

Universidade do Minho  
Laboratórios de Informática III  
Guião III

a96106, Miguel Silva Pinto

a94557, Délio Miguel Lopes Alves

a97613, Pedro Miguel Castilho Martins

# 1 - Introdução

Neste terceiro guião, foi necessário alargarmos as funcionalidades do nosso programa desenvolvido até ao guião 2, com a implementação de um menu interativo que permite a um utilizador procurar e aceder à informação dada pelas queries de uma maneira mais intuitiva. Outra funcionalidade exigida ao programa foi a possibilidade de verificarmos o correto funcionamento do nosso programa, bem como a sua eficácia, através da criação de um executável que verifica o tempo de execução das queries e a validade dos seus resultados.

Para além destas novas funcionalidades, o programa tem de ser capaz de manipular e processar um grande volume de dados, surgindo a necessidade de otimizar certos aspetos do programa para que este seja capaz de responder em tempo útil ao que lhe é pedido pelo utilizador.

## 2 - Solução técnica

### 2.1 - Mecanismo de interação

Inicialmente, começamos por criar o mecanismo de interação onde damos a possibilidade ao utilizador de executar as queries de forma interativa e de forma a visualizar o seu resultado com fácil consulta e navegação dos resultados.

Primeiramente ao executar o programa são feitas as preparações para a execução das queries, como a filtragem dos ficheiros de entrada para apenas conter as linhas válidas e a organização desses ficheiros nos seus respetivos catálogos. Com os dados organizados, printamos a tabela onde mostramos ao utilizador uma pequena descrição de cada query.

1	Quantidade de bots, organizações e utilizadores	
2	Número médio de colaboradores por repositório	
3	Quantidade de repositórios com bots	
4	Quantidade média de commits por utilizador	
5	Top N de utilizadores mais ativos num determinado intervalo de datas	
6	Top N de utilizadores com mais commits em repositórios de uma determinada linguagem	
7	Repositórios inativos a partir de uma determinada data	
8	Top N de linguagens mais utilizadas a partir de uma determinada data	
9	Top N de utilizadores com mais commits em repositórios cujo owner é um amigo seu	
10	Top N de utilizadores com as maiores mensagens de commit por repositório	
0	Sair do programa	
Insira opção: <input type="text"/>		

O programa depois aguarda o input do utilizador e executa a respetiva query com os argumentos que o utilizador decidir usar, no caso de se tratar duma query parametrizável.

Para apresentar os resultados das queries criamos um novo módulo **printer.c** com as funções que permitirão essa apresentação. Primeiramente processamos o ficheiro de output na pasta "saida", criado pela execução da query, analisando os valores dos resultados e inserindo-os num determinado formato de string, correspondente à apresentação dos resultados da respetiva query, através da função **sprintf**. Após ter a linha que irá ser imprimida no formato desejado, a string é inserida num array de strings em que cada elemento deste array representará uma das linhas a imprimir como resultado final. O número de linhas imprimidas por página é determinado através da variável global ao módulo **NUM\_PAG**, uma vez que a função **printPag**, responsável pela impressão das linhas de uma página, tem em conta esse valor na altura de imprimir as linhas de uma determinada página de resultados.

Tendo como exemplo a execução da query 5, com os respetivos argumentos, o utilizador encontra o seguinte sub-menu.

```
Insira opção: 5
Insira número de utilizadores: 100
Insira a data inicial (YYYY/MM/DD): 2014/01/01
Insira a data final (YYYY/MM/DD): 2016/01/01

-----
1º -> ID:25906 ; Login:iant ; N° de commits:30
2º -> ID:30477 ; Login:samps ; N° de commits:30
3º -> ID:42846 ; Login:ootput ; N° de commits:30
4º -> ID:55055 ; Login:deepb ; N° de commits:30
5º -> ID:57126 ; Login:adamdunford ; N° de commits:30
6º -> ID:58623 ; Login:mdhorton ; N° de commits:30
7º -> ID:66361 ; Login:mqh444 ; N° de commits:30
8º -> ID:66637 ; Login:Dabg ; N° de commits:30
9º -> ID:70310 ; Login:guillem ; N° de commits:30
10º -> ID:84009 ; Login:tkutsenko ; N° de commits:30
11º -> ID:92002 ; Login:cctalbott ; N° de commits:30
12º -> ID:94545 ; Login:mkopit ; N° de commits:30
13º -> ID:109147 ; Login:dgschwind ; N° de commits:30
14º -> ID:113388 ; Login:koepfw ; N° de commits:30
15º -> ID:127869 ; Login:deakons ; N° de commits:30
16º -> ID:131451 ; Login:jsevilleja ; N° de commits:30
17º -> ID:137285 ; Login:r1m ; N° de commits:30
18º -> ID:180058 ; Login:Salat39 ; N° de commits:30
19º -> ID:193140 ; Login:shirleyman ; N° de commits:30
20º -> ID:237415 ; Login:petervdpas ; N° de commits:30
-----

-----Página 1 de 5-----
P -> Próxima
A -> Anterior
S <N> -> Saltar para página
Q -> Sair da query

Opção: █
```

Para navegar pelas várias páginas, o utilizador dispõe de uma lista de opções apresentadas na parte inferior, podendo ir para a próxima página, para a anterior ou saltar diretamente para uma página específica. Se o utilizador desejar deixar de ver estes resultados e passar para outra query, pode fazê-lo com a letra Q, reaparecendo a lista de opções das queries, podendo escolher a que pretender executar de seguida.

## 2.2 - Testes funcionais e de desempenho

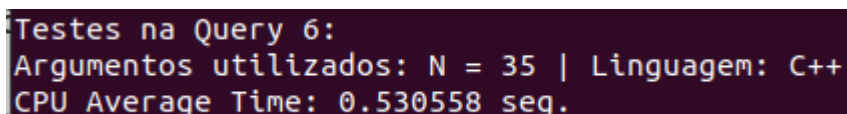
Com o módulo do menu concluído, começamos a desenvolver o módulo de testes que nos permitirá testar o tempo de execução das queries e validar o seu correto funcionamento.

De forma a testar a eficácia de diferentes inputs das queries com uma maior flexibilidade, desenhamos o executável de testes de maneira a receber como argumento o nome de um ficheiro de texto com os comandos a testar, bastando alterar a composição do ficheiro para testar diferentes inputs.

Para avaliarmos o desempenho de cada uma das queries, executamos cada uma das queries várias vezes e calculamos o tempo que cada execução demora em média. O número de testes feitos é controlado pela variável global do módulo **NUM\_TESTS** permitindo alterar facilmente o número de execuções que queremos que sejam efetuadas. Para calcular o tempo utilizamos a função **clock()** presente na library “**time.h**”, guardando o valor da função nas variáveis **start** e **end**, calculando a sua diferença e dividindo-a pelo número de ciclos do clock por segundo (**CLOCKS\_PER\_SEC**) para obter o tempo de execução em segundos.

Para calcular a média entre os tempos de execução de cada query fomos, a cada iteração, somando os tempos das várias execuções e guardando em duas variáveis **max** e **min** o tempo máximo e mínimo para depois subtrair estes valores ao tempo total das execuções. De seguida dividimos esse valor total pelo número de vezes que a query foi executada menos 2, correspondendo às execuções que demoram mais e menos tempo.

A imagem seguinte demonstra o aspeto do resultado final do teste de tempo à query 6 juntamente com os argumentos utilizados.



```
Testes na Query 6:  
Argumentos utilizados: N = 35 | Linguagem: C++  
CPU Average Time: 0.530558 seg.
```

Para os testes funcionais foi criado outro módulo que compara os ficheiros output de cada query com os ficheiros localizados na pasta **expectedFiles** que contém os resultados que devem ser esperados. Para a comparação dos ficheiros criamos uma função **compareFiles** que compara dois ficheiros, o ficheiro output com o ficheiro esperado, e verifica se são iguais. No caso de um carácter ser diferente essa função indica a linha onde os ficheiros são diferentes permitindo-nos identificar com mais facilidade onde ocorreu o erro.

A imagem seguinte demonstra como são apresentados os testes funcionais. Neste exemplo a query 1 foi executada de forma correta enquanto na execução da query 2 foi detetado um erro na linha 2.

```
Testes Funcionais:  
A Query 1 foi executada de forma correta  
A Query 2 não foi executada de forma correta, sendo encontrado uma diferença na linha 2
```

## 3 - Resultados e discussão

### 3.1 - Otimização

Com novos ficheiros de entrada do guião 3 foi necessário fazer mudanças em várias queries pois com o maior volume de dados, o tempo de execução de algumas queries deixavam de apresentar tempos aceitáveis.

#### 3.1.1 - Queries estatísticas

Os tempos de execução de algumas queries estatísticas, apesar do maior volume de dados dos novos ficheiros, continuaram a ser executadas em tempo aceitável como é o caso das queries 1, 2 e 4.

No entanto a query 3 ficou afetada pelo maior número de dados do ficheiro de *users* porque para o seu funcionamento era necessário percorrer todos os *users* e coletar os ids dos “**Bots**”. Para contornar esse problema e agilizar o processo, foi criado um ficheiro de resultados intermédios onde na primeira execução da query os ids dos bots são guardados em ficheiro na pasta **tmp**. Assim nas próximas execuções da query não é necessário percorrer todos os *users* de novo para obter os ids dos bots.

Outra otimização na query 3 foi a procura no array de ids dos bots. Inicialmente estávamos a fazer uma procura linear pelo o array de ids (sendo este array

obtido através do ficheiro intermédio), porém reparamos que era possível reduzir o tempo de pesquisa, ordenando o array e fazendo uma procura binária.

Antes de implementarmos as otimizações, a query era executada, por média, em 2 segundos. (Nota: todos os tempos apresentados foram medidos através do cálculo da média de 10 execuções, removendo a execução mais e menos demorada.)

```
Testes na Query 3:  
CPU Average Time: 2.122057 seg.
```

Após as otimizações, notou-se uma redução significativa no tempo médio de execução, passando a query a ser executada em 0.44 segundos.

```
Testes na Query 3:  
CPU Average Time: 0.441621 seg.
```

### 3.1.2 - Queries parametrizáveis

Antes das otimizações, reparamos que o tempo de execução de algumas das queries parametrizáveis demoravam uma quantidade de tempo considerável, chegando a ultrapassar os 30 segundos dependendo dos argumentos. As queries críticas eram a query 5 e 6, apresentando um tempo não aceitável, sendo superior a 5 segundos.

```
CPU Average Time used by Query5: 31.214657 seg.  
CPU Average Time used by Query6: 6.520478 seg.
```

Para a query 5, de forma a obter um resultado melhor em termos de tempo de execução, decidimos recorrer à criação de um ficheiro que guarda os ids, juntamente com a respetiva data de commit, ordenados relativamente à data de commit. Desta forma, após uma primeira execução da query 5, este ficheiro permite fazer uma procura das datas contidas no intervalo requisitado pelo utilizador de uma maneira mais rápida, sem sermos obrigados a percorrer todos os *commits* e verificando-os individualmente se pertencem àquele intervalo ou não. Uma desvantagem deste método é um pequeno aumento de tempo gasto na execução da query pela primeira vez devido à criação deste ficheiro intermédio, mas este aumento é compensado pela maior rapidez com que futuras chamadas desta query poderão ser executadas uma vez que o ficheiro com os valores organizados já foi criado.

Outra melhoria que permitiu melhorar significativamente o tempo da query 5, foi a alteração das estruturas de dados auxiliares onde guardamos o id de um

commiter associado com o seu número de *commits* no determinado intervalo de datas, para posterior ordenação relativamente ao nº de *commits*. Antes utilizávamos listas ligadas para guardar estes valores, usando um algoritmo similar ao **BubbleSort** para ordenar a lista relativamente ao nº de *commits* para futura apresentação de resultados. Ao mudar de estrutura auxiliar para um array de structs contendo as mesmas informações (id e respetivo nº de *commits*), permitiu-nos usar a função **qsort** da biblioteca “**stdlib.h**” que ordena as structs utilizando o algoritmo **quickSort**, passando a gastar muito menos tempo para ordenar estes valores. Esta mudança de estrutura de dados auxiliar foi a principal causa de melhoria de tempo da execução da query 5, na qual antes das otimizações, com o intervalo de datas (2016/06/23 - 2018/01/12) a query demorava, em média, 31.2 segundos passando a demorar apenas 0.5 segundos com os mesmos argumentos.

Quanto à query 6, sem otimizações e com determinados argumentos, chegava a demorar 144 segundos.

```
Testes na Query 6:  
Argumentos utilizados: N = 35 | Linguagem: C++  
CPU Average Time: 144.908025 seg.
```

Para melhorar esta performance, adotamos a mesma estratégia utilizada na query 5 de mudar de estrutura de dados auxiliar que guarda o id associado com o nº de *commits* (em repositórios com uma determinada linguagem) para ser ordenada relativamente ao nº de *commits*. Em vez de listas ligadas para guardar estes valores, utilizamos um array de structs permitindo uma ordenação mais rápida com o uso do algoritmo **quickSort** utilizado pela função **qsort**. Como o número de elementos distintos nestas estruturas de dados tende a ser mais elevado nesta query, a importância de usar um algoritmo que ordena mais rapidamente estes elementos é ainda maior, evidenciando-se a significativa diferença notada nos tempos de execução antes e depois da melhoria.

```
Testes na Query 6:  
Argumentos utilizados: N = 35 | Linguagem: C++  
CPU Average Time: 0.528987 seg.
```

Nas restantes queries não houve diferenças significativas na sua implementação, uma vez que já apresentavam um tempo de execução aceitável. Apenas alteramos as estruturas auxiliares (similarmente às alterações

anteriores), mas sem grandes melhorias impactantes uma vez que o número de elementos distintos nestas estruturas tende a ser menor, sendo menos impactante um melhor algoritmo que as ordena mais eficazmente.

### 3.2 - Resultados

Na seguinte tabela estão os resultados dos testes de desempenho feitos por cada elemento do grupo e os argumentos utilizados. Todos os tempos apresentados foram medidos através do cálculo da média do tempo de 10 execuções, excetuando a execução mais e menos rápida.

Os valores na tabela estão representados em segundos.

Query	Miguel	Délio	Pedro
<b>1</b>	0.000049	0.000309	0.000020
<b>2</b>	0.141882	0.155203	0.075351
<b>3</b>	0.376228	0.319561	0.214635
<b>4</b>	0.000035	0.000318	0.000016
<b>5</b> N=50 Data Inicial: 2005/02/23 Data Final: 2019/01/12	0.854809	1.025495	0.609242
<b>6</b> N=35 Linguagem: C++	0.909809	0.598263	0.500801
<b>7</b> Data: 2016/12/20	1.090785	0.965910	0.697379
<b>8</b> N=40 Data: 2010/04/20	0.438294	0.294070	0.298361
<b>9</b> N=50	0.191140	0.198123	0.114439
<b>10</b> N=20	0.910134	0.836481	0.646424



Na seguinte tabela estão as especificações dos ambientes de execução de cada elemento do grupo.

Especificações	Miguel	Délio	Pedro
Sistema Operativo	Ubuntu 20.04	Windows 11	Ubuntu 20.04
CPU	Intel Core i7-10510U	AMD Ryzen 7 4700U	Intel Core i7-1165G7
Memória	16 GB	16 GB	16 GB
Disco	1 TB SSD	512 GB SSD	1 TB SSD

## 4 - Conclusão

Para finalizar gostaríamos de dizer que estamos satisfeitos com os resultados finais produzidos pelo nosso código. Consideramos que o nosso mecanismo de interação consegue apresentar de forma visualmente apelativa os resultados pedidos e os nossos tempos de execução das queries apresentam-se todos abaixo da marca limite dos 5 segundos, como apresentado na tabela.

Apesar de estarmos contentes com o nosso trabalho, reconhecemos que ainda haveria espaço para melhoria em alguns aspetos do projeto. Um exemplo é a nossa query 10 que não apresenta o output totalmente correto relativamente ao que é pedido no guião 2, uma vez que decidimos que deveríamos empenhar-nos mais nos novos mecanismos a implementar no guião 3.