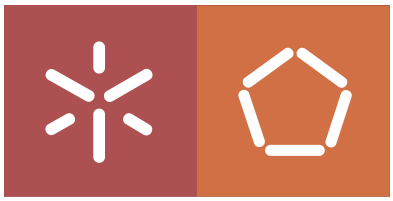

Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática

Grupo 11

Tiago Matos Guedes A97369
Diogo Filipe Durães Ribeiro A89981
Luís Filipe Araújo Ferreira A98286

Laboratórios de Informática III

2º Ano, 1º Semestre

Departamento de Informática

11 de novembro de 2024

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.

2 Modelação

2.1 Arquitetura de Aplicação

A arquitetura do projeto é constituída por 5 partes principais:

- **Parser**: Função é ler e tratar os dados da forma mais propícia a que possam ser inseridos nas estruturas de dados, as hashtables de cada entidade.
- **Entidades**: São basicamente o objeto a ser tratado no nosso projeto, cada uma com as suas características e atributos, neste caso os artistas, as músicas e os utilizadores.
- **Gestores**: Conectam as entidades com as suas respetivas estruturas de dados, conseguindo manipular cada uma delas, criando, eliminando, inserindo objetos nas estruturas para posteriormente serem manipuladas.
- **Utils**: Funcionalidades que podem ser utilizadas em qualquer parte do código do projeto, facilitando o acesso por parte de qualquer estrutura.
- **Queries**: São o objetivo principal da primeira fase do projeto, onde se encontra o código necessário para serem respondidas com sucesso.

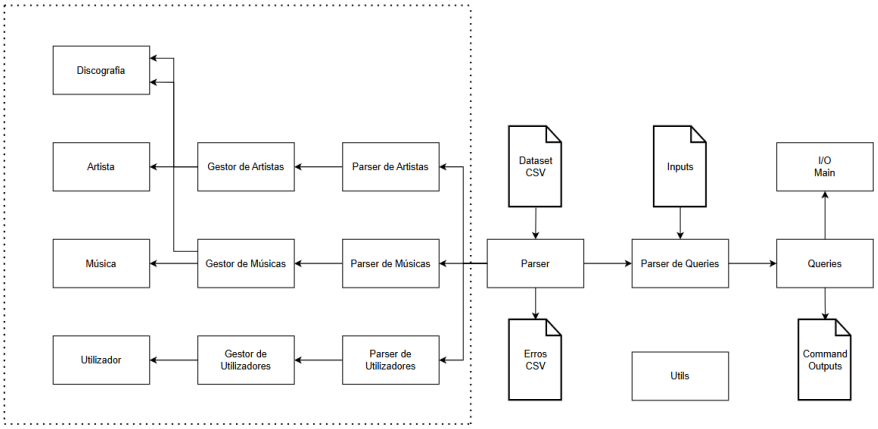

Figura 1: Maquete do Projeto

2.2 Organização dos Dados

2.2.1 Estruturas de Dados

Nos gestores as estruturas de dados utilizadas são as seguintes: *Hashtable* de Artistas, *Hashtable* de Músicas e *Hashtable* de Utilizadores. As estruturas foram implementadas com a biblioteca *GLib-2.0* para facilitar o uso e minimizar erros.

2.2.2 Porquê estas Estruturas?

A escolha das estruturas visou otimizar o tempo de execução em cenários com grandes quantidades de dados. As *Hashtables* permitem acesso rápido a dados específicos guardados nas respetivas estruturas através da *key*, facilitando assim o processo para as queries necessárias á fase 1 do projeto.

3 Implementação

3.1 Verificações

A verificação da validez dos ficheiros, dividiu-se pelo número dos mesmos. Cada ficheiro é verificado de maneira semelhante, linha a linha, o ficheiro é lido e por cada linha o seu conteúdo é verificado, sintáticamente e logicamente. A validez da linha diferencia entre a escrita da linha na estrutura de dados previamente definida, da escrita no ficheiro de erros para o ficheiro em questão, processo que se repete até ao final de cada ficheiro.

Detalhadamente, o primeiro ficheiro a ser analisado é necessariamente o ficheiro de artistas, devido à independência de outros ficheiros para a sua veri-

ficação, seguindo-se o ficheiro de músicas, dependente de artistas para a sua verificação lógica, e por fim, o ficheiro de utilizadores, dependente de músicas.

A validação da linha difere do ficheiro de utilizadores para os restantes, no momento de atribuição de valores a variáveis auxiliares, no caso dos utilizadores, a atribuição ocorre apenas após a verificação válida positiva e essa verificação é feita com a linha no seu formato original (duplicada). Nos restante casos, a atribuição é feita imediatamente após a leitura da linha, e em alternativa à verificação da linha na sua íntegra, a verificação é mais precisa e feita às variáveis auxiliares, já com valores atribuídos.

Excluíndo as funções de validação específica, em todos os casos a linha acaba formatada por *strsep()*, seguida de uma remoção de aspas. São usadas outras funções auxiliares, tendo todas o objetivo de formatar a linha para formato válido de entrada para a estrutura de dados.

Com a mesma importância, o ficheiro de queries é interpretado linha a linha, iniciando-se estruturas e outras variáveis necessárias, antes da leitura do ficheiro. Por cada linha é executada uma query.

3.2 Query 1

3.2.1 Método adotado

Para a realização da 1ª *query*, que tem como objetivo obter informações sobre um utilizador desejado, a função query1 tem como função procurar e retornar os dados principais de um utilizador através do seu username. A função contém 3 parâmetros essenciais para o funcionamento:

- **username**: Do tipo *string* é o nome do utilizador a procurar.
- **gestorUser**: Do tipo da estrutura GestorUser que fornece acesso e permite operações relacionadas com a *HashTable* de utilizadores.
- **outputfile**: Do tipo *FILE* que é o apontador para o arquivo de saída onde os dados serão guardados.

3.2.2 Funcionamento e Aplicação

A função *query1* executa uma pesquisa rápida e simples na *HashTable* de utilizadores, realizada pelo gestorUser, retornando informações de forma rápida e eficiente. Esta função é ideal para consultas, tornando-a eficiente para uso em cenários em que apenas um conjunto de dados específicos do utilizador é necessário.

3.3 Query 2

3.3.1 Método adotado

Para a realização da 2ª *query* estipulada, criou-se uma lista ligada com parâmetros relevantes dos artistas para o problema. A lista ligada à qual associamos o nome discografia, devido às semelhanças com a definição da palavra, contém 6 parâmetros essenciais para a seleção e impressão dos dados, respetivamente:

- **id**: Do tipo *long int*, o identificador único de cada artista, seja individual, ou de grupo.
- **name**: Em formato *string*, o nome associado ao identificador do artista.
- **country**: Também em formato *string*, o país do artista, uma variável opcional na 2ª *query*.
- **duration**: A variável mais relevante para a correção da 2ª *query*, a duração de todas as músicas onde o determinado artista se encontra referenciado como participante. Assume formato inteiro para uma comparação entre artistas diferentes direta. Assumimos a duração em segundos.
- **type**: Para o processo de escrita de resultado não complicar, incluímos o tipo de artista. Tipo de artista que, como apenas será individual, ou em grupo, é do tipo *enum*.
- **next**: O essencial apontador para fazer da estrutura, uma lista ligada, apontando para a próxima estrutura seguindo os moldes aqui descritos, mas para diferentes artistas.

3.3.2 Funcionamento e Aplicação

Começando pelo objetivo final, no momento de pesquisa de discografias, a procura seria eficiente pois cada artista já estaria organizado na lista por ordem decrescente de duração da mesma, ou seja, para casos sem especificação de país, o numero de listas percorridas seria igual ao exigido pela *query*. Com o objetivo final traçado, restam 3 etapas vitais, preenchimento da lista com os dados provenientes da tabela *Hash* de artistas, incrementação em cada artista já na lista com a respetiva duração proveniente, em segundos, de uma música presente na tabela *Hash* de músicas, e por fim, a ordenação decrescente de artistas, por duração.

O preenchimento da lista com artistas recorreu a funções da biblioteca *glib*, para o caso, *g_hash_table_foreach()*. Com o auxílio desta função, e dos gestores dos artistas, preenchemos a lista com o *id*, o nome, o país e o tipo de cada artista, por esta ordem, e tal como veremos à frente, também auxiliou ao incremento sucessivo de durações.

Com recurso à função mencionada no parágrafo anterior e dos gestores de músicas, por cada artista presente na música, incrementamos na sua respetiva duração, a duração da música em segundos.

Terminados os 2 primeiros passos, resta organizar a lista ligada por ordem decrescente de duração de artista.

Todo o processo acima descrito, relativo à lista ligada discografia, é feito antes da interpretação de dados do ficheiro de inputs. Finalizada a discografia, passamos à resolução propriamente dita da 2ª *query*.

Tratando-se de uma *query* do tipo 2, se o programa distingue 3 componentes, é invocada a função que considera o país como condicionante. No cenário mais simples, é invocada a função que responde às N maiores discografias. Ambas as funções têm a simples tarefa de percorrer a lista ligada, identificar os casos favoráveis, e nesse caso, formatar os dados para o pretendido.

3.4 Query 3

3.4.1 Método Adotado

Para a realização da 3ª *query*, que tem como objetivo listar, por ordem decrescente, os géneros de música mais populares numa determinada faixa etária, foi feita a seguinte análise.

A função *query3* recebe 5 argumentos para poder retornar o output esperado:

- **ageMin**: do tipo *int*, representa a idade minima da faixa etária.
- **ageMax**: do tipo *int*, representa a idade máxima da faixa etária.
- **gestorUser**: do tipo da estrutura GestorUser que fornece acesso e permite operações relacionadas com a *HashTable* de utilizadores.
- **gestorMusic**: do tipo da estrutura GestorMusic que fornece acesso e permite operações relacionadas com a *HashTable* de musicas.
- **output**: do tipo *FILE* que é o apontador para o ficheiro de saída onde os dados serão guardados.

3.4.2 Funcionamento e Aplicação

Nesta query é feita uma iteração pela tabela de utilizadores, na qual o gestor de utilizadores é responsável, para cada utilizador verificamos se a idade deste utilizador está compreendida entre o intervalo de idades estabelecido (*ageMin, ageMax*), caso esteja, obtemos o *array* de músicas que o utilizador deu *like* e iteramos neste *array*. Para cada música no *array* de *likes*, procuramos na tabela de músicas (*HashTable*) por uma correspondência com o *ID* da música, recorrendo a uma função de *lookup* encapsulada (*searchMusic*) no gestor de músicas.

Para cada música encontrada na *HashTable* de músicas, obtemos o género correspondente através do *getter* (*getMusicGenre*), verificamos então se esse género já existe no *array "genres"*, que armazena os géneros encontrados. Se o género já existe no *array*, incrementamos o contador de likes no *array "genre_likes"* para o género específico. No caso do género não existir no *array "genres"* e o limite máximo de géneros não tenha sido alcançado, adicionamos um novo género no *array "genres"* e inicializamos o seu contador de *likes* a 1.

Finalmente, após a iteração de utilizadores, verificamos antes da escrita no ficheiro se a *flag* auxiliar tem um valor de 0, esta *flag* permite-nos determinar se na iteração foi encontrado pelo menos um utilizador que esteja compreendido na faixa etária, no caso de não ter sido encontrado nenhum utilizador, escrevemos no ficheiro de *output* uma linha vazia. No caso dos *arrays* auxiliares pelo o cálculo de *likes* para cada género tenha sido populado, isto é, a *flag* auxiliar tem um valor 1, ordenamos estes dois *arrays* tendo em conta a popularidade por ordem decrescente, considerando também o caso empate de *likes* prevalecer a ordem alfabética. Após a ordenação, fazemos uma listagem no ficheiro de *output* os géneros mais populares e os respetivos *likes* para cada.

4 Desempenho

De modo a analisar o desempenho das queries desenvolvidas, foram realizados testes em cada máquina dos elementos do grupo.

- Dispositivo 1:
  - Processador: Intel Core i5-12450H 2.50GHz
  - Memória RAM: 16GB
  - SO: Windows WSL2 Ubuntu 22.04.6
- Dispositivo 2:
  - Processador: Intel Core i3-8130U 2.2GHz
  - Memória RAM: 8GB
  - SO: Windows WSL2 Ubuntu 22.04.6
- Dispositivo 3:
  - Processador: Intel Core i7-7700HQ 2.8GHz(4 CPUs)
  - Memória RAM: 10GB
  - SO: Ubuntu 20.04.6 LTS (VM)

	Dispositivo 1	Dispositivo 2	Dispositivo 3
Query 1	0.000031s	0.000073s	0.000221s
Query 2	0.000375s	0.001024s	0.002558s
Query 3	13.374786s	26.970126s	27.375434s
Tempo de Execução Total	13,6216s	27.3572s	28.0216s
Memória Utilizada	319MB	319MB	319MB

Tabela 1: Análise de Desempenho

Os resultados foram obtidos através do programa de testes desenvolvido que valida a correta implementação das queries, bem como o desempenho do programa.

Podemos verificar que a *query* 1 é executada com bastante rapidez, devido ao facto da pesquisa de um utilizador ser consideravelmente eficiente através da *HashTable* de utilizadores.

No que toca à *query* 2, o tempo de execução, apesar de superior à *query* 1, considera-se também bastante reduzido, devido à simplificação do processo auxiliado por uma lista ligada ordenada por durações de discografias por ordem decrescente. No caso de país indefinido, a lista ligada será acedida tantas vezes quanto o número de discografias pedidas à *query*. No caso em que existe

especificação de país, o melhor caso será igual ao caso anterior, no pior dos casos a lista será acedida tantas vezes quantos artistas existirem. Subtraíndo o tempo de execução total, pela soma das *queries*, tendo o dispositivo 3 como exemplo, assumindo que todo o restante tempo de execução foi consumido pelo preenchimento da lista ligada, o tempo necessário foi 0.643387s.

No entanto, a *query* 3 possui um tempo de execução muito mais elevado em relação às *queries* anteriores, devido a alguns fatores que contribuem para um aumento no custo computacional:

Iteração na tabela de utilizadores, esta *query* precisa de percorrer todos os utilizadores para verificar se a idade deles está compreendida na faixa etária, este processo por si só já tem um custo considerável e, como sabemos as tabelas de *hashing* não são eficientes de todo para este tipo de operações.

Lookups repetitivos na tabela de músicas, para cada música que um utilizador tenha like, a *query* realiza um *lookup* na tabela de músicas para que seja possível obter informação relativa ao género da música.

5 Dificuldades Sentidas

Foram encontradas algumas dificuldades ao longo do projeto na resolução de *memory leaks*, inicialmente não foi feito todos os *free's* e por consequência, foi necessária uma análise extensa com o auxílio do *valgrind* para resolver este problema.

6 Conclusão

Este projeto permitiu adquirir conhecimento acerca de como modularizar um trabalho desta dimensão, gerir da melhor forma os recursos utilizados e tentar obter o maior equilíbrio entre eficiência e recurso gasto. Entre as componentes melhores desenvolvidas a nosso ver destaca-se a estruturação inicial dos dados, passando diretamente dos arquivos *CSV* a informação pronta a ser utilizada pela respetiva *query*. Temos a opinião de que nesta parte fomos muito eficientes e temos uma performance excelente. Por outro lado, alguns aspetos podiam claramente ser melhorados (e serão), como a modularização de certas funções que possivelmente estão mal encaixadas, como também claramente a performance da *query3*, onde poderíamos ter um tempo menor do que o obtido, ao qual vamos trabalhar para isso. Para melhorar no futuro seria interessante melhorar a nossa velocidade de processamento da 3ª *query*, como também melhorar a escalabilidade do nosso projeto, pois será inevitável a queda de performance do mesmo com o aumento de dados.