Departamento de Informática
Escola de Engenharia
Universidade do Minho

Relatório Fase 2 (Final)

Laboratórios de Informática III

Licenciatura em Engenharia Informática

2022/2023

Grupo 53

Diana Teixeira - a97516

Flávia Araújo - a96587

Marta Ribeiro - a95408

05/02/2023

Índice

Introdução 1
Estruturas de dados utilizadas 2
Desenvolvimento da aplicação 3
  Modularidade e encapsulamento 3
Modos de execução 4
Queries 5
  Query 1 5
  Query 2 5
  Query 3 6
  Query 4 6
  Query 5 6
  Query 6 7
  Query 7 7
  Query 8 7
  Query 9 8
Conclusão 9

Introdução

No âmbito da unidade curricular de “Laboratórios de Informática III”, foi proposto o desenvolvimento de um sistema de organização de dados na área de viagens, sendo os dados dessas, bem como os dados dos condutores e utilizadores, introduzidos em estruturas de dados e colocados posteriormente em catálogos e organizados de maneira otimizada para permitir um acesso mais rápido e resultados mais eficientes. Para auxílio da estruturação de dados, utilizámos estruturas pertencentes à biblioteca GLib2 da GNOME para a manipulação das coleções de dados, bem como estruturas da nossa própria autoria.

Após a primeira fase do trabalho, e respetiva defesa, vamos trabalhar sobre o que nos é pedido para esta segunda fase, e tentar melhorar os pontos que não conseguimos antes.

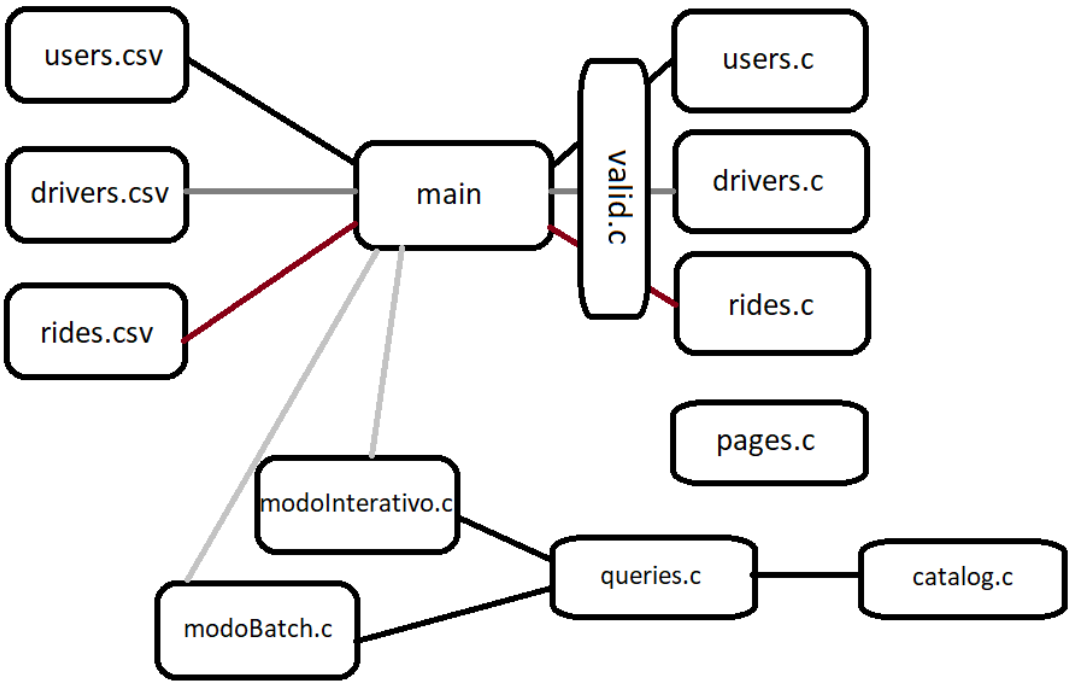
Estruturas de dados utilizadas

Durante a primeira fase, como forma de armazenar a informação proveniente dos ficheiros .csv, optamos inicialmente por utilizar estruturas de *arrays*. Isto porque são estruturas de dados que ocupam menos memória; só teríamos necessidade de fazer inserções nos *arrays* na fase inicial (para preencher com os dados); e existiria a possibilidade de fazer *lookups*.

No entanto, após algumas dificuldades, e como vimos que estávamos a ficar num impasse, recorremos à *glib2*, passando assim à utilização das *HashTables* para manipular as coleções de dados. Foram utilizadas tanto para armazenar a informação dos *users,* como dos *drivers* e das *rides*. Temos noção de que é algo que pode custar mais no tempo de execução, e, por isso, é um dos pontos que procuramos vir a melhorar para a segunda fase do projeto.

Na segunda fase, e após a defesa da primeira, decidimos manter as *HashTables*, pois cremos que seria a melhor estrutura para os nossos dados. Para a parte das *queries*, onde guardamos os dados para os poder ordenar, utilizamos estruturas de *arrays.*

Desenvolvimento da aplicação



Modularidade e encapsulamento

Como forma de modularmos o nosso código, separamos os ficheiros em ficheiros *.c* e *.h*.

Nos ficheiros *.c*, temos o *main.c* que é o programa principal que utiliza os restantes módulos, e é o único que tem acesso a toda a informação e todos os *includes* dos *header files* e librarias, como por exemplo, *string.h*. Os restantes (*array.c, buffer.c, catalog.c, date.c, drivers.c, queries.c, rides.c* e *useres.c*), têm no início apenas o include do *header file* correspondente e em alguns casos, as librarias que são necessárias.

Para cada ficheiro *.c*, temos o seu *header file (.h)* correspondente (excetuando o *main.c*), onde estão as assinaturas das funções dos *.c* e estruturas de dados; e também os *includes* necessários para aquele módulo.

Posto isto, garantimos, então, que seguimos as regras do encapsulamento, mantendo todos os dados e funções de cada módulo, desconhecidas e imutáveis para os módulos que não necessitam de lhes ter acesso; tornando os dados viáveis, uma vez que se fornecem apenas cópias nos casos em que estes são necessários.

Evitamos o uso de variáveis globais por ser uma abordagem perigosa para o programa.

Modos de execução

Para esta segunda fase, era necessária a implementação do modo *batch* (que já foi feita para a primeira fase) e o modo interativo.

Para escolher qual o modo em que queremos executar as *queries,* o programa pede inicialmente para inserirmos o modo que pretendemos utilizar.

O *batch* consiste em receber dois parâmetros: o primeiro com o caminho para a pasta onde estão os ficheiros de entrada; e o segundo com o caminho para o ficheiro que contém a lista de comandos para as *queries*, no caso, o ficheiro com o nome *commands.txt*. Neste, cada linha tem o número da *query* que se pretende fazer, com os parâmetros necessários; e após a execução de cada linha, o resultado de cada linha, é guardado num ficheiro *commandN_output.txt* (na pasta Resultados), onde N corresponde ao número da linha no ficheiro *commands.txt*.

No interativo, tal como o nome indica, vamos interagindo com o terminal, onde inicialmente pede para inserirmos a *query* que pretendemos o resultado. Após isso, vai pedindo os parâmetros que são necessários à realização dessa mesma *query*. Para a apresentação dos resultados, temos um menu onde é feita a **paginação**.

Esta paginação aparece em forma de menu, onde podemos escolher se queremos ver a página atual, a seguinte, ou a anterior. Ou eventualmente, sair deste menu.

Queries

Da primeira fase, tínhamos 1/3 das *queries* (4,5 e 6) totalmente funcionais, não alterando nada nestas. Para esta fase, implementamos as restantes.

Query 1

A *query* lista o resumo do perfil de um utilizador ou condutor.

Esta *query* é chamada através do comando *1 <ID>* e se for um utilizador, apresenta *nome;genero;idade;avaliacao_media;numero_viagens;total_gasto;* se for um condutor, apresenta *nome;genero;idade;avaliacao_media;numero_viagens;total_auferido.*

No modo interativo, o programa pergunta se pretendemos listar um utilizador ou um condutor.

No *batch*, é feito antes um teste. Se <ID> for um número, corresponde ao ID de um condutor, e é utilizada a função *q1d_response*. Se não for, sabemos que é o *username* de um utilizador, sendo a função *q1u_response.*

- Na função *q1u_response,* vamos à *Hash* dos *users* procurar o *user* que foi pedido na *query*. Após isso, vamos buscar o seu *nome, género* e *idade* (que é calculada).

Iterando pelas *rides*, vai procurando as que têm o *username* correspondente ao pedido.

De cada vez que for encontrada uma *ride*, é incrementado um contador que vai dar origem ao *numero_viagens* e também vai ser utilizado para o cálculo da média. Procura na *Hash* a avaliação dessa viagem, e é incrementado noutra variável. Para calcular a média da avaliação, é dividida a avaliação total contada, pelo contador do número de *rides* encontradas. Por fim, conforme a classe do carro, o preço da viagem é calculado, somando a gorjeta que foi dada (se tiver sido dada), e é incrementado numa variável.

- Na função *q1d_response*, a implementação é igual, mas em vez de ir à *Hash* dos *users* procurar o ID, vamos à *Hash* dos *drivers* encontrá-lo.

Query 2

A *query* lista os N condutores com maior avaliação média.

Esta *query* é chamada através do comando *2 <N>* e apresenta N linhas com os campos *id;nome;avaliacao_media*.

Iterando pelos *drivers*, vai aferindo o ID e o nome do condutor, informação esta que vai sendo guardada numa *struct* de *arrays* com as avaliações, o número de viagens e a data da última viagem a 0.

De seguida, esses valores vão ser atualizados pelos que vêm das *rides*. Por fim, os resultados são ordenados por ordem decrescente de avaliação média.

Query 3

A *query* lista os N utilizadores com maior distância viajada.

Esta *query* é chamada através do comando *3 <N>* e apresenta N linhas com os campos *username;nome;distancia_total*.

Iterando pelos *users*, vai aferindo *username* e nome de utilizador. Estes valores são guardados numa *struct* de *arrays,* com distância e data da última viagem realizada a 0. De seguida, estes valores são atualizados pela informação proveniente das *rides*. Os valores são ordenados por ordem decrestende de distância viajada.

Query 4

A *query* 4 calcula o preço médio das viagens (sem considerar gorjetas) numa determinada cidade.

Esta *query* é chamada através do comando *4 <city>* e apresenta apenas o *preco_medio* calculado como *double*, com 3 casas decimais.

Iterando pelas *rides*, aferindo o id do condutor; vai à *Hash* dos *drivers* encontrar o condutor correspondente, ver a classe do carro; e também a distância e cidade (*rides*).

Após isto, testa-se se a cidade na qual o condutor se deslocou naquela viagem, é igual à cidade passada como argumento na *query*. Se sim, existe um contador que é incrementado e o preço dessa viagem é calculado conforme a classe do carro (*Basic, Green, Premium*) e adicionado ao preço das viagens anteriores. Se não, continua a iterar pelas *rides*.

No final, para obter a média, é dividido o preço total de todas as viagens naquela cidade pelo contador.

Query 5

A *query* 5 calcula o preço médio das viagens (sem considerar gorjetas) num determinado intervalo de tempo.

Esta *query* é chamada através do comando *5 <data A> <data B>* e apresenta apenas o *preco_medio* calculado como *double*, com 3 casas decimais.

Iterando pelas *rides*, aferindo o id do condutor; vai à *Hash* dos *drivers* encontrar o condutor correspondente, ver a classe do carro; e também a distância e a data (*rides*).

Após isto, testa-se se a data na qual o condutor se deslocou naquela viagem, está compreendida entre as datas passadas como argumento na *query*. Se sim, existe um contador que é incrementado e o preço dessa viagem é calculado conforme a classe do carro (*Basic, Green, Premium*) e adicionado ao preço das viagens anteriores. Se não, continua a iterar pelas *rides*.

No final, para obter a média, é dividido o preço total de todas as viagens naquela cidade pelo contador.

Query 6

A *query 6* calcula a distância média das viagens, numa cidade, num determinado intervalo de tempo.

Esta *query* é chamada através do comando *6 <city> <data A> <data B>* e apresenta apenas a *distancia_media* calculado como *double*, com 3 casas decimais.

Iterando pelas *rides,* são aferidas a data e a cidade.

Após isto, testa-se se a cidade na qual o condutor se deslocou naquela viagem, é igual à passada como argumento na *query;* e depois se a data daquela viagem está entre os valores passados também como argumento na *query*. Se sim, existe um contador que é incrementado e a distância dessa viagem é adicionada às distâncias das viagens anteriores. Se não, continua a iterar pelas *rides*.

No final, para obter a média, é dividida a distância total de todas as viagens naquela cidade entre aquelas datas, pelo contador.

Query 7

A *query* lista os top N condutores numa determinada cidade, ordenados pela avaliação média do condutor.

Esta *query* é chamada através do comando *7 <N> <city>* e apresenta *id;nome;avaliacao_media.*

Iterando pelos *drivers,* são aferidos o ID e o nome do condutor, que são guardados numa *struct* de *arrays* com os valores das avaliações e *tips* a 0. Posteriormente, itera pelas *rides*, para preencher estes valores. De seguida, são ordenados pela avaliação média do conditor.

No final, é escrito no ficheiro de *output* o ID e nome do condutor e respetiva avaliação média.

Query 8

A *query 8* apresenta a lista das viagens nas quais o utilizador e o condutor são do género passado como parâmetro, e têm perfis com X ou mais anos; ordenando-a das contas mais antigas para as mais recentes.

Esta *query* é chamada através do comando *8 <gender> <X>* e apresenta o *id_condutor, nome_condutor, username_utilizador* e *nome_utilizador.*

Iterando pelas *rides,* é aferido o username do condutor e o id do utilizador; bem como depois os géneros de cada um e os anos de conta.

Após isto, testa-se se os géneros do utilizador e condutor são iguais ao género passado como argumento na *query*. Se sim, afere-se o nome do condutor, o nome do utilizador e o *username* do utilizador. Estes valores são inseridos *struct* de *arrays* que depois vai ser ordenada por anos de conta, de forma decrescente. No final, é escrito no ficheiro de *output* os id's nomes e *username* do condutor e utilizador. Se não, continua a iterar pelas *rides*.

Query 9

A *query 9* apresenta a lista das viagens nas quais o passageiro deu gorjeta, num intervalo de tempo, ordenando-a por ordem de distância percorrida.

Esta *query* é chamada através do comando *9 <data A> <data B>* e apresenta o *id_viagem, data_viagem, distancia, cidade* e *valor_gorjeta.*

Iterando pelas *rides,* é aferida a data da viagem.

Após isto, testa-se se a data daquela viagem está entre as datas passadas como argumento na *query*. Se sim, afere-se se foi dada uma gorjeta na viagem. No caso de haver gorjeta, estes valores são inseridos numa *struct* de *arrays* que depois vai ser ordenada por ordem decrescente de distância, e é escrito no ficheiro de *output* o id, distância, cidade e a data. Se não, continua a iterar pelas *rides*.

Conclusão

Após a realização deste projeto, apesar dos desafios que fomos encontrando, demos o nosso melhor para os tentar ultrapassar.

Sabemos que é um trabalho que está sempre sujeito a melhorias, mas sentimos que os nossos objetivos foram maioritariamente cumpridos.