RELATÓRIO

GUIÃO 3

Laboratórios de Informática III

2021/2022

Licenciatura em Engenharia Informática | 2ºAno

Diana Ribeiro Mateus | A95985

Pedro Marcelo Bogas Oliveira | A95076

Rodrigo José Teixeira Freitas | A96547

Índice

1. Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática III, foi proposta a realização de um trabalho que inclui vários guiões, sendo que este relatório incide sobre o Guião 3.

O Guião 3 consiste na utilização da linguagem C para o tratamento de ficheiros de grande volume, nomeadamente ficheiros do tipo CSV (*comma-separated values*), incluindo a geração de catálogos de dados incluídos nesses ficheiros, a leitura de ficheiros TXT (*text file*) com comandos a ler e a executar, o desenvolvimento de mecanismos de interação entre programa e utilizador, a criação de testes funcionais e de avaliação de desempenho e a otimização de *features* já implementadas, sempre respeitando os conceitos de modularidade e encapsulamento, que apresentam um grande destaque neste projeto. A geração dos catálogos de dados inclui a leitura de ficheiros *csv* através de uma função que permite o *parsing* e a verificação da validade dos dados. Para além disso, a leitura de ficheiros *txt* e/ou a interação com a interface desenvolvida permitem ler e executar os comandos disponíveis. Finalmente, ocorre a geração de ficheiros finais do tipo *txt* ou a apresentação do seu conteúdo no menu, com o output pretendido da respetiva *query* e obtido a partir dos catálogos produzidos inicialmente.

Foi necessário recorrer a princípios mais avançados de programação, tal como o uso de estruturas de dados eficientes para armazenar e consultar grandes volumes de informação, garantindo sempre, tal como mencionado anteriormente, o encapsulamento dos dados e a estruturação do projeto em módulos.

Os objetivos principais deste trabalho passam pela utilização da linguagem C e de todas as suas funcionalidades, com destaque na modularidade e encapsulamento, estruturas dinâmicas de dados, testes de desempenho, otimização de *features* já implementadas em guiões anteriores, e ainda o desenvolvimento de um menu interativo, para além da consolidação de todos os conhecimentos essenciais ao desenvolvimento de um projeto.

2. Otimização do Programa

O desenvolvimento do guião 3, numa primeira fase, passou pela otimização dos recursos implementados em guiões anteriores.

2.1. Gestão de Dados

De uma forma geral, as estruturas de dados utilizadas nos guiões anteriores restringiam, enormemente, o desempenho do programa, a um nível de tempo de execução. Desta forma, foram utilizadas *Hashtables*, ou seja, uma estrutura de dados especial que associa chaves de pesquisa a valores. O seu principal objetivo é, a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado. Este tipo de estrutura adequa-se perfeitamente ao tipo de projeto que se foi desenvolvendo, considerando que são necessários vários acessos à informação principal em cada *query*.

Assim, o tempo de acesso em média das estruturas utilizadas para os catálogos passou de $O(\sqrt{N} * N)$ a $O(1)$, diminuindo o tempo de acesso aos catálogos. No que concerne ao tempo de inserção, ambas as estruturas apresentam uma complexidade temporal constante $O(1)$, não sofrendo alteração, por isso, neste aspeto.

Outras otimizações a nível dos dados passam pelas alterações das funções principais de manuseamento das linhas dos ficheiros, que passaram a incluir a filtração por utilizador, e a troca de estratégia de verificação dos dados, passando de *switchs* para *flags*, como também as funções de verificação de datas., que permitiram a diminuição considerável do tempo de criação de catálogos e verificação de dados.

2.2. Queries

Outro aspeto que foi necessário otimizar foi o desempenho das *queries*. Com os novos ficheiros de grande volume e as novas estruturas de dados, foi possível otimizar o desempenho de algumas *queries* já existentes, para um tempo considerado útil (abaixo de cinco segundos).

3. Módulos e Estruturas de Dados

3.1. Main

O módulo principal permite diferenciar os dois *pathways* relacionados com a compilação com e sem ficheiro de comandos. Para além disso, inlcui a abertura e o fecho dos ficheiros *csv*, bem como a chamada de funções associadas à verificação e criação dos catálogos e à gestão das *queries*.

3.2. Verificação

Este módulo permite a verificação e a filtração dos ficheiros dos *commits* e dos repositórios, consoante o ID de repositório e a quantidade de *commits*, respetivamente.

3.3. Interface

O módulo da interface trata da parte interativa do programa com o utilizador, permitindo a escolha de *queries* para visualização de resultados, bem como o avanço nas páginas para a visualização de todos os resultados. No final, é possível retornar ao menu principal e sair deste, voltando para o terminal.

Este módulo executa a criação e verificação dos catálogos antes de apresentar o menu principal. Após a escolha da *query*, o programa executa a *query* correspondente, cria o ficheiro com o output e percorre o ficheiro para imprimir os resultados no terminal.

3.4. Catálogos

Os módulos dos catálogos, nomeadamente, dos utilizadores, *commits* e repositórios, contêm as estruturas de dados para o armazenamento definidas, isto é, as *Hashtables*, bem como todas as funções auxiliares associadas a estas, tal como, operações de inserção, pesquisa e obtenção de dados associados aos utilizadores, *commits* ou repositórios. Ainda se encontra incluída a função de verificação dos diferentes campos associados a cada linha do ficheiro correspondente e a filtração do diferentes catálogos.

Para além destas funções, apresentam ainda funções de procura e inserção de informação em estruturas auxiliares, úteis para a execução das *queries*. De entre elas, encontram-se, por exemplo, as funções *countTypeBots* e *friendsTree*, no ficheiro *HT_users.c*, as funções *countCommitsReposLanguage* e *compareAllDates* no ficheiro *HT_commits.c* e as funções *filterReposLang* e *countRepos* no ficheiro *HT_repos.c*.

3.5. Estruturas Auxiliares

Existem dois módulos associados às estruturas auxiliares, *auxiliarStruct* e *auxiliarTree*.

O primeiro módulo trata estruturas auxiliares do tipo lista ligadas.

Este módulo permite a realização de operações de inserção, pesquisa e obtenção de dados nela presentes. Para além disso, apresenta algumas funções auxiliares associadas às estruturas definidas, utilizadas em várias *queries*, tal como a *mostFreq* e a *countInfoStr*, bem como a realização de operações de libertação de memória destas estruturas auxiliares.

```c
struct data
{
    char *info;
    struct data *next;
};
```

```c
struct string
{
    char *info;
    int counter;
    struct string *next;
};
```

```c
struct topUsersStruct
{
    char *login;
    char *id;
    int counter;
};
```

Figura 1 - Estruturas de Dados Auxiliares DATA, STRING e TOPUSERSSTRUCT

O segundo módulo trata de árvores binárias e todas as funções associadas a este tipo de estrutura e permite a realização de operações de inserção, pesquisa, contagem de nodos e libertação de memória.

```c
struct tree
{
    int value;
    struct tree *l, *r;
};
```

Figura 2 - Estrutura de Dados Auxiliares TREE

3.6. Queries

O módulo das *queries* é o responsável pelo fluxo destas, tratando de receber o input do utilizador, chamar a função da *query* correspondente, fornecendo-lhe o input necessário e enviando o output para os ficheiros correspondentes, para poder ser apresentado ao utilizador.

3.7. Funções Auxiliares

Este módulo apresenta as funções auxiliares, normalmente, utilizadas por vários módulos, tais como funções de verificação de campos, tal como *testInt* ou *testString*, e ainda, a função de *hash_key* utilizada para mapear dados nas *Hashtables*.

3.8. Testes de Desempenho

O módulo dos testes de desempenho apresenta uma função *main* que permite a criação e verificação dos catálogos, bem como o processamento do ficheiro de comandos e o redireccionamento para uma outra função que trata dos testes de cada *query*, a função *test*. Esta última permite a contagem do tempo em cada teste, bem como a comparação de resultados entre ficheiros produzidos e ficheiros esperados e a impressão do tempo e resultado da comparação, através das funções auxiliares *compare* e *printTest*. De ressalvar que apesar de alguns dos ficheiros produzidos e esperados conterem o mesmo conteúdo, é possível que a ordem pelo que os resultados aparecem possa ser diferente. Neste caso, os

testes apenas foram desenhados para compararem resultados exatamente iguais, constituindo uma das falhas do nosso programa.

4. Complexidade das Estruturas

Para a criação dos catálogos dos utilizadores, *commits* e repositórios utilizaram-se *Hashtables*. Esta estrutura foi utilizada para otimizar o tempo de execução das *queries*. A diminuição da complexidade do tempo de pesquisa dos dados melhorou bastante o tempo despendido nas *queries*, tal como mencionado anteriormente. Desta forma, consideramos que a alteração do tipo de estrutura foi bem-sucedida e produziu resultados esperados, a nível do tempo de execução do programa.

Além disso, foram utilizados quatro tipos de estruturas auxiliares (*data*, *string*, *topUsersStruct* e *tree*).

A primeira estrutura *data* é uma lista ligada que permite a inserção de dados únicos, ou seja, apenas uma informação em *string*. Foi utilizada, por exemplo, na *query* 7 para guardar o ID dos repositórios inativos a partir de uma data.

A segunda lista ligada usada, *string*, permite essencialmente a contagem de dados. Apresenta uma *string* como variável a guardar e um inteiro, *counter*, que conta o número de vezes que a primeira aparece. Foi utilizada, por exemplo, na *query* 8 para guardar o número de vezes que uma linguagem é utilizada.

A *topUsersStruct* é um *array* com várias informações em cada índice, utilizada principalmente, nas *queries* em que era necessário um top N de utilizadores (*query* 5 e 6), já que a inserção nesta estrutura permitia a inserção, num *array* de tamanho N, dos dados por ordem de tamanho da informação pedida. Apresenta em cada índice uma variável em formato *string* para guardar o login, outra para guardar o ID também em formato *string* e por último uma variável *int*. A utilização desta estrutura está associada a um campo inserido no catálogo de *users* inicial (*auxCount*) que contava uma informação específica e pedida pela *query*, como por exemplo, as vezes que um utilizador realizava um *commit* num intervalo de datas específico, de forma a ser possível determinar os utilizadores mais ativos (*query* 5).

A utilização de árvores binárias contém apenas uma variável do tipo inteiro. Esta estrutura foi utilizada, por exemplo, na *query* 2, com o intuito de guardar o ID dos utilizadores que seriam colaboradores.

5. Testes de Desempenho

De forma a medir o tempo de execução do nosso programa foi utilizada a biblioteca *time.h*, através da utilização da função da biblioteca, *clock(),* que associada à compilação/execução do módulo dos testes permitia a obtenção dos valores de execução.

Para qualquer um dos testes foram utilizados os comandos abaixo:

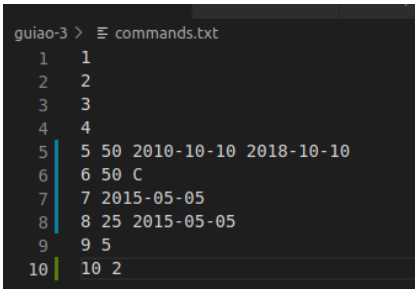

Figura 3 - Comandos utilizados na execução das *queries*

Cada *query* foi executada dez vezes, tendo sido descartados os valores mínimos e máximos e tendo sido feita a média dos restantes valores. Utilizaram-se três máquinas para a realização dos testes, com as seguintes características:

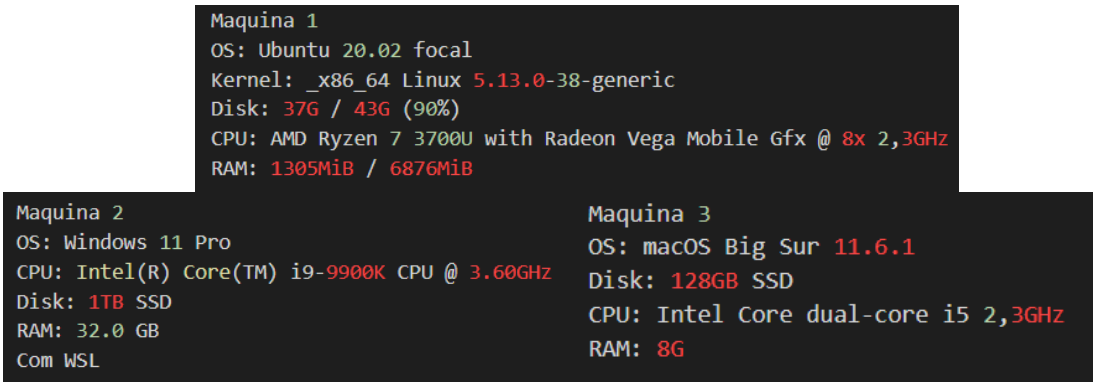

Figura 4 - Especificações das máquinas utilizadas nos testes de tempo e desempenho

O primeiro teste realizado é relativo ao desempenho dos resultados obtidos pelas *queries*. Foram utilizados os ficheiros de saída do guião 2, com um tamanho de 126MB, que permitiram a comparação dos resultados do presente guião com os resultados obtidos no guião anterior. Os resultados obtidos apresentam-se de seguida:

Queries	Maquina 1		Maquina 2		Maquina 3	
Teste	tempo	resultados	tempo	resultados	tempo	resultados
Query 1	0.162 s	Certos	0.133 s	Certos	0.132 s	Certos
Query 2	0.363 s	Certos	0.250 s	Certos	0.285 s	Certos
Query 3	0.291 s	Certos	0.234 s	Certos	0.276 s	Certos
Query 4	0.139 s	Certos	0.086 s	Certos	0.110 s	Certos
Query 5	0.717 s	Errados	0.516 s	Errados	0.585 s	Errados
Query 6	0.420 s	Errados	0.320 s	Errados	0.376 s	Errados
Query 7	0.210 s	Errados	0.164 s	Errados	0.232 s	Errados
Query 8	0.075 s	Certos	0.055 s	Certos	0.054 s	Certos
Query 9	1.781 s	Errados	1.352 s	Errados	1.939 s	Errados
Query 10	--- s	---	--- s	---	---	---

Figura 5 - Resultados obtidos na execução das *queries* com os ficheiros do guião 2

De uma forma geral:

- As *queries* executaram em tempo útil (menos de 5 segundos), exceto a *query* 10. Apesar das tentativas, não foi possível colocar esta *query* a executar em tempo útil;
- As *queries* 1, 2, 3, 4 e 8 apresentam os resultados iguais aos ficheiros esperados;
- As *queries* 5, 6, 7, e 9 não apresentam os resultados iguais aos ficheiros esperados. Existem neste caso, 3 situações que explicam estes resultados. A primeira, inclui o que acontece nas *queries* 5 e 6. Os resultados do guião 3 estão efetivamente errados e o código associado a estes *queries*, ao ser reescrito para as novas estruturas, não funciona como o que se pretendia. A segunda situação, isto é, o que acontece na *query* 7, é que os resultados obtidos são iguais aos resultados esperados, contudo, a ordem em que aparecem nos ficheiros de saída do teste é diferente da ordem dos ficheiros com os resultados esperados. Como o teste desenvolvido compara linha a linha, o resultado do teste indica que os resultados são diferentes, quando na realidade apenas estão dispostos de uma forma diferente. A última situação é a da *query* 9, que acontece porque a ordem com que os dados aparecem na *Hashtable* é diferente daquela que existe nos ficheiros originais. Desta forma, resultados em *borderline* com os mesmos valores serão diferentes, e por isso o resultado do teste será que os resultados são diferentes, quando na realidade estão também corretos;
- Comparando com os testes do guião 2, é notória a otimização feita no programa, não só a nível das *queries*. Apesar dos comandos utilizados terem sido diferentes dos utilizados no guião 2, foram utilizados comandos mais exigentes, com um número de resultados maior a obter em cada *queries*. Desta forma, *queries* que anteriormente estariam muito lentas tal como a 5 e a 7, com tempos de execução de 29.7 segundos e 65.67 segundos, respetivamente, passaram a estar muito mais rápidas, executando em 0.717 segundos e 0.232 segundos, no máximo, respetivamente.

Além destes testes, foram utilizados os ficheiros do guião 3, ficheiros maiores com 574MB de tamanho global, mais de quatro vezes maior que o tamanho dos ficheiros utilizados no guião 2. De forma a testar se as *queries* executavam em tempo útil, usaram-se as máquinas especificadas acima, nomeadamente a 2 e a 3. A máquina 1 não permitia a execução do programa com os ficheiros grandes, provavelmente, por falta de memória.

Os resultados apresentam-se de seguida:

Queries	Maquina 2		Maquina 3	
Teste	tempo	resultados	tempo	resultados
Query 1	1.266 s	Certos	1.380 s	Certos
Query 2	0.969 s	Certos	1.130 s	Certos
Query 3	1.109 s	Certos	1.257 s	Certos
Query 4	0.578 s	Certos	0.644 s	Certos
Query 5	7.172 s	Errados	8.013 s	Errados
Query 6	1.875 s	Errados	2.052 s	Errados
Query 7	0.672 s	Errados	0.783 s	Errados
Query 8	0.188 s	Certos	0.191 s	Certos
Query 9	12.578 s	Errados	13.260 s	Errados
Query 10	--- s	---	--- s	---

Figura 6 - Resultados obtidos na execução das *queries* com os ficheiros do guião 3

De uma forma geral:

- A *query* 10 continua a não executar os resultados num tempo útil;
- A máquina 2 com umas especificações mais avançadas do que a máquina 3, nomeadamente, processador mais rápido e mais memória de disco e RAM, executa as *queries*, num tempo sempre menor do que a máquina 3;
- As *queries* 5 e 9 não executam em tempo útil (menos de 5 segundos) enquanto as restantes executam em tempo útil.

6. Conclusão

A realização deste trabalho foi importante na consolidação dos conhecimentos da linguagem C, através da aplicação dos conceitos de modularidade e encapsulamento, no desenvolvimento de testes e interface, bem como na interpretação de comandos e extração de dados.

Quanto à modularidade e encapsulamento, concluímos que o objetivo foi cumprido, não só pelo cuidado apresentado na implementação de módulos úteis e lógicos, como também na criação de *header files* que garantissem o encapsulamento e a implementação de tipos opacos.

O desenvolvimento dos testes ficou um pouco aquém do esperado, bem como o desenvolvimento e alteração das *queries* para as novas estruturas de dados, considerando que os resultados nem sempre foram os esperados, quer devido às *queries*, quer devido à qualidade dos testes. Os tempos das *queries* também não satisfizeram as nossas expectativas.

No que concerne à criação do menu, concluímos que também não foi implementado de forma exemplar considerando que não é possível retroceder de página nem seguir para uma página em específico.

Sentimos que podíamos ter ido mais longe relativamente aos resultados e tempos obtidos nas *queries*, bem como no desenvolvimento do menu e dos testes de desempenho. Poderiam ter sido implementadas mais *features* nos dois últimos módulos, sendo que um dos impedimentos foi o tempo de realização do projeto. Quanto às *queries* poderiam ter sido implementadas estruturas de dados e funções auxiliares mais eficientes.

Relativamente aos testes de tempo e desempenho, concluímos que estes ficaram muito aquém do esperado, principalmente devido ao que foi mencionado anteriormente relativamente ao desenvolvimento dos testes e das próprias *queries*.

Desta forma as dificuldades que mais sentimos neste guião foi a própria gestão do tempo, considerando que tivemos de aprender uma nova estrutura de dados, implementá-la e alterar os guiões anteriores de forma que pudessem funcionar para as novas condições. Para além disso, despendemos de muito tempo na otimização inicial com a criação de catálogos e na reimplementação das *queries*, o que não nos permitiu ir mais longe nos testes e menu desenvolvidos.

Em conclusão, este trabalho deixou um pouco a desejar no que diz respeito aos objetivos cumpridos, contudo, permitiu-nos ter um processo intenso de aprendizagem relativamente ao manuseamento de novas estruturas de dados, gestão de tempos de execução e no desenvolvimento de *software* me equipa.