A96807 – Rafael Conde Peixoto

A97941 – Diogo Filipe Oliveira da Silva

Índice

Conteúdo

1 Introdução

Nesta primeira fase do trabalho foi desenvolvido um programa tendo em conta os principais conceitos de encapsulamento e modularidade.

Este trabalho prático resume-se em carregar três ficheiros de dados para memória, para futuramente a informação ser processada (parsing), com o objetivo de responder a diversas questões. Sendo divididas as estruturas de dados e as respetivas funções que atuam nessas estruturas, por diversos ficheiros, ficando mais organizado, tentando ao máximo respeitar as regras de encapsulamento e modularidade.

2 Estruturas de dados

Primeiramente, definimos três structs com as correspondentes instâncias de cada tipo de dados (users, rides, drivers).

Para guardar todos os conjuntos de dados, provenientes dos ".csv's" implementamos três tabelas de Hash para cada tipo de dados (users, rides, drivers).

Nós decidimos implementar tabelas de Hash devido á sua facilidade de compreensão e devido aos acessos ao array de tempo contante Ȯ(1).

USERS

A tabela de Hash do modulo users é uma tabela closed addressing/ chaining de tamanho 100003(nº primo) de forma a resolver possíveis colisões que possam vir a ocorrer nessa posição. O campo username é usado como chave sendo adicionado à cabeça da lista ligada da posição atribuída pela função hash.

```
struct user {
        ...
        struct user * next;
};
```

RIDES E DRIVERS

A tabela de Hash do modulo rides é uma tabela open adressing de tamanho 1000000. O campo id é usado como chave sendo adicionado à posição atribuída pela função hash.

A tabela de Hash do modulo drivers é uma tabela open adressing de tamanho 10000. O campo id é usado como chave sendo adicionado à posição atribuída pela função hash.

Como os id's em ambos os casos são diferentes dentro dos seus módulos não existem colisões.

Parsing

Através da função *parsing* os dados dos .csv's são transferidos para as diferentes estruturas de dados (Tabelas de Hash), HashTableRides (rides.csv), HashTableDrivers (drivers.csv) e HashTableUsers(users.csv) para depois serem usados nas diferentes queries.

De seguida através da função *parsingInput* cada linha do input.txt será lida individualmente e consequentemente será executada a respetiva instrução (query) dando retorno do resultado dentro de um ficheiro apelidado de "commandX_output.txt" onde X é o nº da instrução. Caso a instrução (query) não esteja implementada/ ou o valor seja nulo ele cria um ficheiro vazio.

Queries Implementadas

Query 1

A query 1 recebe um id (char) que tanto pode ser de um user como de um driver. No caso de ser um driver a query vai verificar se a conta está ativa (*lookupStatusDriver*), na hashTableDrivers (Tabela de Hash Drivers). Caso a conta não esteja ativa retorna um Null. Caso contrário dará retorno na forma: nome; genero; idade; avaliacao_media; numero_viagens; total_aferido.

O nome, género, idade são obtidos respetivamente pelas funções *lookupNomeDriver*, *lookupGeneroDriver* e *lookupIdadeDriver* na hashTableDrivers (Tablela de Hash dos Drivers), já a avaliação_media, numero_viagens são calculados pela função *lookupAvalNViagemTotAufDrivers* que precorre a hashTableRides.

No caso de ser um user a query vai verificar se a conta está ativa (*lookupStatusUser*), na hashTableUsers (Tabela de Hash Users). Caso a conta não esteja ativa retorna um Null. Caso contrário dará retorno na forma: nome; genero; idade; avaliacao_media; numero_viagens; total_gasto.

O nome, género, idade são obtidos respetivamente pelas funções *lookupNomeUser*, *lookupGeneroUser* e *lookupIdadeUser* na hashTableUser (Tablela de Hash dos *Users*), já a avaliação_media, numero_viagens são calculados pela função L*ookupAvalNViagemTotGastoUsers* que precorre a hashTableRides.

Query 4

A query 4 recebe o nome de uma cidade (char) e calcula o preço médio das viagens para isso vai percorrer a hashTableRides comparando o nome da cidade em cada posição e caso encontre uma igualdade, vai pegar o id do driver (*lookupDriverRides*) e a distância (*lookupDistanceRides*) percorrida nessa viagem e calcula o preço da viagem (*precoViagem*) que vai ser adicionado á variável "valor" e incrementa a variável "divide". No final dá retorno á razão entre as variáveis "valor" e "divide".

Query 5

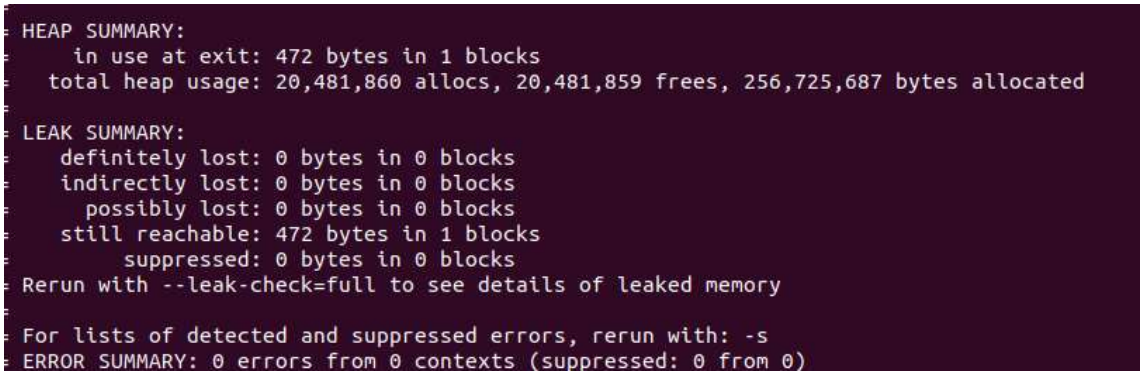
A query 5 recebe duas datas (dataInicio e dataFim) que estão na forma de dd/mm/yyyy (char's) e calcula o preço médio das viagens no intervalo de tempo definido pelas duas datas para isso vai percorrer a hashTableRides comparando a data da viagem verificando se a data encontrasse dentro desse intervalo de tempo. Na eventualidade de pertencer ao intervalo vai pegar o id do driver (*lookupDriverRides*) e a distância (*lookupDistanceRides*) percorrida nessa viagem e calcula o preço da viagem (*precoViagem*) que vai ser adicionado á variável "precoAtual" e incrementa a variável "divide". No final dá retorno da razão entre as variáveis "precoAtual" e "divide".

Query 6

A query 6 recebe o nome de uma cidade (char) e duas datas (dataInicio e dataFim) que estão na forma de dd/mm/yyyy (char's) e calcula a distância média percorrida nessa cidade no intervalo de tempo definido pelas duas datas para isso vai percorrer a hashTableRides comparando a cidade caso encontre uma correspondência vai verificar se pertence ao intervalo de tempo, se porventura pertencer vai buscar a distância percorrida nessa viagem (*lookupDistanceRides*) e vai ser adicionada à variável "distanciaAtual" e incrementa a variável "divide". No final dá retorno da razão entre as variáveis "distanciaAtual" e "divide".

Medição de desempenho

Durante a execução do programa são realizados cerca de 20.481.860 de allocs 20.481.859 free's e cerca de 256Mb de memoria alocada, demorando em média 3,15s.



Conclusão

Nós estamos satisfeitos com a estrutura de dados que utilizamos, mas durante o desenvolvimento do hashTableUsers sentimos que podíamos arranjar uma função de hash mais eficiente que provavelmente, pode obrigar a certas alterações na estratégia na 2º fase.

Em termos desempenho o nosso programa executa num tempo média aceitável mesmo que as queries que estão definidas não sejam muito complexas.

Concluindo, durante a realização desta primeira fase descobrimos várias novas estruturas de dados, melhor gestão de memoria e novos conceitos como a modularidade e encapsulamento.