



UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

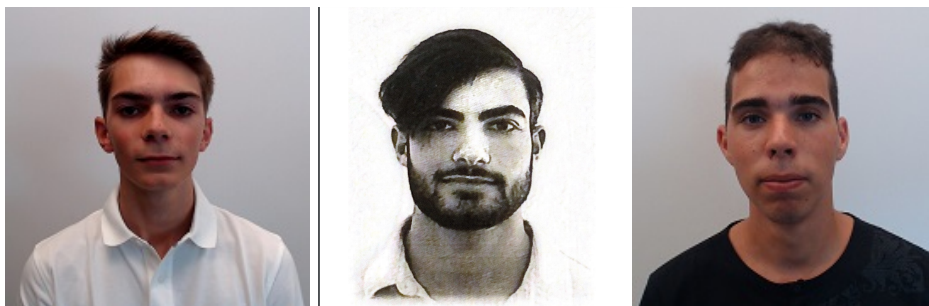
Laboratórios de Informática III

Guião II

Grupo 28

Rui Monteiro (A93179) Rodrigo Rodrigues (A93201)
Daniel Azevedo (A93324)

Ano Letivo 2021/2022



Conteúdo

1	Introdução	3
2	Arquitetura da Aplicação	4
2.1	Parsing dos dados	4
2.2	Interpretação dos comandos	4
2.3	Catálogo de Utilizadores	6
2.4	Catálogo de Repositórios	6
2.5	Catálogo de Commits	6
3	Exercícios	7
3.1	Queries estatísticas	7
3.1.1	QUERY 1: Quantidade de <i>bots</i> , organizações e utilizadores	7
3.1.2	QUERY 2: Número médio de colaboradores por repositório	7
3.1.3	QUERY 3: Quantidade de repositórios com <i>bots</i>	7
3.1.4	QUERY 4: Quantidade média de <i>commits</i> por utilizador	8
3.2	Queries parametrizáveis	8
3.2.1	QUERY 5: Top N utilizadores mais ativos num determinado intervalo de datas	8
3.2.2	QUERY 6: Top N de utilizadores com mais <i>commits</i> em repositórios de uma determinada linguagem	8
3.2.3	QUERY 7: Repositórios inativos a partir de uma determinada data	9
3.2.4	QUERY 8: Top N de linguagens mais utilizadas a partir de uma determinada data	9
3.2.5	QUERY 9: Top N de utilizadores com mais <i>commits</i> em repositórios cujo <i>owner</i> é um amigo seu	9
3.2.6	QUERY 10: Top N de utilizadores com as maiores mensagens de <i>commit</i> por repositório	10
4	Resultados	11
5	Conclusão	13

1 Introdução

Este documento é um relatório técnico acerca do Guião 2 do Trabalho Prático de Laboratórios de Informática III. Esta fase do trabalho tem como objetivos principais a consolidação do uso de ferramentas essenciais ao desenvolvimento de projetos em C e consolidação de conhecimentos adquiridos nesta Unidade Curricular como modularidade, encapsulamento, estruturas dinâmicas de dados e medição de desempenho.

Neste relatório apresentamos as Estratégias utilizadas para resolver os problemas propostos (nomeadamente as Queries), a estruturação da arquitetura da aplicação e todos os requisitos impostos no enunciado do trabalho prático.

Palavras-Chave: **Modularidade, Encapsulamento, Estruturas Dinâmicas de Dados, Queries/Funcionalidades**

2 Arquitetura da Aplicação

2.1 Parsing dos dados

Parte do código em que é realizada a leitura e armazenamento de dados dos 3 ficheiros.

Para tal foram desenvolvidas as seguintes funções:

- **load_users**: carrega o ficheiro de users e insere-os, um a um, numa HashTable.
key = userID
value = GH_USER u
- **load_repos**: carrega o ficheiro de repos e insere-os, um a um, numa HashTable.
key = reposID
value = GH_REPOS r
- **load_commits**: carrega o ficheiro de commits e insere-os numa HashTable cuja chave é *reposID* e o valor é um GArray de commits.
key = reposID
value = (Array de GH_COMMIT)
- **load_sgg**: dado o caminho para os 3 ficheiros, carrega os catálogos de users, commits e repos.

2.2 Interpretação dos comandos

O interpretador de comando corresponde ao código responsável por ler o ficheiro de comandos, interpretar e executar a respetiva query.

Para tal foram desenvolvidas as seguintes funções:

- **build_query**: realiza o parsing de um comando e armazena os argumentos numa struct gh_query.
- **interpretador**: invoca a função responsável por resolver uma query.
- **lerFicheiroQueries**: carrega o ficheiro de comandos e, para cada comando, invoca a função build_query que "constrói" uma query, e invoca o interpretador para executar a query.

```

void interpretador(GH_QUERY q, SGG sgg, FILE *f){
    switch (q->numero) {
        case 1:
            query1(f);
            break;
        case 2:
            query2(sgg, f);
            break;
        case 3:
            query3(f);
            break;
        case 4:
            query4(sgg, f);
            break;
        case 5:
            query5(sgg, q->N, q->data_inicio, q->data_fim, f);
            break;
        case 6:
            query6(sgg, q->N, q->language, f);
            break;
        case 7:
            query7(sgg, q->data_inicio, f);
            break;
        case 8:
            query8(sgg, q->N, q->data_inicio, f);
            break;
        case 9:
            query9(sgg, q->N, f);
            break;
        case 10:
            query10(sgg, q->N, f);
            break;
        default:
            printf("ERROR! Ocorreu um erro na leitura do ficheiro das query!\n");
    }
}

```

2.3 Catálogo de Utilizadores

De modo a armazenar os utilizadores, utilizou-se uma **GHashTable**, sendo a sua **chave** o identificador do utilizador e o **valor** associado *GH_USER* u que é um apontador para *struct gh_user* que contém armazenada todas as informações de um user, desde o login até o número de repositórios.

2.4 Catálogo de Repositórios

De modo a armazenar os repositórios, utilizou-se uma **GHashTable**, tendo como **chave** o identificador do repositório e **valor** associado *GH_REPOS* r que é um apontador para *struct gh_repos*, que possui as informações na íntegra de um repositório como, por exemplo, o identificador do dono do repositório ou a linguagem.

2.5 Catálogo de Commits

De modo a armazenar os commits utilizou-se uma **GHashTable** cuja **chave** é o identificador do repositório e o valor associado é um **GArray** que contém os commits feitos nesse repositório.

3 Exercícios

3.1 Queries estatísticas

De modo a otimizar o desempenho da aplicação, as funcionalidades relativas às queries estatística foram implementadas nas funções de **load** dos ficheiros.

3.1.1 QUERY 1: Quantidade de *bots*, organizações e utilizadores

As quantidades de bots, organizações e utilizadores são calculadas e armazenadas em 3 variáveis globais do módulo de users durante o parsing do ficheiro de utilizadores. O incremento das variáveis é feito conforme a avaliação do tipo de cada user que é carregado, sendo que o valor de cada uma das variáveis contém a quantidade de cada tipo.

O módulo das Estatísticas pode aceder as variáveis através das funções *get_bot*, *get_organization*, *get_user*.

3.1.2 QUERY 2: Número médio de colaboradores por repositório

Começamos por calcular o número de colaboradores, ou seja, utilizadores que sejam autores ou que tenham feito commits para o repositório. Este cálculo é feito com o auxílio de uma **GHashTable** em que as **chaves** são o identificador do utilizador. Visto que só é necessário saber o número total de colaboradores, calcula-se o tamanho da **GHashTable** referida anteriormente, guardando-o numa variável global.

Por fim calcula-se a média dividindo o número de colaboradores pelo tamanho da **GHashTable** do repositórios.

3.1.3 QUERY 3: Quantidade de repositórios com *bots*

A quantidade de repositórios com bots é armazenada com o auxílio de uma variável global do módulo de commits, esta é calculada durante o carregamento do ficheiro de commits, verificando se os colaboradores do commit são bots, se assim for o repositório é adicionado à uma **GHashTable**.

Depois de se realizar o parsing de todos os commits é calculado o tamanho da **GHashTable**, ficando armazenada na variável global referida anteriormente.

Este valor pode ser obtido por outros módulos através da função *get_Repos_with_Bots*.

3.1.4 QUERY 4: Quantidade média de *commits* por utilizador

A quantidade média de commits por utilizador é obtida calculando o número de commits, para isso é usado uma variável global que vai sendo incrementada paralelamente ao parsing dos commits, posteriormente é calculado o tamanho da hashtable que contém os utilizadores. Finalmente é calculada a média dividindo o primeiro valor pelo segundo.

3.2 Queries parametrizáveis

3.2.1 QUERY 5: Top N utilizadores mais ativos num determinado intervalo de datas

Estratégia utilizada:

- Inicialmente, começamos por percorrer a HashTable dos commits (**chave:** *idRepositorio*) e averiguar, como o auxílio da função *posterior*, se a data de cada um dos commits se encontra dentro do intervalo dado como input na execução da query.
- Armazenar numa HashTable com **chave:** *idCommitter* e **valor:** *inteiro*, os *idCommitter* que realizaram commits dentro do intervalo de tempo. O valor é o número de commits realizados pelo utilizador identificado pela chave, este é incrementado sempre que se encontra mais um commit, do mesmo utilizador, que se encontre dentro do intervalo de data.
- Por fim, usa-se a função *topN* que obtém da HashTable as N chaves (*idCommitter*) com maior números de commits (maior valor), armazenando num array com N posições.

3.2.2 QUERY 6: Top N de utilizadores com mais *commits* em repositórios de uma determinada linguagem

Estratégia utilizada:

- Percorrer a HashTable dos commits (**chave:** *idRepositorio*) e averiguar se a linguagem do repositório é a mesma que foi passada como argumento à query.
- Armazenar numa HashTable com **chave:** *idCommitter* e **valor:** *inteiro*, os *idCommitter* que realizaram commits da linguagem em questão. O valor é o número de commits realizados pelo utilizador identificado por *idCommitter* e incrementa sempre que se encontra mais um commit da linguagem em questão realizado por este id.
- Função *take.top* que obtém dessa HashTable as N chaves (*idCommitter*) com maior número de commits (maior valor).

3.2.3 QUERY 7: Repositórios inativos a partir de uma determinada data

Estratégia utilizada:

- Percorrer a HashTable dos commits (**chave:** *idRepositorio*) e averiguar, para cada repositório, se foram feitos commits em datas posteriores à data pretendida.
- Caso não hajam, o repositório considera-se inativo a partir da data em questão.

3.2.4 QUERY 8: Top N de linguagens mais utilizadas a partir de uma determinada data

Estratégia utilizada:

- Percorrer a HashTable dos repositórios (**chave:** *idRepositorio*) e averiguar se a data de atualização do repositório, *updated_at*, é anterior à data passada como argumento à query.
- Armazenar numa HashTable com **chave** : *linguagem* e **valor**: inteiro, as linguagens utilizadas a partir da data pretendida. O valor é o número de vezes que a linguagem foi utilizada (commits) e incrementa sempre que se encontra um novo commit feito daquela linguagem.
- Função *busca_top* obtém dessa HashTable as N chaves(*linguagem*) com maior número de ocorrências (maior valor).

3.2.5 QUERY 9: Top N de utilizadores com mais *commits* em repositórios cujo *owner* é um amigo seu

Estratégia utilizada:

- Percorrer a HashTable dos commits (**chave** : *idRepositorio*) e averiguar se o Committer é amigo do Owner do repositório, através da função *isFriend*.
- Armazenar numa HashTable com **chave** : *committerID* e **valor**: inteiro, os committerID que têm o ownerID na lista de following e de followers. O valor é o número de commits realizados pelo utilizador identificado por committerID e incrementa sempre que encontra mais um commit neste repositório realizado por este id.
- Função *take_top_query9* que obtém dessa HashTable as N chaves (*committerID*) com maior número de commits (maior valor).

3.2.6 QUERY 10: Top N de utilizadores com as maiores mensagens de *commit* por repositório

Estratégia utilizada:

- Percorrer a HashTable dos commits (**chave** : *idRepositorio*, **valor** : *Array* de Commits) e verificar se cada repositório tem N ou mais commits. Em caso afirmativo, o output é de N utilizadores. Caso contrário, o máximo de linhas de output para esse repositório é o número de commits (entre 0 e N-1).
- Para cada commit, calcula-se o tamanho da sua mensagem e posteriormente percorre-se um array ordenado de acordo com o tamanho da mensagens, com N posições ou com número de commits do repositório, posições.
- Durante a inserção no array verifica-se se o utilizador que realizou o commit já se encontra no mesmo. Se sim, compara-se o tamanho de sua maior mensagem até ao momento com o tamanho da nova mensagem e, se for menor, atualiza-se o tamanho da maior mensagem.
- Após serem tratados todos os commits de um repositório, serão escritos, num ficheiro, os N utilizadores com as maiores mensagens para cada repositório. Caso o repositório contenha menos do que N committers, serão dados como output as maiores mensagens, de forma ordenada, de todos os utilizadores do mesmo.

4 Resultados

Query1

```
guião-2 > saída > comand1_output.txt
1 Bot: 72
2 Organization: 22903
3 User: 404409
```

Query2

```
guião-2 > saída > comand2_output.txt
1 0.51
```

Query3

```
guião-2 > saída > comand3_output.txt
1 225704
```


Query4

```
guião-2 > saída > comand4_output.txt
1 2.00
```


Query5

```
guião-2 > saída > comand5_output.txt
1 7356599; redvinylrobot; 30
2 7442933; AndyAnkrah; 30
3 4724756; chillozz; 30
4 7285124; karbanfeynman; 30
5 1431986; AustinBlackstone; 30
6 3149815; ejavice; 30
7 3265904; B35815; 30
8 1711791; wesleymusgrove; 30
9 1417037; rogerd330; 30
10 214245; boredagainpiston; 30
11 4727657; glametl; 30
12 1330685; atkay; 30
13 8324083; devDesign; 30
14 598285; VizLibrary; 30
15 1265173; walkingmontage; 30
16 1074633; brshewmaker; 30
17 9050495; basiurajobs; 30
18 7270117; hilmanseptian; 30
19 3036564; carlosfernandez; 30
20 5626705; rahmanadrianprasetya; 30
```

Query6

```
guião-2 > saida >  comand6_output.txt
1 2671590; yaronvia; 30
2 55055; deepb; 30
3 3428041; xlyytcy; 30
4 25024138; mangoeatsscicada; 30
5 19216335; reilic; 30
```

Query8

```
guião-2 > saida >  comand8_output.txt
1 None; 59828
2 Java; 23544
3 JavaScript; 23464
4 Python; 14402
5 HTML; 13666
6 PHP; 8829
7 C#; 7695
8 C++; 7129
9 CSS; 6900
10 Ruby; 6108
```

Query9

```
guião-2 > saida >  comand9_output.txt
1 8182699; Folken97410
2 20779782; sousaAnderson
3 5787056; aeryen
4 18280522; faropoulos
5 16578626; kiri93
6 33816406; skyBlush
7 14332891; jeffreyfei
8 5901666; mnpmanuel
9 21248607; juniorfreitas2
10 16857363; Sysnett
```

5 Conclusão

Terminada a implementação do guião 2, consideramos que conseguimos alcançar o objetivo principal: manipular e tratar grandes quantidades de dados.

As maiores dificuldades encontradas pelo grupo durante este guião passaram pela escolha e manipulação de estruturas de dados assim como a implementação de algoritmos capazes de lidar com as grandes quantidades de informações, de modo a que cada uma das queries pudessem ser executadas, usando a menor quantidade possível de recursos computacionais.