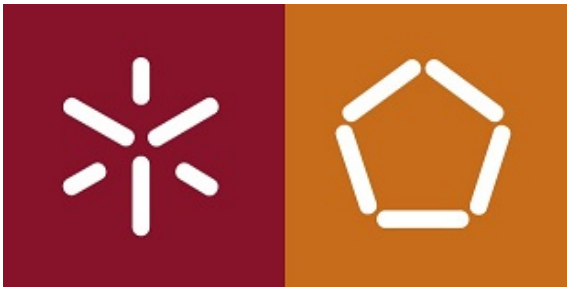


Universidade do Minho
Departamento de Informática

Github - Gestão de dados

LEI - Laboratórios de Informática 3

Grupo 9

5 de fevereiro, 2022

Filipa Gomes
(a96556)

Ricardo Oliveira
(a96794)

João Loureiro
(a97257)

Introdução

Este guião tem como objetivo principal expandir e refinar os conceitos e funcionalidades introduzidos no guião anterior, juntamente com a adição de novas funcionalidades.

O primeiro dos objetivos é desenvolver um menu a ser invocado quando o programa não recebe um ficheiro de entrada como argumento. Este deve, no início, exibir as todas as possíveis queries a executar e as suas respetivas funcionalidades. Após o utilizador inserir o número e o argumento da query desejada, o programa deve executa-la e imprimir os resultados com um mecanismo de paginação, também controlado pelo utilizador.

O segundo objetivo envolve criar um mecanismo de testes que, dados os testes das queries, executa-os, calculando e, depois exibindo, o tempo de execução de cada uma e comparando os resultados obtidos aos esperados. Se a query for executada em menos de 5 segundos, considera-se executada em tempo útil. O último objetivo consiste na otimização e construção de um mecanismo eficiente de gestão de dados, um meta-catálogo, para que consigam fazer o parsing de ficheiros de maior dimensão de dados e as queries requisitadas pelo utilizador em tempo útil.

Espera-se no fim deste guião que o programa continue de acordo com os princípios de encapsulamento e modularidade abordados durante o semestre e que consiga efetuar os objetivos de forma eficiente e em tempo útil.

Capítulo 1

Meta-Catálogo

1.1 Estruturas de Dados

Uma das grandes mudanças feitas em comparação com o guião 2, foi a introdução de um meta-catálogo.

Este,representado pela *struct meta_catalog*, contem os catálogos de users, repos e de commits, também como várias estatísticas globais contidas na stats (de tipo *GStats*).

Os catálogos, à semelhança do guião 2, contêm os registos e informações (do seu tipo respetivo) pertinentes para a execução das várias queries, guardados e geridos através de hash tables construídas por nós.

A *GStats*, como o nome indica, contem estatísticas globais sobre os registos que facilitam a execução das queries mais "simples" (1 a 4). Com a introdução do meta-catalogo, alteramos também as funções que inserem os registos e as que devolvem as informações destes, no seu respetivo catalogo, para que sigam a nova estrutura. A maioria destas funções encontram-se no novo ficheiro *meta_catalog.c*.

A introdução deste deriva em grande parte a uma necessidade de melhorar o encapsulamento de dados do programa, uma vez que impede a interação direta entre as 'querys' e os dados guardados em memória, prevenindo assim possiveis alterações involuntárias de campos que, de outro modo, não deviam poder ser alterados.

1.2 Eficiência

Para testar a eficiência deste meta-catalogo, executou-se o guião 2 (sem o meta-catalogo) e o guião 3 (com o meta catalogo), utilizando os ficheiros de entrada disponibilizados para o guião 2 (visto que o guião 2 não foi feito com ficheiros tão grandes, quanto os do guião 3, em mente) e o *global_tests.txt* como argumento (ver pagina 5).

Os resultados, obtidos na máquina do João (ver especificações na pagina 7), mostram uma diminuição significativa do tempo de execução sendo que no guião 2 demorou cerca de 28.5 segundos a executar, e no guião 3 cerca de 5.6 segundos.

Nota: Resolveu-se executar este teste nesta máquina em particular, devido a ser uma Virtual Machine (naturalmente menos rápida que um sistema operativo instalado).

Estes resultados são produto das otimizações feitas para o guião 3, dentro das mais notáveis, a implementação do meta-catalogo, o que levou a um aumento significativo de desempenho(mesmo na maquina menos poderosa do grupo), e a resolução de problemas a nivel de código como leaks de memória.

Assim, a implementação deste novo método permitiu reforçar os princípios chaves de encapsulamento e modularidade (evita acessos indevidos aos catálogos), e permitiu reduzir o tempo de execução das queries em comparação ao guião 2, não alterando o seu funcionamento.

Capítulo 2

Menu Interativo

2.1 Menu

De acordo com as indicações do enunciado deste guião, em adição ao programa poder ser invocado com a diretoria de um ficheiro como argumento (como implementado no guião 2), o programa pode ser executado sem um argumento.

Neste caso, após fazer o parsing usual dos ficheiros localizados na pasta de entrada, é apresentado ao utilizador um menu, igual ao que se encontra na figura, com o número e descrição de cada query, bem como a possibilidade de dar reload aos ficheiros de entrada, ver mais sobre o mesmo, ou ainda introduzir o nome de um ficheiro de comandos para que estes sejam executados e guardados. Este último processo executa de maneira idêntica ao guião 2.

2.2 Implementação do menu e paginação

Para implementar o menu, alteramos algumas funções que já usávamos no guião 2. Quando o programa é invocado sem argumentos, usamos uma nova versão da função loop *(interpeter.c)* que sempre que é invocada imprime o menu e, de seguida, recebe os argumentos necessários para executar cada query, caso esta seja escolhida para ser executada. Quando invocado com

argumentos, a função *interpete_query_file (interpeter.c)* é invocada, e as queries são executadas bem como no guião anterior.

Modificamos também todas as funções que tratam de cada query. Isto deve-se, maioritariamente à implementação do meta-catalgo e de uma procura de resolver possiveis problemas de encapsulamento anteriores. Quando uma destas é invocada pelo menu, recebem como um argumento um ficheiro *NULL*, sendo que quando invocadas com o método do guião 2 (um ficheiro de comandos como argumento inicial do programa), este ficheiro passa a ser o ficheiro de output onde se encontram os resultados (como visto no guião 2) e serve como meio de indicar às próprias querys onde deverá ser guardado o 'output'. No final de cada uma, os rezultados ou são escritos e devolvidos diretamente ao utilizador, caso estas sejam invocadas através do menu e o seu resultado seja estatistico (queries 1 a 4 e 12 a 14) ,ou são armazenados em ficheiros para futuro uso. É de notar que nas queries 5 a 11, ainda que acessadas através do menu, têm os seus dados aramazendados localmente.

O ficheiro *pagination.c* inclui todas as funções utilizadas para a criação e interação do menu de resultados.

Para facilitar a construção deste menu, resolvemos guardar os resultados da query executada num ficheiro temporário *cache.txt*, que como o nome sugere, simula o efeito de uma cache. Cada linha deste ficheiro corresponde a um dos resultados obtidos.

Depois, é invocada a função *loop_pages*, que utiliza a cache para calcular o número de páginas necessários para apresentar todos os resultados (sendo que cada página apresenta, no máximo, 20 resultados). Feito isto, invoca a função *print_page*, que imprime sequencialmente os resultados a partir da cache, e juntamente com ela desenha o menu completo.

```
Insira opção:
1
*~~~~*
| 1  |  Bot : 72; Organization : 22903; User : 404409
*~~~~*
```

Este processo é repetido para as diferentes páginas que o utilizador escolher para visualizar, até este decidir sair do menu. Neste último caso, o utilizador volta para o menu inicial, e a função *clean_cache* elimina o agora obsoleto ficheiro da cache.

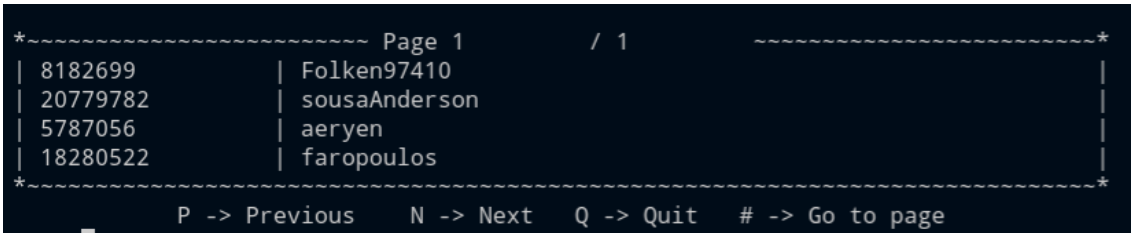
Resolvemos usar esta estratégia de modo a evitar possiveis quebras do encapsulamento e modularidade, uma vez que estes dados necessitariam de ser armazenados em memória até ao momento de serem usados pelo menu. Isto permitiu também um melhor desempenho a nível de performance, uma vez que estes dados, sendo guardados em disco, podem ser libertados da memória, diminuindo assim, as necessidades do programa.

2.3 Execução das queries

A quando da seleção de uma das querys para execução o utilizador insere, caso necessário, os argumentos desejados para a execução da query, de modo a obter, no STDOUT, os resultados para o pedido efetuado.

Das queires 1 a 4 e 12 a 14, o resultado é devolvido de imediato (devido a estas queries serem sobre estatísticas globais), enquanto da 5 a 11, o utilizador recebe os resultados sobre a forma de um menu de paginação interativa. Este segue um formato de 80x24 caracteres, para uma melhor vizualização por parte do utilizador.

```
Insira opção:
9
Número desejado de utilizadores:
4
```



O utilizador pode então usar os comandos mostrados no ecrã para interagir com o menu, com o objetivo de observar os resultados obtidos: P retrocede para a página anterior, N avança para a seguinte, Q volta ao menu e # (número da página desejada) avança para uma determinada página.

Capítulo 3

Testes de Desempenho

3.1 Desenvolvimento dos testes

O último exercício a abordar é o do desenvolvimento do executável dos testes de desempenho e qualidade. Este executável deve apresentar os tempos de execução de cada query, verificar se cada uma destas foi executada em tempo útil (menos de 5 segundos) e verificar, ainda, se os resultados obtidos são iguais aos resultados esperados.

Primeiramente, deparamo-nos com a necessidade de alter o ficheiro makefile para que este gerasse, para além do executável normal do guião-3, um executável de nome './tests'. Este, quando executado, apresenta um menu interativo onde, o utilizador, pode escolher o tipo de testes a realizar: testes de performance, cuja funcionalidade é verificar se as queries são executadas em tempo útil, ou os testes de qualidade, que verificam se os resultados obtidos das queries são iguais aos esperados.

Cada respetivo tipo de teste tem uma pasta dedicada a si na pasta testes, tendo ela dentro uma pasta de entrada (onde os ficheiros de entrada desejados devem ser colocados), um ficheiro *.txt, que contem os comandos desejados para testar as queries, e no caso dos de qualidade, uma pasta que contem os ficheiros esperados.

Após escolhida a opção, o programa faz o parsing dos respetivos ficheiros de entrada, e de seguida utiliza o ficheiro de texto, que por padrão são iguais ao *global_tests.txt* (cujo conteúdo se encontra em baixo), como argumento para executar as queries.

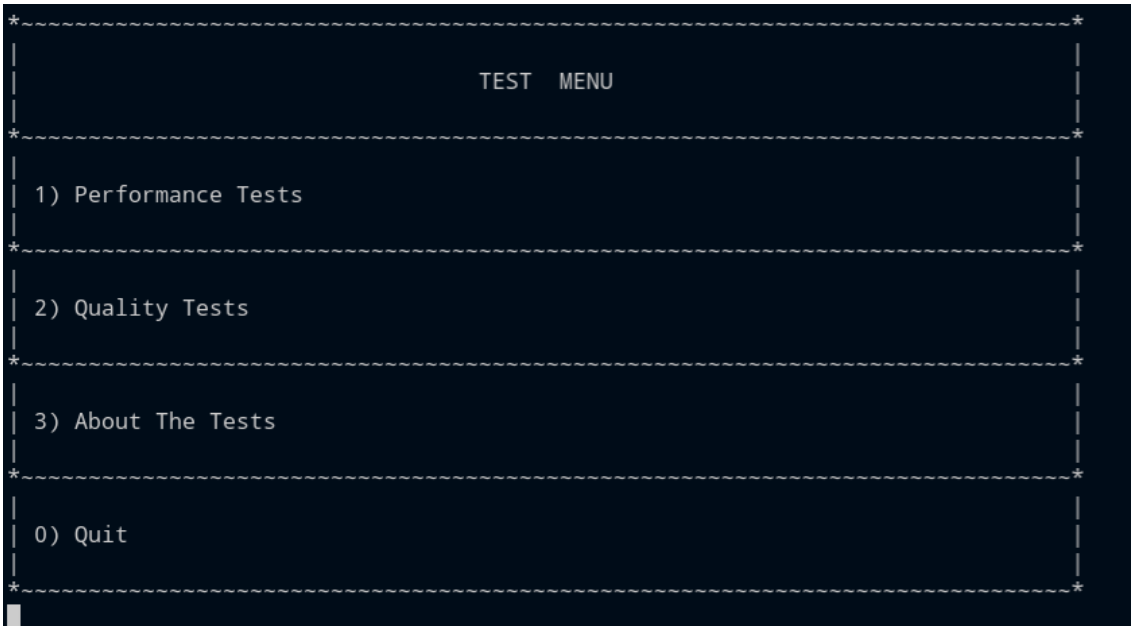
```
1
2
3
4
5 100 2015-10-10 2017-10-10
6 100 Java
7 2015-01-01
8 100 2017-01-01
9 5
10 50
11 80
12
13
14
```

Usamos estes argumentos em particular porque, para alem de acharmos que permite obter um resultado mais global em termos de tempo de execução, foram também os mesmos que usamos para o guião 2, o que nos permitiu analisar mais facilmente as diferenças entre os 2 (apesar da diferença no tamanho dos ficheiros de entrada dos dois).

Naturalmente, para testar outras combinações de testes para este guião, tudo que o utilizador tem de fazer é alterar o conteúdo do ficheiro txt desejado, sendo de notar, que para testes de qualidade, devem ainda, ser alterados os ficheiros de output esperado, de modo que as funcionalidades sejam verificadas.

Após todos os testes serem executados, o utilizador recebe a informação sobre os seus testes na forma de um menu interativo controlado da mesma forma que o menu do guião 3 (semelhante

ao que se encontra na imagem seguinte).



3.2 Execução dos testes

Em relação aos testes de performance, para alem dos tempos de execução de cada query, calculamos também o tempo que demorou a fazer o parsing (Boot Time) e o tempo total de execução (Total Run Time), para facilitar e completar as analises de desempenho.

Todas as funções que tratam dos testes de execução encontram-se no ficheiro *maintests.c*. Para calcular os tempos de execução, usamos as funções predefinidas da libraria *time.h*, em particular a clock. Esta é invocada no início e no fim de uma determinada operação, sendo guardada em duas variáveis diferentes. O tempo final de execução é obtido pela diferença entre o clock do fim e do início, dividida pelo número de clocks por segundo. Usamos este método por ser o mais recomendado e o que pareceu mais útil na nossa pesquisa.

Quando o tests é executado e o tipo de testes é escolhido, a função main deste ficheiro faz o parsing dos ficheiros como usual. Após calculado e o tempo que demorou a executar esta tarefa, o programa invoca a função *tests*, que recebe a flag correspondente ao teste a executar, usando por sua vez a *do_testes* para executar as queries.

A *do_tests*, uma versão modificada da função *interprete_query_file*, recebe uma flag como argumento que lhe indica qual o tipo dos testes a fazer. Independentemente do tipo requisitado, ela executa e calcula o tempo de execução de cada query, guardando cada um destes num array de doubles. Se forem requisitados os de qualidade, também invoca a função *verify_results*, responsável por comparar os ficheiros de resultado obtidos aos esperados. A função utiliza os dois ficheiros correspondentes a query a ser testada, comparando o conteúdo deles e contando o número de "erros", ou seja, o número de carateres diferentes entre eles. Este valor é depois armazenado num array de inteiros para depois serem exibidos.

Os ficheiros esperados foram obtidos através da execução do guião 2 com o ficheiro *global_tests.txt* como argumento, e como ficheiros de entrada os ficheiros do guião 2 disponibilizados na Blackboard. Tivemos, no entanto, de aumentar a capacidade das hashtables utilizadas no guião 2 para que tivessemos resultados comparáveis, visto que o tamanho da hashtable altera a ordem em que os resultados são obtidos e, por sua vez os Outputs. A versão utilizada é a que se encontra atualmente disponivel no Github.

Por último, obtém o tempo total de execução (através da função *get_tot_time*) e imprime os

resultados sob a forma de um menu interativo (gerado e controlado pela função *tests*). A função *print_test_results*, como o nome indica, usa os arrays dos resultados para imprimir o tempo de execução de cada query e calcular e exibir quantos segundos acima ou abaixo do tempo útil esteve cada uma, e/ou exibir o número de diferenças (vulgo "erros") que o resultado obtido teve em comparação ao esperado.

Resultado dos testes

Em seguida, registamos os resultados da execução do executável de testes deste guião. Os testes foram realizados individualmente, na sua própria máquina, por cada membro do grupo, com as seguintes observações:

- os ficheiros de entrada utilizados foram os disponibilizados na Blackboard para o guião 3;
- utilizou-se o *global_tests.txt* presente na pasta testes do repositório (cujo conteúdo encontra-se na página 5).

Para acompanhar a analise dos resultados, aqui estão as especificações do ambiente de execução de cada elemento do grupo:

- João: Sistema operativo Ubuntu a correr numa máquina virtual no Windows, com 4 GB de RAM e 1 core atribuido Intel Core i7-10750H 2,592GHz.
- Ricardo: Sistema operativo Manjaro. 14 GB de RAM e AMD Ryzen 7 3700U 8x cores 2.3GHz.
- Filipa: Sistema operativo Manjaro. 8 GB de RAM e Intel i7-1065G7 8x cores 3.9GHz.

Maquina	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Joao	0.000008	0.000082	0.000003	0.000003	1.43	1.74	1.66	0.59
Ricardo	0.000010	0.000007	0.000003	0.000004	1.97	2.40	0.22	0.72
Filipa	0.000010	0.000006	0.000003	0.000003	0.27	0.37	0.08	0.18

Maquina	Q9	Q10	Q11	Q11	Q12	Q13	Boot	Total
Joao	0.93	2.13	1.27	0.13	0.04	0.03	19.19	27.46
Ricardo	1.13	2.95	1.64	0.16	0.049	0.047	26.27	37.77
Filipa	0.17	0.72	0.019	0.018	0.011	0.014	4.41	6.04

Analisando os resultados, podemos observar que todas as queires foram executadas em tempo útil. Devido ao seu processador mais rapido, a Filipa tem tempos significantemente mais rapidos que os outros elelmentos.

Os tempos de execução das primeiras 4 queries e das ultimas 3 foram os mais baixos e são semelhantes em todos os elementos, o que era de esperar dado a implentação do *GStats*.

Nas outras queries, já houve diferenças significativas, sendo a query 10 a que mais tempo demorou a executar para todos, com a 6 em segundo lugar. Estes resultados já eram esperados, já que a 10 devolve o top N de utilizadores com as maiores mensagens de commit em cada repositorio, ou seja, tem de devolver varias mensagens de cada repositorio. Por sua vez, a 6 devolve o top N de utilizadores com mais commits em repositórios de uma determinada linguagem, o que exige um cross-checking intensivo por parte do programa.

Capítulo 4

Observações

4.1 Queries extra

A fim de adicionar algumas funcionalidades extras ao projeto acabamos por implementar 4 queries extras (query 11 a 14).

A query 11 devolve o top N de utilizadores com as maiores mensagens de commit nos registos, bem como fora anteriormente implementada no guião dois.

Já as queries 12, 13 e 14 (criadas neste guião) contêm, respetivamente, o número de registos de users, commits e repos validos dos respetivos ficheiros de entrada. São, portanto, estatísticas extras de teste, que nos permitiu controlar a verificação dos ficheiros de entrada, garantindo assim a introdução do guião 1 no projeto final.

4.2 Problemas e desafios

Este guião, apesar de ser o final, trouxe ao grupo menos problemas, uma vez que fomos capazes de compreender melhor os nossos objetivos e limites no que toca à manipulação de ficheiros e melhor controlo de memória.

O principal problema surgiu com a quantidade de memória desperdiçada inicialmente, no entanto, ao recorrer ao software Valgrind, fomos capazes de identificar uma grande parte dos problemas e resolver os 'leaks' de memória.

Finalmente, o último problema com que nos deparamos foi a quando da análise qualitativa da query 7, uma vez que estavamos a ser deparados com erros. Aquando de uma análise mais profunda do ficheiro de output contendo as linhas "erradas", descobrimos que estes não eram na verdade erros, mas sim repositórios inativos que o guião dois não fora capaz de detetar. Tendo assim a filtragem de dados sido melhorada a nivel da query 7.

Conclusões

Após trabalhar neste guião, conseguimos reforçar e consolidar ainda mais os princípios de encapsulamento e modularidade e conseguimos otimizar o programa para que conseguisse ser executado com ficheiros com maior número de registos num tempo aceitável.

Conseguimos que todas as queries fossem executadas em tempo útil, sendo que grande parte do tempo total de execução é devido ao parsing dos ficheiros (fenómeno explicado pelo grande tamanho dos ficheiros). Este é o maior ponto que pode ser melhorado, possivelmente através de uma melhor gestão de memoria, tendo sido esta, tambem, a nossa maior dificuldade.

Em suma, conseguimos implementar todas as funcionalidades propostos para este guião, e acrescentar algumas também, e obter mais conhecimento e proficiência com a linguagem C e alguns conceitos chaves da programação. E apesar de o projeto não estar perfeito, sentimos-mos bastantes satisfeitos com o resultado final.