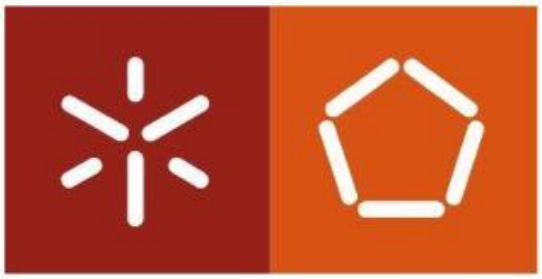


UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

MIEI/ LEI+MEI

UC Laboratórios de Informática III - 2021/2022 | Grupo 11

David Duarte (A93253)

João Castro (A97223)

Conteúdo

Introdução

O projeto (guião 3) do âmbito da UC de Laboratórios de Informática III tinha como principais objetivos apresentar os dados, previamente obtidos dos outros guiões, de uma forma visualmente mais apelativa, apresentar e analisar os testes de desempenho ao nosso *software*, bem como realizar uma otimização na gestão de dados, de modo a que fosse possível receber ficheiros de dados com tamanhos maiores, e portanto, que os catálogos fossem capazes de gerir essa mesma quantidade de dados.

O maior desafio com que nos deparámos foi realizar uma gestão de memória a dados mais volumosos, no qual será posteriormente abordado. De modo, a se obter uma visualização de resultados mais apelativa, foi decidido implementar um menu interativo, com um sistema de paginação para dados de maiores dimensões. Por fim, um outro desafio era apresentar testes de desempenho a cada query, assim como, comparar os seus *outputs* com os dos ficheiros gerados do guião 2 (expected_files), que serão os nossos ficheiros de referência.

Desenvolvimento

2.1. Interface

De modo a facilitar a interação utilizador-programa, neste guião, propôs-se desenvolver um menu interativo. A interface que desenvolvida procura apresentar os dados obtidos pelo processo de encapsulamento nas *queries*.

Desta forma, implementou-se um menu simples com uma utilização prática, onde se encontram disponibilizadas as várias opções que fazem responder a respetiva *query*. Este menu interativo é apenas disponibilizado quando só se executa o programa se nenhum argumento extra `$ ./guiao-3`, como por exemplo, o ficheiro de comandos (*commands.txt*). De maneira a se obter um menu com uma interface mais sofisticada decidiu-se navegar pelas opções através da ocorrência do evento de pressionar a seta para cima e a seta para baixo. A interface do programa é a que se apresenta na imagem inferior.

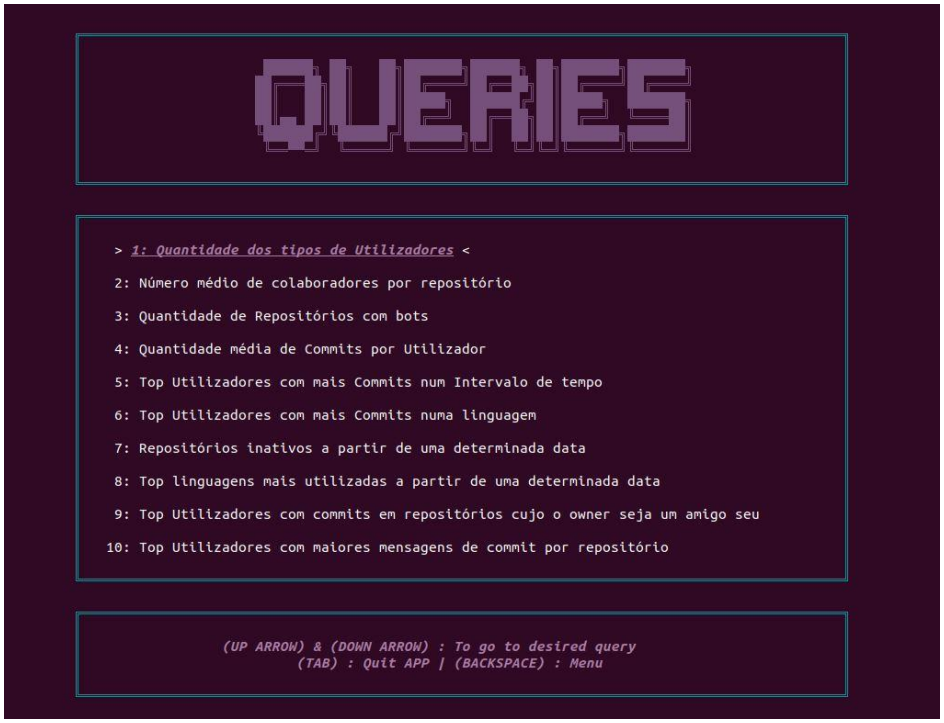


Imagem 1.1 – Menu Interativo com listagem de funcionalidades

Desta forma, a navegação pelo menu de queries através das teclas de seta para cima e seta para baixo, assim como a tecla espaço para voltar ao menu quando se estiver a visualizar uma das opções. Para implementar estas funcionalidades inicializou-se um buffer que armazena o código da tecla pressionada. Dependendo do número *ASCII*, adicionou-se uma componente *integer* que é aumentada quando a seta para baixo é pressionada e decrementada na seta para cima, ainda outra componente ***q*** caso o *'ENTER'* seja pressionado. Assim, consoante o valor destes inteiros, uma dada *query* seria invocada, e consequentemente, os dados seriam visíveis no terminal.

Quando o menu é carregado, o conjunto de catálogos seria inicializado, de modo a que o trabalho de *parsing* dos ficheiros de entrada seja realizado. Posteriormente, os dados são acedidos pelas *queries* através do encapsulamento. A *struct SGR* (Sistema de Gestão de Repositórios) contém os três catálogos: *users*, *commits* e *repositories,* pelo que quando estes estão a ser carregados é disponibilizado no terminal o processo de *loading* de cada um destes catálogos. O carregamento estará finalizado quando o *parsing* dos ficheiros estiver completo através das funções que se encontram em *catalogs_file_work.c,* e quando o processo de validação for realizado por parte do guião 1.

Imagem 1.2 - Loading dos catálogos

Imagem 1.3 - Loading time

Como é possível visualizar na imagem 1.3, o menu interativo só é disponibilizado até à ocorrência de um evento (pressionar a tecla *'ENTER'*). Por outro lado, também disponibilizamos uma informação adicional, o tempo de carregamento do conteúdo da *struct* SGR. O design dos *outputs* das *queries* é apresentado dentro do retângulo intermédio cujas dimensões são de 93x25. Deste modo, a apresentação dos resultados das *queries* estatísticas não é afetada por estas dimensões, porém nas queries parametrizáveis em que é obtido o top N de utilizadores, a dimensão dos resultados é maior pelo que há a necessidade de criar um sistema de navegação. Desta forma, definie-se como estratégia, criar ficheiros temporários dentro da pasta 'paginacao', estes possuem o conteúdo das *queries* parametrizáveis. Pensou-se que seria mais prudente colocar 19 linhas por página, sendo que foi utilizado como recurso uma lista duplamente ligada, *FTABLE*, para facilitar retroceder ou proceder na página pretendida.

Esta *FTABLE* armazena os nomes dos ficheiros correspondentes a cada página. Assim, caso seja pressionada a seta da esquerda obter-se-ia o nome do ficheiro da página anterior através da função `get_ftable_prev` e através da função `get_ftable_next` o nome do ficheiro da página posterior. Por fim, é feita a leitura do ficheiro temporário fazendo *printf* de cada uma das linhas do ficheiro usando a *fgets*.

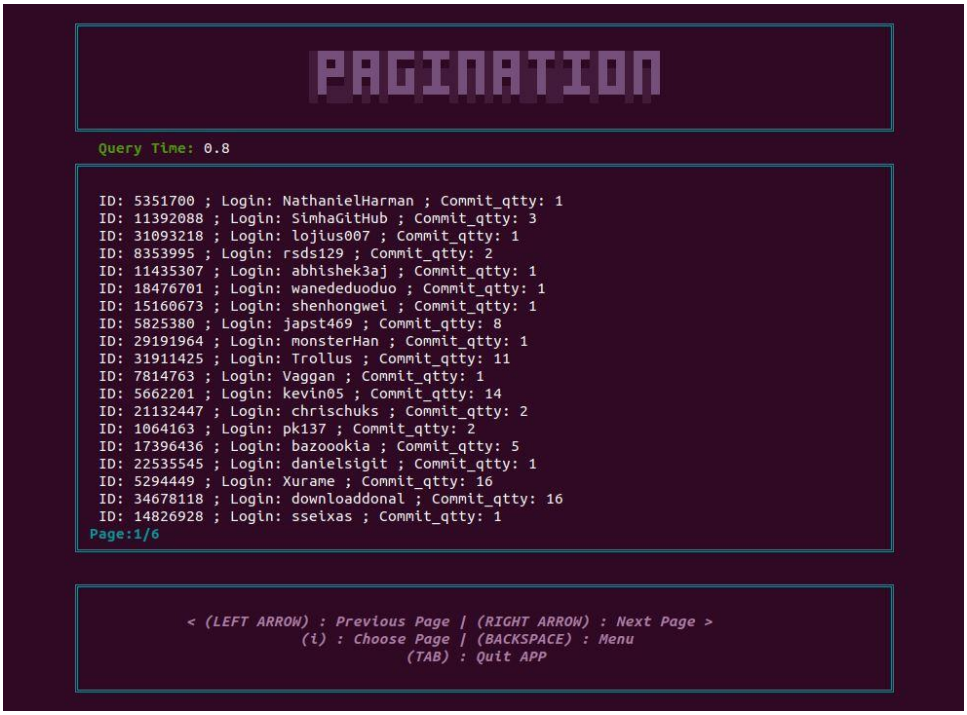


Imagem 1.4 - Exemplo representativo de paginação correspondente à query 5

Como é possível observar na imagem superior a seta direita permite avançar para a página seguinte, enquanto que a seta esquerda retrocede na página. Por outro lado, ao pressionar 'BACKSPACE' retorna-se para o menu principal, sendo que os ficheiros auxiliares são eliminados (o mesmo acontece quando é pressionado TAB, que é utilizado para terminar o programa). Para além disso, ao pressionar a tecla `i` o utilizador pode selecionar a página que pretende visualizar das páginas disponíveis.

2.2. Testes funcionais e de desempenho

No desenvolvimento de um software, assim como em outras áreas da ciência e afins, é importante testar o nosso programa de forma a detetar os possíveis defeitos e efetuar as respetivas melhoras no *software*. Para isso é necessário fazer uma análise minuciosa sobre os dados que obtemos. Através da testagem é possível detetar, por exemplo, um erro que existia na *query* 3 do guião 2, o que demonstra a importância de realizar testes. Nesta componente do projeto decidiu-se implementar um menu para realizar os testes, de forma a tornar o programa o mais intuitivo possível.

Imagem 2.1 - Menu de testes às queries

2.2.1. Resultados obtidos e análise

No software de testagem, o programa espera por um input correspondente ao número da *query* que se encontra disponibilizada no menu. De modo a medir o tempo de execução de cada *query* recorreu-se à biblioteca *time.h,* fazendo assim uso da macro ***CLOCKS_PER_SEC*** para medir a diferença de tempo desde que começa até acabar a respetiva *query*. A baixo, encontram-se alguns tempos de medição de duas amostras das 5 primeiras *queries* deste software.

Dados obtidos por João Castro

Query 1	Query 2	Query 3	Query 4	Query 5
0.00458s	0.00060s	0.00125s	0.00059s	0.00636s
0.00044s	0.00051s	0.00059s	0.00044s	0.00054s
0.00053s	0.00071s	0.00043s	0.00037s	0.00071s
0.00062s	0.00041s	0.00055s	0.00040s	0.00052s
0.00044s	0.00066s	0.00050s	0.00056s	0.00104s
0.00060s	0.00046s	0.00050s	0.00039s	0.00064s
0.00041s	0.00053s	0.00047s	0.00041s	0.00113s
0.00048s	0.00065s	0.00049s	0.00041s	0.00070s
0.00057s	0.00085s	0.00041s	0.00061s	0.00068s
0.00033s	0.00058s	0.00048s	0.00063s	0.00076s

Dados obtidos por David Duarte

Query 1	Query 2	Query 3	Query 4	Query 5
0.00075s	0.00081s	0.00028s	0.00009s	0.00027s
0.00020s	0.00016s	0.00042s	0.00021s	0.00034s
0.00026s	0.00022s	0.00018s	0.00019s	0.00026s
0.00029s	0.00023s	0.00027s	0.00021s	0.00031s
0.00025s	0.00022s	0.00038s	0.00021s	0.00034s
0.00021s	0.00022s	0.00024s	0.00022s	0.00036s
0.00022s	0.00028s	0.00019s	0.00019s	0.00024s
0.00022s	0.00015s	0.00019s	0.00016s	0.00031s
0.00019s	0.00018s	0.00022s	0.00017s	0.00020s
0.00018s	0.00024s	0.00019s	0.00019s	0.00041s

Média	Query 1	Query 2	Query 3	Query 4	Query 5
João Castro	0.000900s	0.000596s	0.000567s	0.000481s	0.001794
Média	Query 1	Query 2	Query 3	Query 4	Query 5
David Duarte	0.000277s	0.000271s	0.000256s	0.000481s	0.000304

Se se determinar o desvio padrão do tempo de desempenho da *query* 5 pode-se realizar uma diferença entre as médias obtidas para as duas amostras relativamente à *query* 5, e assim rejeitar ou não a hipótese nula de existirem diferenças significativas nas duas médias. Para isso recorre-se a uma distribuição normal padrão.

Características da máquina - João Castro

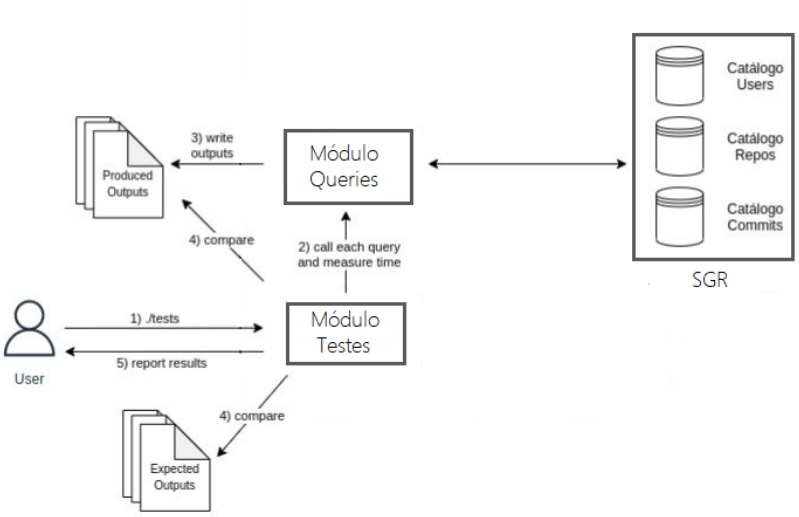
- OS: Ubuntu 20.04.3 LTS 64bit/ RAM: 3.8Gi/ CPU(s):2

Características da máquina - David Duarte

- OS: Ubuntu 20.04 LTS 64bit/ RAM: 4Gi/ CPU(s): 4

Como é possível observar através dos dados apresentados, o tempo de execução das *queries* estatísticas é muito pequeno (aproximadamente zero), visto que, estas apenas têm o trabalho de aceder aos dados em *SGR* e fazer imprimir os mesmos.

Inicialmente, na realização dos testes da comparação dos ficheiros, faz-se a partir da leitura dos dois ficheiros linha a linha, usando comandos como *fopen*(com dois apontadores *FILE*), *fgets* e *fclose*. No entanto, ao longo dos testes o programa dizia que a última linha dos ficheiros era diferente quando na verdade não era. Portanto, optou-se por proceder à comparação caractere a caractere até ao “end of file”, sendo que quando houvesse um caractere diferente, o programa iria informar o utilizador que se realiza os testes, da linha em que existe essa diferença. Utiliza-se como referência, os ficheiros obtidos do guião 2(expectedX_output.txt). O fluxo de testagem, muito bem como as dependências entre os módulos, segue aquele que era pretendido pela equipa docente.



2.3. Gestão de Dados

Em relação à gestão de dados, utilizou-se uma *struct* nomeadamente, *SGR (Sistema de Gestão de Repositórios),* que contém os catálogos de *users*, *commits* e *repos*. Cada catálogo contém a informação necessária para executar as *queries.* Para conseguir obter a informação que se encontra no *SGR*, foram utilizadas funções de *get,* tanto no *SGR* como nos respetivos catálogos. Para colocar informação nesta estrutura de dados utilizou-se as funções de *set*. Esta, gestão não é a melhor, pois para ficheiros com um maior volume de dados, o programa não consegue aguentar tanta informação em memória.

2.4. Ajustes técnicos dos guiões anteriores

Procede-se a algumas modificações nos guiões anteriores, nomeadamente ao nível da *Makefile*. No guião 2 conseguiu-se implementar a *query* 5 com algumas limitações. A estratégia para a mesma passa por contruir uma *hashtable* à medida que era feito o *parsing* do ficheiro de *commits*. Esta *hashtable* pertence à *struct* catalogos_commit e para cada *key*(*author_commit*) faz corresponder o respetivo número de *commits*. Como na biblioteca da *glib* é possível armazenar as chaves numa estrutura de dados(*array*) pode-se verificar quais são os *ids* que possuem um maior número de *commits*, ainda que este processo resulte num "program killed". Por encapsulamento, consegue-se obter o *login* através da função `get_Login`, após obter o respetivo *user* através da função `sgr_get_CU_GUser`.

Por outro lado, corrigiu-se o guião 1, porém o programa como mencionado anteriormente, não aguenta com uma maior *stream* de dados como os que se encontram nos ficheiros do guião 3.

3. Conclusão

3.1. Reflexão

Ao longo deste projeto, encontramo-nos com dificuldades na forma de armazenamento dos dados, assim como na gestão de memória. Por outro lado, o desenvolvimento das *queries* parametrizáveis demonstrou-se difícil, pois não foi eram encontradas formas de conseguir realizar as validações necessárias para apresentar os resultados corretos.

Na realização deste projeto, aprendemos como trabalhar com a *GLIB*, para armazenar informações necessárias. Desenvolvemos também uma maior capacidade para a gestão de memória, no tratamento de ficheiros assim como na utilização do *MVC*, (*model*, *view* e *controller*) utilizado na arquitetura do projeto para uma maior organização do mesmo. Por outro lado, desenvolvemos uma maior capacidade em resolver problemas, conseguimos também desenvolver o nosso espírito crítico face às nossas estratégias e decisões.