Relatório da Fase 1 - Grupo 03

Francisco Macedo Ferreira (A100660)
Hugo ... Ramos (A100644)
Júlio José Medeiros Pereira Pinto (A100742)

22 de novembro de 2022

1 Introdução

Este relatório tem como intuito abordar a primeira fase do projeto da UC Laboratórios de Informática III do ano letivo 2022/2023.
Nesta fase como objetivos o trabalho necessitava a implementação de *parsing* dos dados de entrada, funcionar através do modo de operação *batch*, realizar pelo menos um terço das *queries* (pelo menos 3 *queries*) e deverá também considerar que o tamanho dos ficheiros de entrada (*users.csv, drivers.csv* e *rides.csv*), em linhas, terá 100.000, 10.000 e 1.000.000, respetivamente. Ainda mais, no projeto, em geral, existe uma necessidade do uso de conceitos de encapsulamento e modularidade.
Este relatório irá abrangir as decisões tomadas pelo grupo tal como o método de raciocínio para desenvolvimento do mesmo.

2 Desenvolvimento

2.1 Pipeline atual

A nossa pipeline foi alvo de bastante discussão entre nós, porém conseguimos chegar a um consenso com a pipeline atual. Neste momento a nossa pipeline funciona da seguinte maneira. A nossa main recebe dois argumentos, argumentos estes que serãos os *paths* para os ficheiros necessários, a pasta do *dataset*, onde se encontram os .csv, e o ficheiro de *input*. De seguida, este abre os nosso ficheiros (ficheiros que se encontram na pasta *dataset*) criando *pointers* para os mesmos, assim podendo realizar o *parsing* destes. No *parsing* a função irá receber um argumento de *stream* (o ficheiro que irá dar *parse*) , uma função para o *parse* linha a linha do ficheiro (Função que irá criar a estrutura que pretendemos através da linha), uma função *register* que irá enviar a estrutura criada para um catálogo e por último um argumento, que do ponto de vista do código não sabemos o tipo do mesmo, porém podemos deduzir que será o catálogo para onde enviaremos as estruturas criadas pelo *parser*.
Após finalizar o *parsing* dos ficheiros, este irá ordenar o nosso catálogo da maneira mais útil para a realização das queries (Irá ordenar o *array* de condutores por score, o *array* de utilizadores por distância total percorrida e o *array* de viagens por data). Desta maneira a tornar as queries mais eficientes quando forem a ser realizadas.
Por último esta irá ler o ficheiro de *input* e irá dar parse ao mesmo, realizando então as queries, através do nosso *query manager* que gere as funções para a realização das queries de maneira a utilizar a query certa dependendo do *input* passado. Assim, após libertarmos a memória alocada, concluindo o nosso pipeline atual.

2.2 Estratégias Seguidas

Para a primeira fase do trabalho decidimos implementar as bases do projeto, como o parser e definir as estruturas dos dados que utilizaremos ao longo do projeto (Como o catálogo para guardar os dados e os *users, drivers* e *rides* de forma a guardar os respetivos campos dos utilizadores, condutores e viagens). Ainda mais implementamos as *queries* 1, 2, 3, 4 e 5.

2.2.1 Parser

De maneira a tornar a leitura e o *parsing* dos ficheiros de entrada relativamente eficiente, decidimos percorrer o ficheiro linha a linha. Sempre que este lê uma linha, de acordo com a função de *parsing* passada nesta, este irá dar *parse* à linha que recebe, transformando o na respetiva estrutura de dados (*users, drivers* e *rides*). Ainda mais este insere a estrutura de dados no respetivo array do catálogo (argumento passado na chamada da função porém do ponto de vista do código argumento o qual não sabemos o tipo).
Futuramente poderemos otimizar o código de maneira a tornar a leitura do ficheiro mais rápido, visto que o tamanho dos ficheiros também irá aumentar, tal como poderemos otimizar o registo das estruturas de dados para o catálogo.

2.2.2 Query Manager

Face a necessidade da gestão das queries, implementamos um *Query Manager*.

2.2.3 Query 1

Para acesso rápido a perfis pelo *id's* de condutores ou *username's* de utilizadores foram utilizadas duas *HashTables* (uma para indexar os perfis de condutores e outra para perfis de utilizadores) com *key id e username* e *value Driver* e User** respetivamente. Esses *HashTables* são populados conforme a leitura dos ficheiros .csv e informações como número de viagens, soma total de avaliações (para poder ser calculado a média), total gasto/auferido eram calculados e guardados na estrutura de dados dos utilizadores e dos condutores[1]. A decisão de buscar informações de utilizadores ou condutores é feita pela confirmação se o primeiro argumento da query é um número ou não: se for um número busca-se por condutores (*id*), se não busca-se por utilizadores (*username*).

Com isto é possível um acesso em tempo constante a essas informações, com o custo da leitura dos ficheiros ser um pouco maior (devido ao calculo necessário de *hashes* para cada *key* e não só). Como na segunda fase o número de perfis irá aumentar exponencialmente e é esperado múltiplos acessos a estas informações, seja de várias queries destas ou do modo interativo ainda a implementar, o custo de leitura dos ficheiros superior é muito justificado.

2.2.4 Query 2

Na query 2, para rápido acesso aos condutores com maior média foi feito *sorting* da *array* (presente no catálogo) de condutores, conforme a sua média, no fim da leitura dos ficheiros. Isto poderá ser otimizado a fazer com que o *sorting* seja *lazy* (será abordado este tópico mais tarde). Durante a execução da query basta obter os N primeiros condutores da *array* e temos a execução em práticamente em tempo constante (agora é copiado os N elementos para uma array, mas poderá ser otimizado).

[1] Pode ser uma possível quebra de encapsulamento. Ainda aguardamos a resposta do docente sobre esse quesito.

2.2.5 Query 3

Para a query 3, tal como na query 2, foi feito um *sorting* da *array* (presente no catálogo) de utilizadores, de maneira a otimizar o acesso aos users com maior distância total percorrida. Tal como na query 2, o método de *sorting* poderá ser otimizado da mesma maneira. Durante a execução desta query o tempo de execução será praticamente constante, porém poderá vir a ser otimizado a maneira como os N primeiros utilizadores são guardados.

2.2.6 Query 4

Nesta query, o preço médio das viagens numa determinada cidade é calculada (em tempo linear conforme o número de viagens por cidade) durante a execução da query. Durante a leitura das viagens, é inserido a viagem conforme a sua cidade numa *HashTable (key: cidade, value: Array de Ride*).* O preço médio não é pré-calculado, pois o cálculo desta é relativamente rápido devido às viagens já estarem separadas por cidade e é expectável que só se aceda a este valor uma vez, por isso guardá-lo será desnecessário. Esta *HashTable* já existe devido à query 7 que necessitará de acesso rápido a viagens conforme a sua cidade.

Futuramente, conforme a expansão do dataset, esta implementação poderá ter que ser reformulada, por causa do cálculo em execução mas para já tivemos bom desempenho com a atual.

2.2.7 Query 5

Já na query 5, no fim da leitura das viagens é feito *sorting* das viagens pela sua data, por ordem crescente. Com isto, assumindo que <data A> e <data B> são os argumentos da query, basta aceder ao primeiro elemento a partir do qual <data A> é menor ou igual do que a data desse elemento. A partir daí, podemos percorrer a lista até encontrar uma viagem que a sua data seja maior que <data B>, acumulando o preço das viagens para no fim calcular a sua média. O tal primeiro elemento é encontrado com uma implementação semelhante à **std::lower_bound** de *C++*, usando *binary search.* Isto é possível devido à lista estar organizada pela data das viagens.

Para evitar percorrer a lista, podia ter sido pré-calculado uma *array* em que cada indíce tinha o somatório de preços para trás desse indíce e o preço médio era calculado subtraindo o *upper_bound* com o *lower_bound* do *range* das datas e dividindo pelo número de elementos entre eles. Essa ideia foi rápidamente descartada devido às datas dos argumentos serem relativamente perto, portanto são poucas as viagens a iterar. Esta solução também iria aumentar consideravelmente o tempo de leitura dos ficheiros.

2.3 Análise de desempenho

Comparando o desempenho da execução das queries 1, 2, 3, 4 e 5 para os ficheiros de input do conjunto de testes expandido das pastas *tests1* e *tests2* (excluindo as queries não implementadas) temos na Tabela 2 os resultados conforme as especificações dos computadores na Tabela 1.

O programa foi compilado com as flags *-O2 -flto*.

	PC 1	PC 2	PC 3
CPU	M1 Pro 8-core (6 perf. e 2 ef.)	Intel i7-8550U (8)	-
RAM	16GB RAM LPDDR5	8GB RAM DDR4 2400MHz	-
Disco	500GB NVME	500GB NVME	-
OS	MacOS Ventura 13.0.1	ArcoLinux Kernel: 6.0.9-arch-1	-
Compilador	Apple Clang 14.0.0	GCC 12.2.0	-

Tabela 1: Performances em diferentes PCs

	PC 1	PC 2	PC 3
tests 1.txt	-	-	-
tests 2.txt	-	-	-

Tabela 2: Tempos de execução em diferentes PCs

2.4 Possíveis otimizações e futuras limitações

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3 Conclusão

Para concluir, acreditamos que o desenvolvimento desta primeira fase foi uma boa maneira de consolidar tanto os nosso conhecimentos de C com conhecimentos mais avançados relacionadas a gestão de memória, encapsulamento e modularidade. Apesar de tudo, a nosso ver ainda existe espaço para melhorias no uso destes conceitos e pretendemos desenvolve-los a um nível ainda mais avançado do que os apresentados na próxima fase do trabalho.