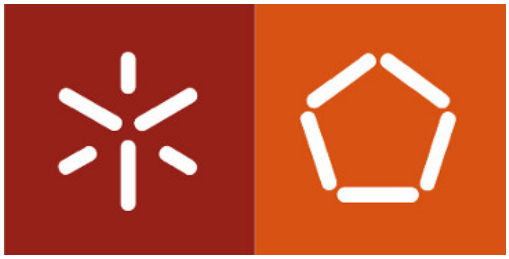

Universidade do Minho

Licenciatura em Engenharia Informática

Grupo 14

Filipe Simões Pereira, A100552

João Pedro Silva Lopes, A100829

Duarte Machado Leitão, A100550

Laboratórios de Informática 3
2º Ano, 1º Semestre
Departamento de Informática

22 de novembro de 2022

Índice

1 Introdução

No contexto da unidade curricular de Laboratórios de Informática III foi proposta a realização de um projeto que permitisse desenvolver conhecimentos essenciais da linguagem C e de Engenharia de Software, nomeadamente, modularidade e encapsulamento, estruturas dinâmicas de dados, validação funcional e medição de desempenho. hyperref

2 Modelação

2.1 Arquitetura de Aplicação

A arquitetura do projeto é constituída por 3 entidades principais:

- ***Parser de dados***: faz a leitura e o tratamento de todos os ficheiros de dados, de modo a que possam ser inseridos nas estruturas de dados do *catálogo*.
- ***Catálogo***: reúne todas as principais funções e estruturas de dados necessárias para a resolução das queries.
- ***Parser de comandos***: faz a leitura e interpretação de todos os comandos de input de modo a executar a respetiva query.

2.2 Organização dos dados

2.2.1 Estruturas de dados

Como referido na secção **Arquitetura de Aplicação**, foi utilizado um catálogo para o armazenamento das principais estruturas de dados do projeto, sendo elas: **Hashtable de Users**, **Hashtable de Drivers**, **Array de Users**, **Array de Drivers**, **Array de Rides**.

Estas estruturas de dados foram todas implementadas recorrendo à biblioteca GLib-2.0, por uma questão de fácil usabilidade da biblioteca, diminuição de linhas de código e menor risco de erros de implementação. Ambas as estruturas (Hashtables e Arrays) foram implementadas utilizando as estruturas já existentes

nesta biblioteca: GHashTable, GPtrArray.

Existem também as subestruturas encapsuladas que compõem as estruturas referidas acima, sendo estas: **User**, **Driver** e **Ride**.

Um **User** contém todos os dados relevantes de um utilizador: username, nome, género, data de nascimento, data de criação da conta, método de pagamento, estado da conta, distância total percorrida, data da última viagem.

Um **Driver** contém todos os dados relevantes de um condutor: id, nome, data de nascimento, género, classe do carro, matrícula, cidade, data de criação da conta, avaliação média, número total de viagens, data da última viagem, estado da conta.

Um **Ride** contém todos os dados relevantes de uma viagem: id, data, id do condutor, username do utilizador, cidade, distância percorrida, avaliação de utilizador, avaliação de condutor, gorjeta, comentário.

2.2.2 Porquê estas estruturas?

A utilização destas estruturas deve-se à ambição de ter um tempo de execução baixo com grandes quantidades de dados. São vários os momentos em que precisamos de aceder rapidamente aos dados de um condutor ou utilizador sabendo o username ou o id, respetivamente. Deste modo foi decidido que Hashtables seriam as estruturas mais adequadas nesta situação.

Foi decidido também o uso de arrays pelo simples facto de serem estruturas de fácil e rápido acesso e simples ordenação. Para além destas vantagens era necessário o armazenamento das viagens e não era justificável o uso de uma hashtable.

3 Implementação

3.1 Datas

Para o tratamento das datas nos ficheiros, foi usado um algoritmo que converte uma data no número de dias a partir de uma data de referência, sendo que no nosso caso essa data é **1/1/1900**, sendo assim o dia **2/1/1900** corresponde ao número 1. Isto facilita trabalhar com datas e diminui o tamanho das estruturas de dados.

3.2 Query 1

Para a primeira query recorremos ao catálogo, mais especificamente às **hash tables** com os dados dos ficheiros **drivers.csv** e **users.csv** e o array com os dados do ficheiro **rides.csv**. A resolução desta query é bastante semelhante para drivers e para users, ambas envolvem procurar o respetivo driver ou user na hash table através do **id/username** fornecido no input e, caso a conta esteja ativa, obter o seu **nome**, **género** e **idade** através de funções definidas para tal. Se a conta estiver inativa a função retorna imediatamente **NULL**. Estando a conta ativa, depois da obtenção dos dados referidos, inicia-se um ciclo que percore a totalidade do **array das rides** e compara o **id/username** de input com o **id/username** de cada ride, sempre que esse campos forem iguais, os restantes valores necessários para a query são atualizados, sendo esses a **avaliação média** do user/driver, o seu **número total de viagens** e o **dinheiro gasto/auferido**. Por fim esses dados são convertidos para **string** e enviados como output no formato pedido para o ficheiro correspondente.

3.3 Query 2

Para a segunda query recorremos ao **GPtrArray** com os dados dos **drivers** que se encontra no catálogo e **ordenamos** esse array de acordo com os critérios pedidos na query, para tal foi feita uma função de comparação entre dois drivers que primeiro obtem a **avaliação média** de cada um, a **data da sua viagem mais recente** e os seus **ids**. São feitas comparações entre esses dados de modo a

ter um função que se enquadre nos requisitos da função de sort de um **GPtrArray** da **glib**. A execução da query passa por primeiro obter o input, ou seja o número (**N**) de drivers pedido, de seguida ordenar o array com a função da **glib** que segue os critérios estabelecidos na nossa função, obter os dados necessários para output de cada um dos (**N**) drivers e converter para string no formato correto, esses dados são enviados como output para o respetivo ficheiro mal fiquem disponíveis. Tendo em conta que o array fica ordenado por **ordem decrescente** de distância, os primeiros elementos terão sempre a maior avaliação média. Se a conta do driver estiver **inativa**, a função que produz o output retorna **NULL** em vez de uma string.

3.4 Query 3

Para a terceira query recorremos ao **GPtrArray** com os dados dos **users** que se encontra no catálogo e **ordenamos** esse array de acordo com os critérios pedidos na query, para tal foi feita uma função de comparação entre dois users que primeiro obtem a **distância total** que cada um viajou, a **data da viagem mais recente** de cada um e os seus **usernames**. São feitas comparações entre esses dados de modo a ter um função que se enquadre nos requisitos da função de sort de um **GPtrArray** da **glib**. A execução da query passa por primeiro obter o input, ou seja o número (**N**) de users pedido, de seguida ordenar o array com a função da **glib** que segue os critérios estabelecidos na nossa função, obter os dados necessários para output de cada um dos (**N**) users e converter para string, esses dados são enviados como output para o respetivo ficheiro mal fiquem disponíveis. Tendo em conta que o array fica ordenado por **ordem decrescente** de distância, os primeiros elementos terão sempre a maior distância viajada. Se a conta do user estiver **inativa**, a função que produz o output retorna **NULL** em vez de uma string.

4 Desempenho

Nesta primeira fase os outputs de qualquer uma das queries implementadas é o esperado e com um tempo de execução razoável entre 1,5 e 2 segundos com margem de erro menor que 0,5s.

5 Conclusão

Em geral a primeira fase correu bem, não houve grandes problemas mas ainda assim existem vários aspetos que precisamos de melhorar na próxima fase. Entre eles a otimização das queries, a implementação de testes de performance e correção, a criação de um módulo de estatíticas para diminuir o tamanho das estruturas de dados e o tempo de acesso aos dados e proporcionar uma melhor ligação entre essas estruturas.