Relatório Fase 1 do Projeto

Laboratórios de informática III

22/11/2022

Gustavo Barros - a100656
Pedro Sousa - a100823
Pedro Viana - a100701
Grupo 15

Índice:

Introdução:

Este projeto foi realizado no âmbito da cadeira de LI3 do 2º ano do curso de LEI, tendo como principais objetivos a consolidação dos nossos conhecimentos da linguagem C, aplicação dos conceitos de modularidade e encapsulamento, utilização de estruturas dinâmicas de dados, ferramentas de compilação e *linkagem*, e testagem funcional e de desempenho do código.

Para controlo de versões e hospedagem de código é utilizado um repositório na plataforma GitHub.

O contexto do projeto consiste em desenvolver uma aplicação, que deverá efetuar o tratamento de uma grande quantidade de dados de uma empresa de transportes guardados em três ficheiros de texto (informações sobre 100k utilizadores, 10k condutores e 1M viagens efetuadas) estes dados serão então manipulados e armazenados em módulos num catálogo que servirá de base de dados para a implementação de 9 *queries*. A aplicação deverá assumir dois modos de execução (*batch* e interativo).

No modo *batch* a aplicação receberá como *input* uma diretoria contendo os três ficheiros de texto (“users.csv”, “drivers.csv” e “rides.csv”) e uma diretoria contendo um ficheiro de texto com as queries a ser executadas pela aplicação (“commands.txt”). O resultado de cada query deverá então ser armazenado num ficheiro (“commandX_output.txt”) e armazenado numa pasta "Resultados".

Este relatório aborda apenas o desenvolvimento da primeira fase do projeto, sendo que na mesma é nos pedida a realização de 3 *queries* funcionais em que se assume o modo *batch* da aplicação.

Estruturas de dados:

Foram nos fornecidos 3 ficheiros com dados para trabalharmos e manipularmos, nomeadamente: “drivers.csv”, “users.csv” e “rides.csv”.

Para facilitar a manipulação dos dados dos ficheiros foram criadas *structs* para cada tipo de dados, respetivamente as *structs* “drivers”, “users” e “rides”.

Criámos uma *struct* aparte para a manipulação da data, “date”.

Quanto às *queries,* foram criadas *structs* para cada uma em particular, e uma *struct* geral para ajudar na sua execução.

Foi utilizado ainda um catálogo para disponibilizar os dados para as funções, o qual tem também uma *struct* própria “catalog”.

Parsing:

A aplicação começa por criar uma base de dados (que o grupo decidiu chamar catálogo) através da função “catalog_new”, este catálogo consiste numa *struct* composta por três apontadores para cada um dos ficheiros de input (“users”, “drivers” e “rides”), um apontador para o ficheiro que contém as *queries* a ser executadas, uma *GHashTable* (*hashtable* da biblioteca “GLib”) que serve de módulo para armazenar os dados de cada *user* e utiliza o *username* como *key*, uma outra *GHashTable* com a mesma funcionalidade mas de cada *driver*, utilizando o *id* dos mesmos como *key*, uma *GList* (lista ligada da biblioteca “GLib”) para armazenar os dados de cada *ride* e uma outra *GList* para armazenar todas as *queries* que serão executadas.

Ao criar o catálogo é necessário fornecer os 4 apontadores dos *FILES* (previamente associados ao texto de *input*). Todas as outras estruturas de dados começam vazias (*hashtables* e listas ligadas).

A razão pela qual o grupo optou por uma *Hashtable* para armazenar a informação de cada *user* e de cada *driver* foi porque desta forma é muito mais fácil procurar um determinado indivíduo através da sua *key*. Já nas *rides*, concluímos que a maioria das *queries* pede informações presentes numa grande quantidade de *rides*, logo seria muito mais eficiente utilizar uma lista ligada, visto que é percorrida com maior rapidez. O mesmo pensamento foi utilizado para armazenar as *queries*.

Após criar a estrutura da base de dados, o próximo passo seria encher a base de dados com a informação presente nos ficheiros de *input*. Para isso é utilizada a função "catalog_load" que é responsável por fazer o *parsing* dos três ficheiros de *input*. O *parsing* é feito de forma bastante simples, dividindo cada linha dos ficheiros (através dos caracteres ‘;’) num

array de *tokens* mais pequenos. O *array* terá um tamanho variável dependendo de qual dos 3 ficheiros está a ser tratado, este tamanho será passado como argumento à função *parser*.

Cada *array* é então associado a uma *struct(user, driver ou ride)* e cada parâmetro da mesma é preenchido com o conteúdo do *array*. No caso dos *users* e dos *drivers* a *struct* é inserida na respetiva *hashtable*. Já nas *rides*, a *struct* é colocada no final da lista ligada.

Fica apenas a faltar o *parsing* do *file* que contém as queries a ser executadas. A função "parse_queries" é responsável por isto. A estratégia do grupo foi definir uma *struct* para armazenar cada *query*, esta *struct* seria composta por um inteiro que representaria o seu tipo, e por um *void array* que guardaria os argumentos da *query*.

Com este formato, cada linha do ficheiro das *queries* é então associada a uma *struct* e seguidamente introduzido à lista ligada presente no catálogo.

Tendo então a base de dados carregada, temos o necessário para executar as queries.

Queries:

Para cada *query* criámos uma *struct* diferente para armazenar as estatísticas necessárias para resolver a query.

Query 1:

Esta *query* tem como objetivo listar um resumo de um perfil através do seu identificador (fornecido como argumento) este resumo poderá assumir duas formas, pois o identificador poderá pertencer a um *user* ou a um *driver*.

Portanto, o grupo resolveu armazenar o identificador do perfil, a soma de todas as avaliações, o total de *rides* efetuadas, e o valor total gasto/auferido.

O identificador é então associado a um *user* ou um *driver* dependendo do seu formato. Este *user/driver* tem o seu status verificado na respectiva *hashtable* para garantir que a sua conta está ativa.

Para obter os restantes dados necessários, é efetuada uma única travessia da lista ligada que contém a totalidade das *rides*, incrementando os parâmetros da *struct* que armazena as estatísticas caso uma *ride* do identificador seja detectada.

O resultado final é então impresso e guardado no formato pedido.

Query 2:

Nesta *query* é pedida alguma informação sobre os "*N*" condutores com melhor avaliação média (nome, idade e avaliação média). Em caso de empate, seguem-se os critérios de desempate: viagem mais recente, e *id* do condutor (crescentemente).

O grupo resolveu armazenar o *id*, nome, *score* total, número de *rides* efetuadas, e a data da *ride* mais recente numa *struct*.

É então criada uma *struct* para todos os *drivers* com conta ativa e preenchidos os parâmetros do *id* e nome. Cada *struct* é então inserida numa lista ligada. De seguida é efetuada uma única travessia da lista ligada das *rides* e recolhidas as estatísticas necessárias para preencher os restantes parâmetros da *struct*.

No final é utilizada uma função de ordenação (obedecendo aos critérios acima referidos) para ordenar a lista. A informação dos "N" condutores do topo é então impressa e guardada no formato pedido.

Query 3*:*

Esta *query* é semelhante à *query* anterior, sendo que a única diferença é que ao invés de retornar os “N” condutores com melhor avaliação média, teria de retornar os “N” utilizadores com maior distância viajada. Portanto o grupo decidiu adotar uma estratégia muito semelhante.

Armazenando o *username*, nome, *score* total, número de *rides* efetuadas, a data da *ride* mais recente e o total de km viajados numa *struct*.

Em caso de empate os critérios de desempate são: viagem mais recente e ordem alfabética do *username.*

Para obter o resultado da *query* é criada então uma *struct* para todos os utilizadores com conta ativa, inserindo-a numa lista ligada, é efetuada uma travessia da lista das *rides*, incrementando todas as estatísticas necessárias.

Uma função de ordenação (que obedece aos critérios de desempate) é então utilizada para ordenar a lista resultante. A informação dos “N” utilizadores do topo da lista é então impressa e guardada no formato pedido.

Conclusão:

Nesta primeira fase do projeto, tivemos dificuldade em aplicar modularidade e encapsulamento na totalidade, presenciamos alguns erros *segmentation faults*, que foram corrigidos posteriormente. O trabalho fluiu com um bom espírito de equipa e cooperação, desta forma conseguimos realizar as 3 *queries* propostas com um tempo curto de execução, o que consideramos um bom resultado.