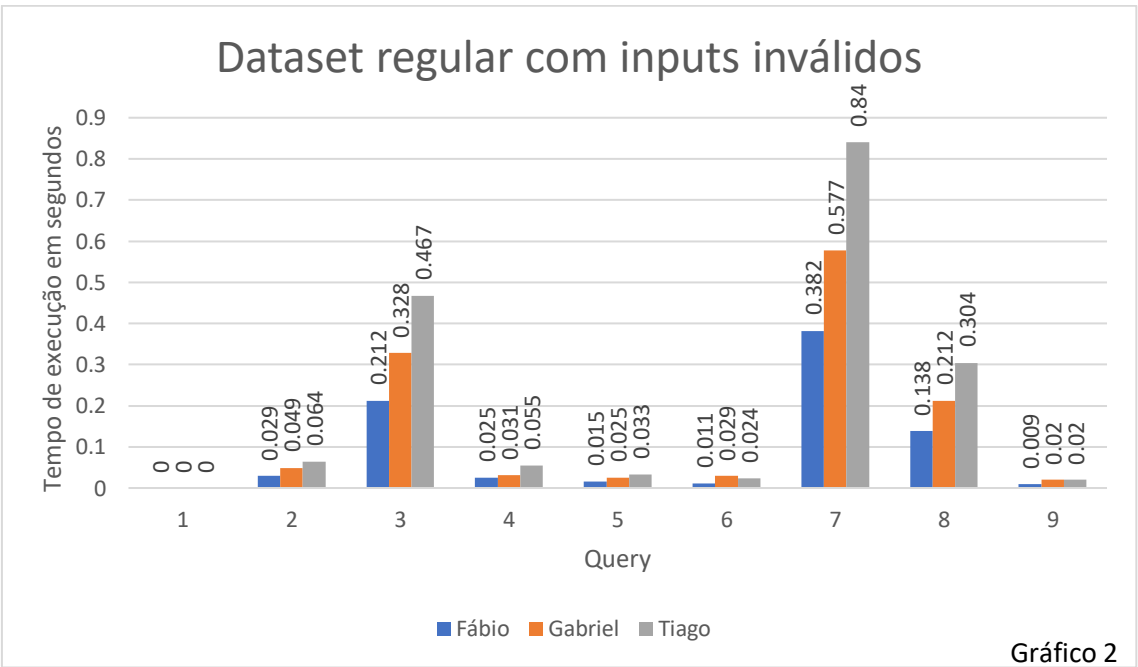


Gráfico 2

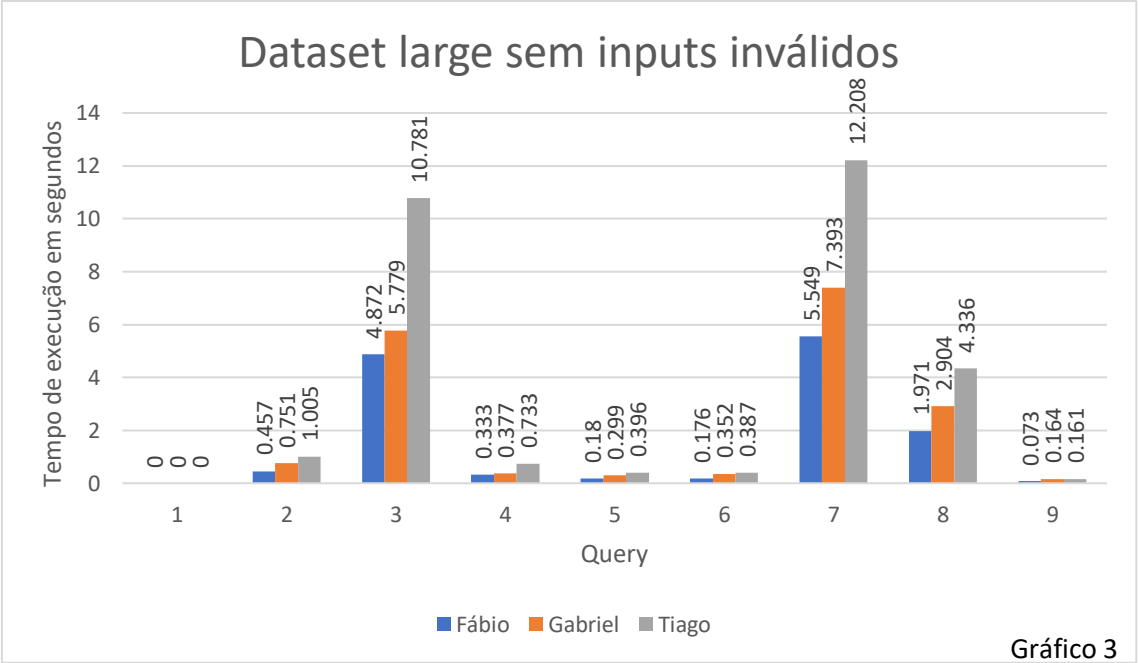


Gráfico 3

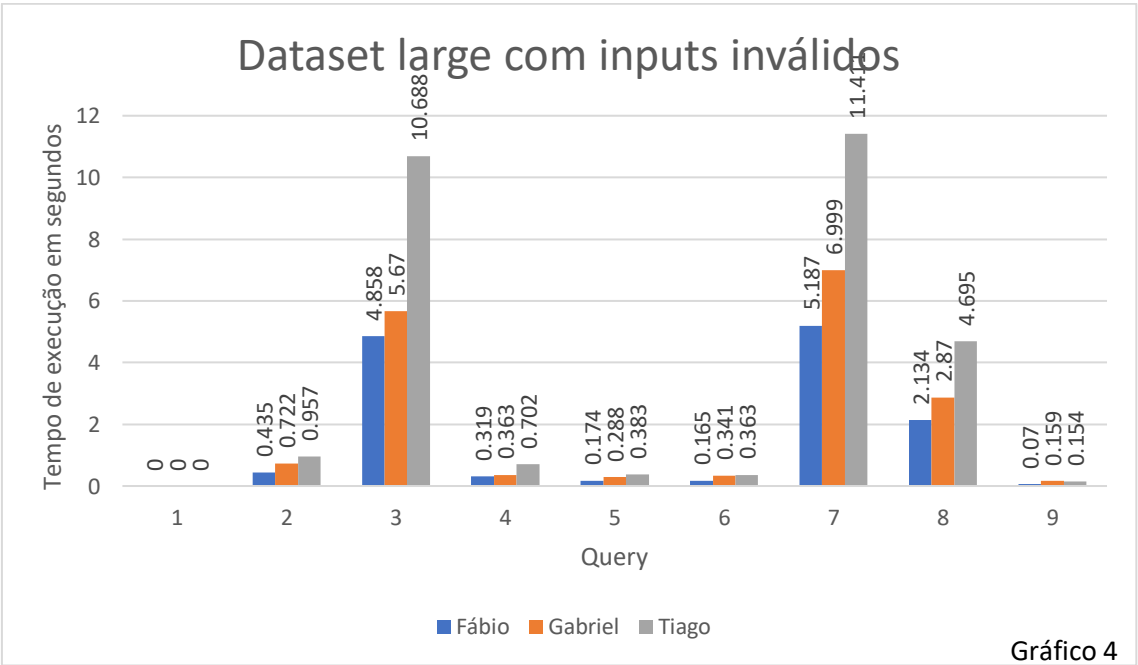


Gráfico 4

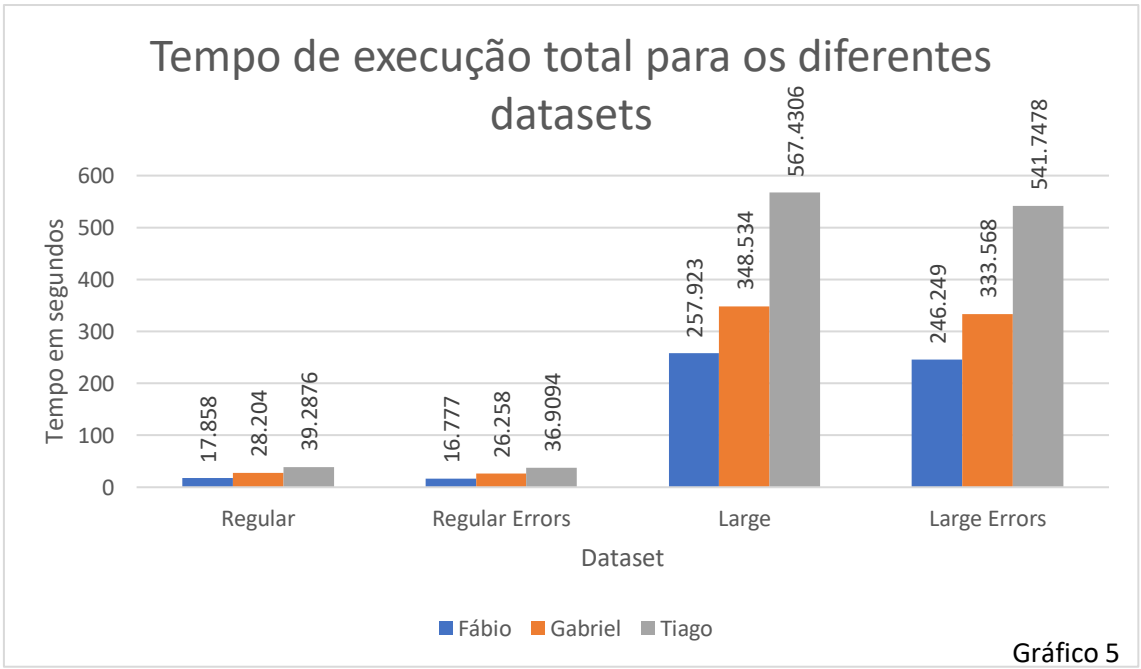


Gráfico 5

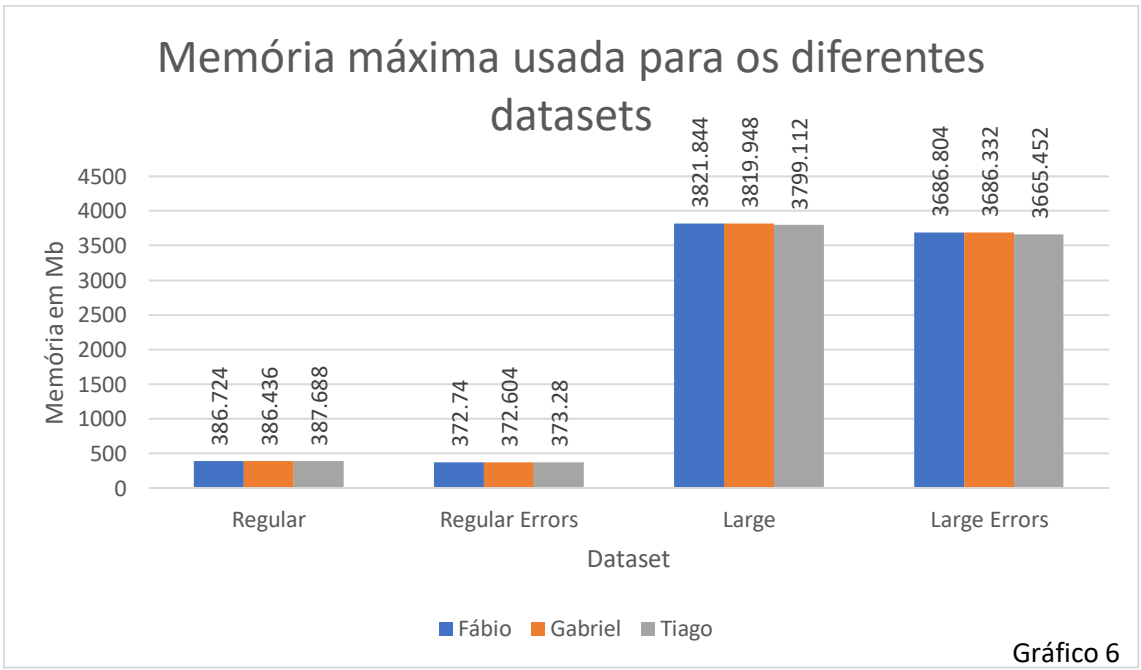


Gráfico 6

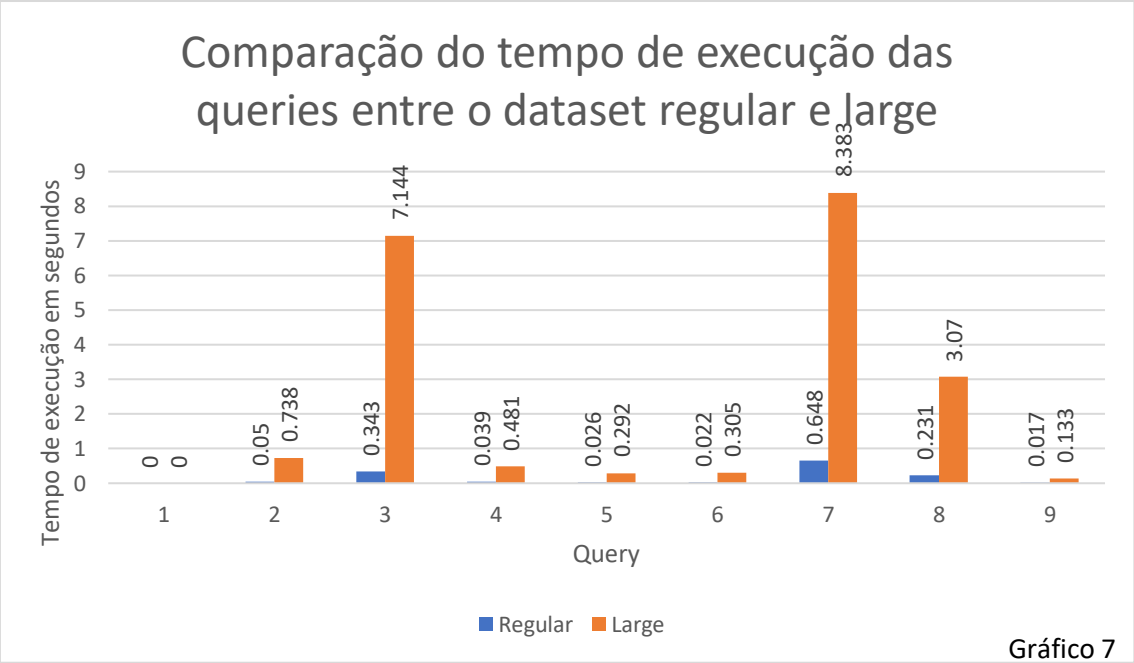


Gráfico 7

Conclusão

Inicialmente usávamos várias estruturas de dados (hashtables, principalmente) para agilizar a realização das queries, em troca de um tempo de inicialização do programa mais longo. Com o dataset regular funcionava, mas quando passámos a testar o dataset large tornou-se inviável pois o consumo de memória era muito elevado em comparação com o tamanho do dataset. Com isto, eliminámos algumas hashtables para consumirmos menos memória, em troca de de um tempo de realização das queries mais longo.

Ainda assim, estamos a utilizar uma quantidade de memória elevada em comparação ao tamanho dos datasets (≈4.3x a mais para o regular e ≈2.9x a mais para o large).

O tempo de execução das queries sobre o dataset regular para o large aumentou em média, ≈13.38x. O que está relativamente em linha com o aumento do tempo de execução total (≈13.75x) e um pouco a baixo do aumento do tamanho do dataset (≈14.8x).

O espaço ocupado poderia, provavelmente, ser melhorado usando diferentes estruturas de dados e o tempo de execução das queries poderia também ser melhorado usando algoritmos de pesquisa e ordenação mais otimizados.