Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III

Fase 1

Grupo 58

Ano letivo 2024/2025

Samuel Gibson Da Cunha Figueiredo Lobato - A106907
Lucas Rafael da Cunha Franco Robertson - A89467
Marco Rocha Ferreira - A106857

Conteúdo

Fase 1 ..... **1**
Grupo 58 1

Introdução **3**

Sistema **4**
Diagrama de arquitetura 4
Gestores 4
Entidades 5
Queries 5
IO 6
Utils 6
Testagem 6

Discussão **8**
Configurações dos computadores 8
Análise de desempenho 8
Funcionamento das queries 9
Query 1 10
Query 2 10
Query 3 10

Conclusão **12**

Introdução

Este relatório configura-se como um material de apoio para o projeto realizado no âmbito da UC Laboratórios de Informática III, pelo grupo 58. O intuito deste trabalho resume-se a:

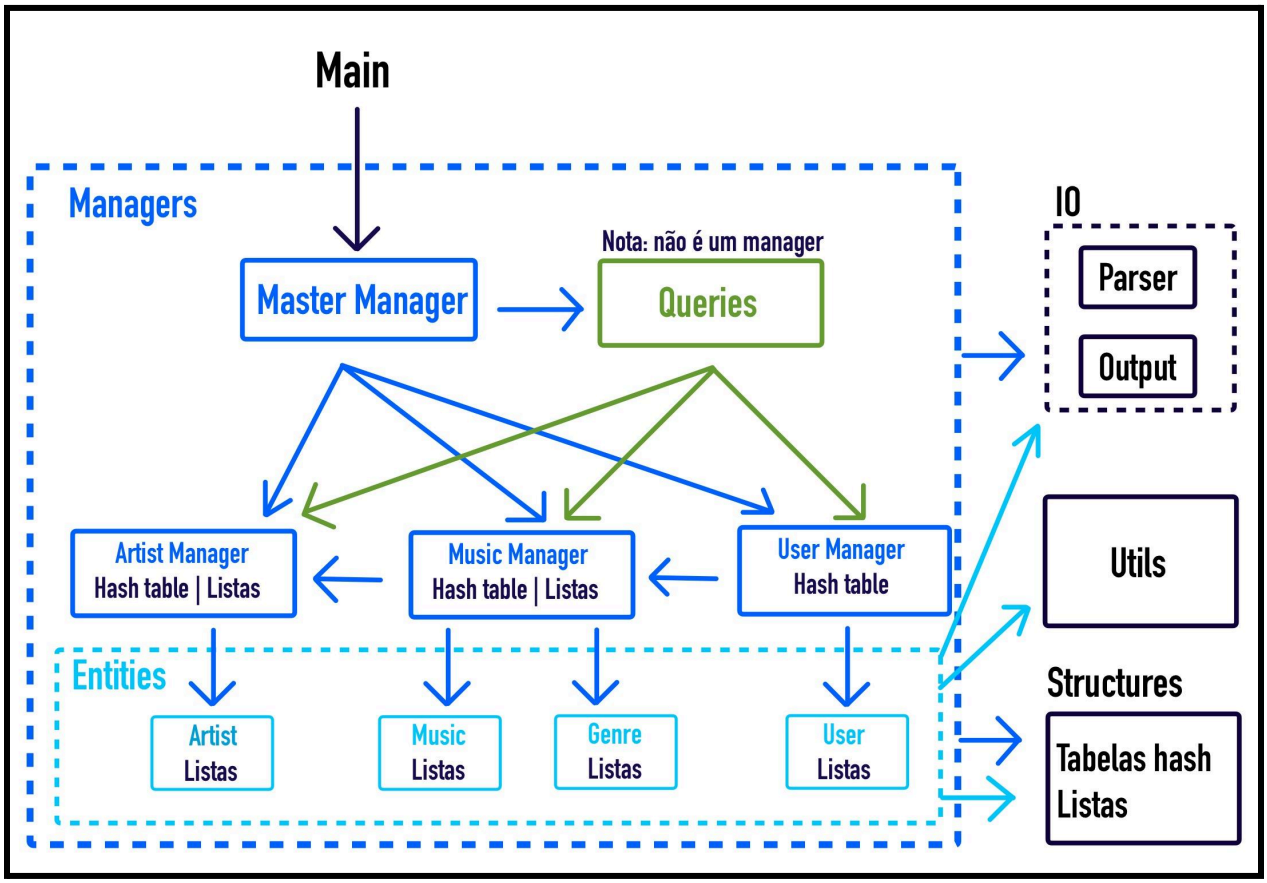
- Desenvolver um conhecimento mais profundo acerca da linguagem de programação C, com ênfase na aplicação do conceito de modularidade no código e no uso de estratégias de encapsulamento;

- Aprender a usar ferramentas que permitem melhorar a gestão de memória, com o principal objetivo de evitar *memory leaks*, que tornam o programa inseguro e instável.

A manipulação de forma organizada de uma quantidade razoável de dados num curto período de tempo e a utilização de uma quantidade de memória bastante adequada são características do nosso programa que merecem ser destacadas, e que serão exploradas nas secções a seguir.

Sistema

Diagrama de arquitetura



Gestores

- Master Manager: comporta os três gestores abaixo descritos e controla a execução do programa;

- Artist Manager: Inclui a tabela de hash dos artistas e um array dos mesmos ordenado segundo a discografia total de cada um;

- Music Manager: Inclui a tabela de hash das músicas e um array dos géneros musicais;

- User Manager: Inclui a tabela de hash dos utilizadores.

Cada um dos três gestores de entidades contém a lógica de armazenamento das mesmas, sendo que a validação lógica é executada aqui (só após ser validada é que uma entidade é efetivamente armazenada). Mais concretamente, o Music Manager valida a lista de artistas de cada música e o User Manager valida as *liked musics* de cada utilizador.

Nota: as três tabelas de hash estão organizadas segundo os IDs das entidades correspondentes.

Entidades

- Artists;
- Musics;
- Genre;
- User.

Cada entidade é instanciável e contém os dados presentes nos ficheiros CSV. No que toca a funções, todos estes módulos disponibilizam *getters* e *setters* para que o acesso aos dados seja controlado, i.e., para que outros módulos não alterem os dados das entidades. Por fim, cada módulo contém ainda as funções de validação sintática relativas à sua entidade.

Queries

Este módulo tem uma estrutura geral que se ramifica em cada tipo de query e é única, ou seja, dependendo do tipo de query a ser processado, os campos são atualizados com a informação relevante dessa query. Depois, na execução de uma nova query, os campos anteriores são sinalizados como inválidos, e assim sucessivamente.

IO

- Parser: módulo que serve para leitura de ficheiros, instanciando estruturas Parser que contêm um apontador para um ficheiro e um campo que indica o número de bytes lidos no último acesso a esse ficheiro. Este último campo tem como propósito verificar se foi, efetivamente, lida alguma coisa e também navegar no ficheiro (recorrendo à função fseek).

- Output: Contrariamente, serve para escrita, quer para o *standard output*, quer para ficheiros. Neste último caso, são instanciadas estruturas Output que contêm um apontador para o ficheiro (criando-o caso não exista).

Utils

De momento, o módulo Utils contém apenas uma função usada por vários outros módulos para validar os campos de lista dos ficheiros CSV.

Testagem

Este módulo contém funções específicas ao programa de testes, cujo objetivo é comparar os resultados obtidos com os que eram esperados, imprimindo para o terminal os ficheiros e a linha dentro deste onde ocorreram os erros.

Discussão

Configurações dos computadores

	PC 1	PC 2	PC 3
Processador	Intel® Core™ i7-1165G7	Intel® Core™ i7-1355U	AMD Ryzen 7 5700U
Frêquencia do CPU	4.7 GHz	5 GHz	4.3 GHz
Capacidade da cache	12 MB	12 MB	8 MB
Números de *cores*	4	10	8
Números de *threads*	8	12	16

Análise de desempenho

	Q 1[1] (ms)	Q 2[1] (ms)	Q 3[1] (ms)	Armazenamento(s)[2]	Resposta a Queries(s)[3]	Free(s)[4]	Total(s)[5]
PC 1	0.001035	0.007540	0.001926	2.430732	0.001608	0.149173	2.581514
PC 2	0.00072	0.004921	0.001134	1.741639	0.001403	0.127251	1.870294
PC 3	0.002820	0.016153	0.003418	3.120427	0.006182	0.203786	3.229954

[1] As colunas “Q1”, “Q2” e “Q3”, fazem referência ao tempo médio necessário para a execução das queries.

[2] Tempo necessário para o armazenamento, validação e organização dos dados.

[3] Tempo total da lógica de resposta.

[4] Tempo necessário para a libertação da memória usada.

[5] Tempo total necessário para a execução do programa.

Os resultados da tabela acima foram obtidos a partir de 5 medições, sendo o resultado apresentado uma média. Além disso, todas os testes foram efetuados em modo de desempenho, com o mínimo de processos a correr ao mesmo tempo e utilizando o *dataset* sem erros.

Como era de esperar, os melhores resultados foram obtidos pelo PC 2, uma vez que possui a maior frequência de relógio, a maior cache e maior número de *cores*. Os segundos melhores resultados, bastante semelhantes ao PC 2, pertencem ao PC 1, que possui características bastante semelhantes, sendo que a única diferença significativa está no número de *cores*. Por fim, o \, que possui características inferiores aos outros dois, obteve os piores resultados.

Analisando a tabela com algum cuidado, é possível notar que o tempo de armazenamento é o mais significativo, sendo bastante próximo ao tempo total. Isto deve-se ao facto de termos optado por fazer a organização dos dados imediatamente após a validação. Na prática, quando é lida uma linha de dados, a informação dessa linha é validada. Se a validação for satisfeita, a informação é armazenada e, além disso, outros dados relevantes deriváveis a partir da linha do ficheiro são calculados. São eles:

- Idade de um utilizador, que é calculada após a sua validação;
- Likes por género, que vão sendo acumulados a cada utilizador validado;
- Duração de uma música, que é adicionada à discografia total dos seus artistas.

Além destes dados, é ainda feita uma ordenação do array de artistas segundo a sua discografia total.

Com este armazenamento concluído, as funções de resposta às queries passam a ter toda a informação que precisam “à disposição”. Desta forma, os tempos de execução das queries são bastante reduzidos, porque o “trabalho pesado” é efetuado aquando do armazenamento e validação.

Funcionamento das queries

Query 1

A query 1 dá-nos o id de um utilizador para devolvermos certas informações dele ou uma linha vazia caso ele não exista, como nós guardamos os utilizadores numa hash table cujas keys são os seus ids, em inteiros, a verificação de se o utilizador está presente lá ou não é rapidíssima pois é só dar lookup do id na tabela, e se ele existir obter as suas informações também o é, pois o utilizador é o data relacionado a essa key.

Query 2

A query 2 consiste em, dado um número natural não negativo N, fornecer os N artistas com maior discografia. Ademais, é possível que se exija o país em específico dos artistas presentes na resposta.

Como ao guardar as músicas válidas adicionamos logo a sua duração aos seus artistas, e no final de lermos o ficheiro de músicas inteiro ordenamos o array de artistas por duração, no momento de responder à query só temos que devolver as primeiras posições desse array.

No caso de nos darem também o país, nós percorremos o array e vamos comparando o país pedido com o país de cada artista. Tendo de percorrer toda a lista se não houverem N artistas do dado país.

Query 3

Não obstante, é formulada uma query que requer do código a capacidade de listar os gêneros mais curtidos numa determinada faixa etária de usuários.

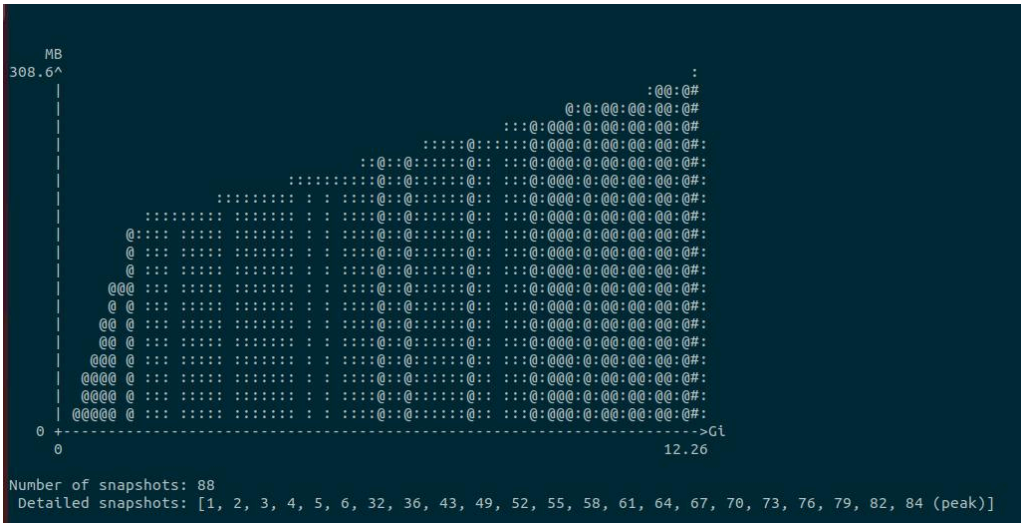
Como citado na resolução da query, durante a leitura dos usuários uma lista de gêneros é construída. A mesma contém internamente estruturas que por sua vez detém uma outra lista, dessa vez com 150 “espaços”, tendo cada estrutura um gênero real associado. Numa determinada estrutura, cada posição dentro da sua lista compreende o número de curtidas que o seu gênero teve de usuários com a idade igual ao índice que aquele posição corresponde. Por exemplo, o primeiro elemento da lista de “Rock” denota o quantidade de curtidas que “Rock” recebeu de pessoas com zero anos de idade.

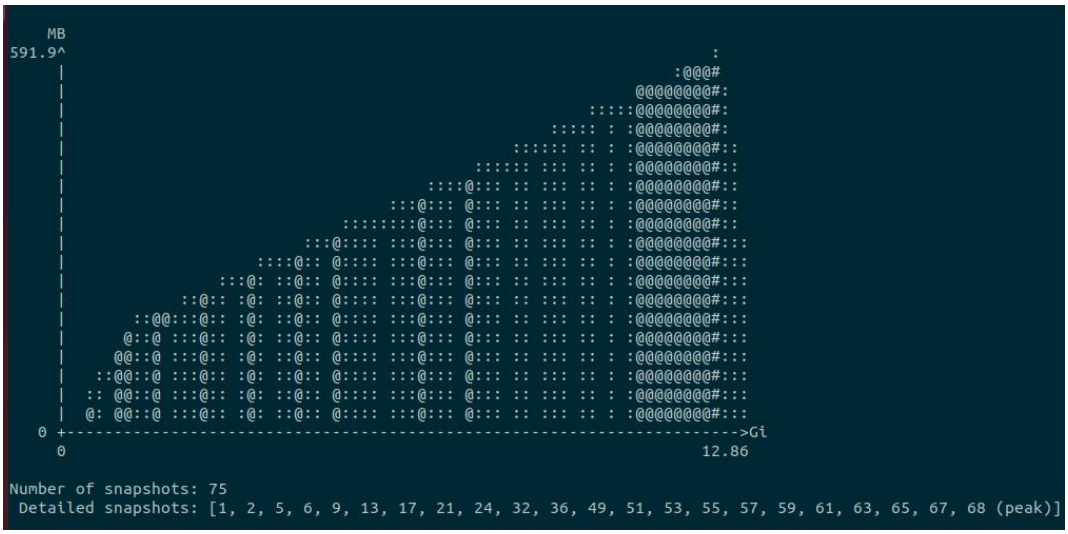
Elaborada a lista, esta será ordenada com base em uma diferença de contagem acumulada associada ao intervalo que a faixa etária delimita, ordenação a qual não será abordada com detalhes neste relatório. Em caso de empate de contagem, os gêneros são organizados em ordem alfabética.

Por último, todos os elementos da lista ordenada serão evidenciados.

Transição de Listas Ligadas para Arrays

A determinada altura do desenvolvimento do projeto, reparamos que estávamos com uma utilização de memória bastante acima do que era expectável. Portanto, após alguma investigação, concluímos que alguns campos para os quais estávamos a usar Listas Ligadas podiam mudar para Arrays, uma vez que não tirávamos qualquer partido das vantagens das listas. Abaixo apresentam-se dois gráficos de utilização de memória em função das instruções executadas, evidenciando a maior eficiência dos arrays para o nosso caso particular.





Conclusão

Após muitas horas de trabalho, conseguimos arquitetar um algoritmo o qual, embora não possua uma interface de usuário, por muito pouco não se configura um trabalho profissional. Um código capaz de operar uma imensa quantidade de dados em um tempo relativamente curto e, graças ao encapsulamento, sem deturpar a informação original. Ademais, o produto final não admite, na medida do possível, vazamentos de memória, porquanto para a execução do programa há funções predefinidas de bibliotecas externas às quais não temos acesso a sua algoritmia (e por conseguinte não conseguimos neutralizar os erros de alocação de memória).

Francamente, dentre os maiores desafios para a conclusão da 1ª fase destaca-se:

- Dificuldade em projetar a arquitetura de sistemas exatamente como a equipa docente exigia: Diversas foram as vezes em que funções tiveram de ser realocadas para pastas completamente distintas da onde haviam sido inicialmente inicializa-das. Para a 2ª fase, é credível que a arquitetura nesse ponto já esteja organizada de forma mais definitiva.
- Complicações com o conceito de encapsulamento: Após certificação do funcionamento das queries, foi apresentado à produção deste código a necessidade de “encapsular” o programa. Isso exigiu uma certa remodulação dos alicerces do trabalho.

O projeto requerido permitiu aos integrantes do grupo revisar a sua base de programação, de forma a identificar déficits pessoais em conceitos fundamentais de computação, e consequentemente corrigi-los.

Por fim, após diversas reestruturações, informa-se que as barreiras à implementação das nossas ideias foram derrubadas com sucesso.