

UNIVERSIDADE DO MINHO  
MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA



---

## Gestor de Snapshots Wikipedia

---

### LABORATÓRIOS DE INFORMÁTICA III

GRUPO 61

*Trabalho realizado por:*

*Número*

Bruno Ferreira  
Marco Barbosa  
Sara Pereira

A74155  
A75278  
A73700

1 de Maio de 2017

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Descrição das estruturas criadas para a abordagem do problema</b>	<b>3</b>
2.1	Parser . . . . .	4
2.2	AVL . . . . .	4
2.3	Revisão . . . . .	4
2.4	Catálogo . . . . .	5
2.5	CatArtigos . . . . .	6
2.6	CatContribuidores . . . . .	7
2.7	Contribuidores . . . . .	7
2.8	Set . . . . .	8
2.9	Utils . . . . .	8
<b>3</b>	<b>Interrogações</b>	<b>9</b>
3.1	Interrogação Número 1 . . . . .	10
3.2	Interrogação Número 2 . . . . .	10
3.3	Interrogação Número 3 . . . . .	10
3.4	Interrogação Número 4 . . . . .	11
3.5	Interrogação Número 5 . . . . .	11
3.6	Interrogação Número 6 . . . . .	11
3.7	Interrogação Número 7 . . . . .	11
3.8	Interrogação Número 8 . . . . .	12
3.9	Interrogação Número 9 . . . . .	12
3.10	Interrogação Número 10 . . . . .	12
<b>4</b>	<b>Interface do utilizador</b>	<b>13</b>
<b>5</b>	<b>Performance</b>	<b>15</b>
<b>6</b>	<b>Makefile</b>	<b>16</b>
<b>7</b>	<b>Conclusão</b>	<b>18</b>

# Capítulo 1

## Introdução

No âmbito da unidade curricular de Laboratórios de Informática III foi proposto que se desenvolvesse um sistema que lidasse com uma enorme quantidade de dados relacionados com uma das plataformas de pesquisa mais utilizadas no mundo, a Wikipédia. Este projeto considera-se um grande desafio para nós pelo facto de passarmos a programar em grande escala, uma vez, que o projeto exige lidar com grandes volumes de dados e com uma complexidade algorítmica e estrutural mais elevada. Nesse sentido, o desenvolvimento deste programa será realizado a partir dos princípios da modularidade (divisão do código fonte em unidades separadas coerentes), do encapsulamento (garantia de proteção e acessos controlados aos dados), da conceção de código reutilizável e escolha otimizada das estruturas de dados.

## Capítulo 2

# Descrição das estruturas criadas para a abordagem do problema

Depois de ser analisado o enunciado do problema, percebeu-se que existem dois módulos principais, sendo um deles constituídos por três snapshots da plataforma realizados em datas diferentes e, o segundo módulo trata-se de uma interface de comunicação com o cliente. No entanto, para a realização do trabalho houve a necessidade de criação de várias estruturas para lidar com a informação fornecida pelos snapshots. Desta maneira, a transferência dos dados, que numa primeira fase se encontram em XML, para as dadas estruturas, é feita através de um parser. Abaixo vão ser descritas todos os passos que nos permitiram chegar às respostas das interrogações.

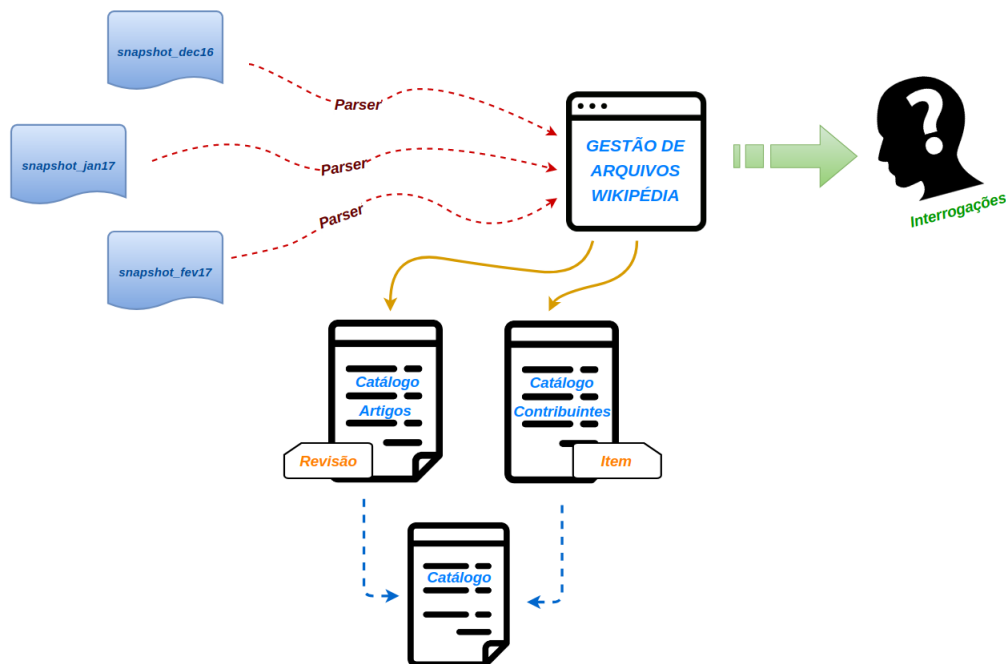


Figura 2.1: Estrutura do trabalho

## 2.1 Parser

Como os snapshots fornecidos que se encontravam em XML, era necessário haver passagem para as estruturas em questão. Desta maneira, foi usada a biblioteca Libxml2 para fazer o parsing dos mesmos. A inserção nas estruturas é realizada através de um array de dados, sendo posteriormente adicionada a informação desse array de dados à estrutura. A inserção é realizada artigo a artigo, ou seja, é inserido um artigo, e o array de dados é depois reutilizado para guardar as informações do artigo seguinte.

## 2.2 AVL

Primeiramente, foi pensada a estrutura que iria abrigar a informação dos artigos, portanto foi decidida que seria uma árvore binária auto-balanceada, com o código generalizado, a fim de respeitar os princípios de modularidade. Esta estrutura está organizada pelo identificador dos artigos. Cada nodo é constituído por um long data (identificador) e um apontador (void\* conteúdo) para uma estrutura que salvaguarda a informação relativa às revisões. No ficheiro avl.c, para além das funções gerais para a criação da árvore, existe ainda a função **void \* getContentAVL(AVL a, long data)** que retorna a estrutura das revisões referentes ao artigo em questão. Uma das vantagens do uso de uma **AVL** é o facto de o tempo de inserção na estrutura ser constante, ou seja, não ser necessário realocar nova memória para toda a árvore, uma vez que a memória é alocada sempre que se pretende inserir um novo elemento. Esta razão foi também conjugada com o facto da procura nas **AVL** ser bastante rápida, visto que só teríamos de procurar num dos ramos da mesma. Esta não é no entanto a melhor estrutura para quando se pretende percorrer toda a árvore, no entanto essa desvantagem foi posteriormente colmatada com a criação da estrutura auxiliar **SET**.

## 2.3 Revisão

A parte atómica das revisões é constituída por uma estrutura denominada **Elemento**. Cada **Elemento** alberga o id (long) da revisão, a data da mesma (char\*), o nome do contribuinte (char\*), o número de bytes do texto (long), o id do contribuinte e o número de palavras do texto (long). Como, para um dado artigo, podem existir diversas revisões, o void\* conteúdo presente nos nodos da **AVL** descrita acima, será um array de **Elemento**. Por isso, existe no ficheiro revisão.c, para além da estrutura **Elemento**, a estrutura deste array. Em cada posição do mesmo, está guardada a informação para o número de revisões efetuadas a esse artigo (long tamanho), assim como a capacidade do mesmo, isto é, o número total de revisões que podem ser adicionadas no array. Para além destes componentes há também o nome do artigo (char\* articleName).

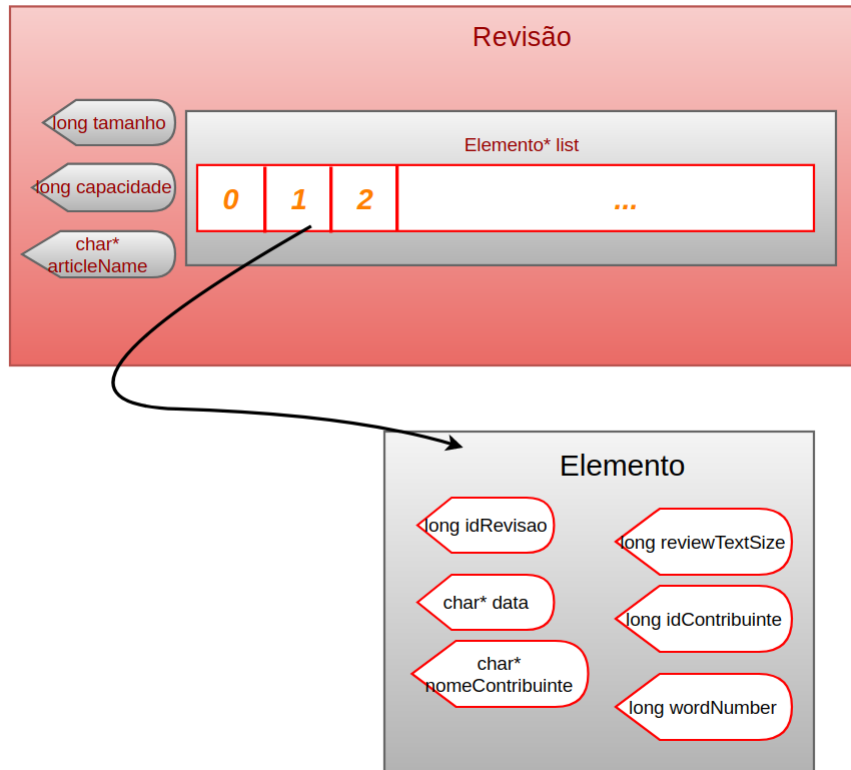


Figura 2.2: Representação da estrutura das revisões

## 2.4 Catálogo

Esta estrutura vai salvaguardar toda a informação tanto para artigos como para contribuidores, sendo esta um array de **AVL**, organizado pelo primeiro dígito do id a inserir. Por exemplo, o id 1024 seria inserido na árvore número um. No ficheiro catalogo.c estão descritas as funções gerais que trabalham com a estrutura **Catalogo**

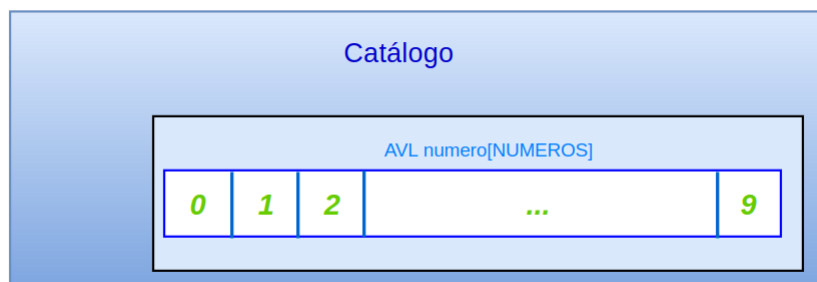


Figura 2.3: Representação da estrutura do catálogo

## 2.5 CatArtigos

Estrutura que guarda toda a informação sobre todos os artigos (Catalogo artigo) e, para além disso, guarda o número de artigos repetidos (long), assim como o número de revisões únicas (long).

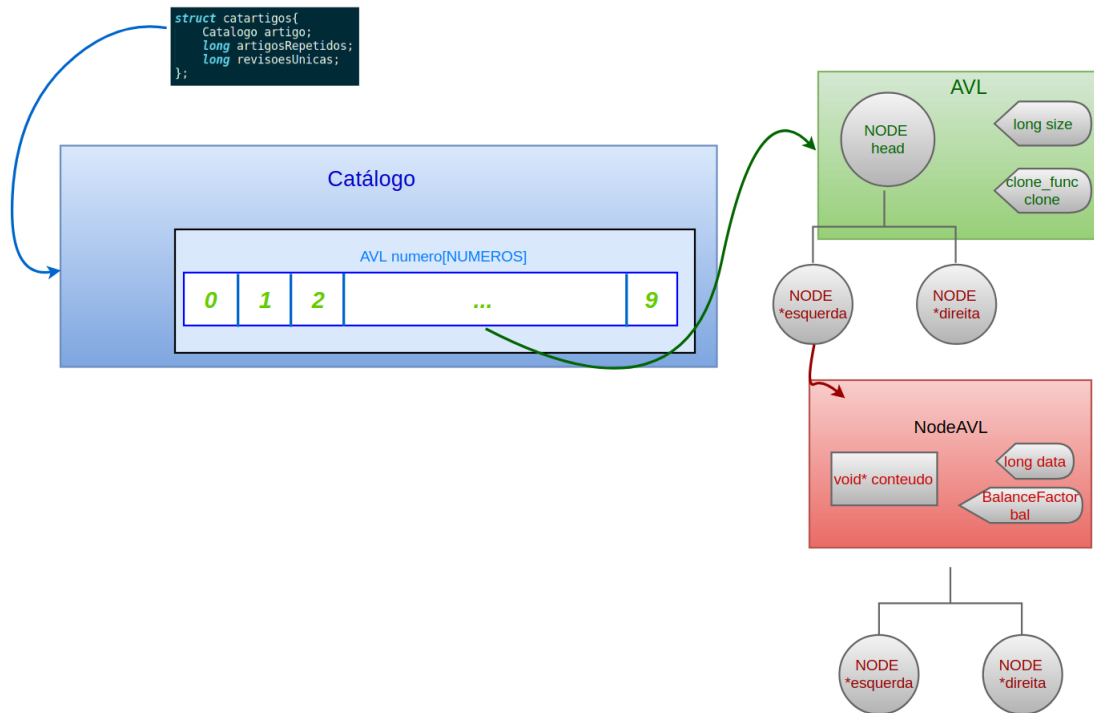


Figura 2.4: Representação da estrutura do catálogo de artigos

## 2.6 CatContribuidores

Estrutura que guarda toda a informação de todos os contribuidores (Catalogo contribuidores).

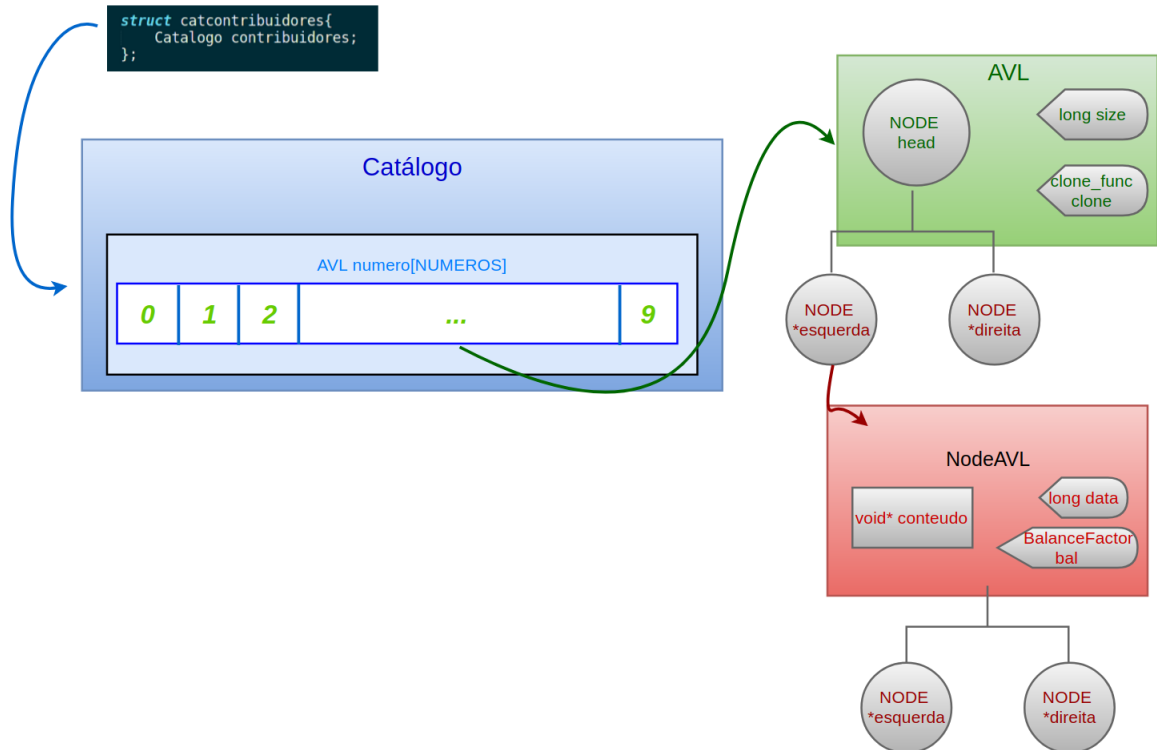


Figura 2.5: Representação da estrutura do catálogo de contribuidores

## 2.7 Contribuidores

Certas interrogações deste projeto pedem especificações de um contribuidor como: dando um identificador e uma estrutura, devolver o nome do respetivo contribuidor. Ou então: dando uma estrutura, devolver um array com os identificadores dos maiores contribuintes. Desta maneira, não só para simplificar a procura, mas também, para aproveitar o módulo da estrutura **AVL**, foi criada uma estrutura **Item** que guarda o nome do contribuidor (`char*`) e o número de contribuições efetuadas pelo mesmo (`long`). As informações guardadas são posteriormente utilizadas para responder as interrogações.



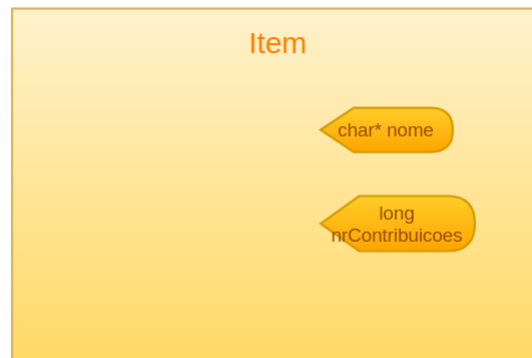


Figura 2.6: Representação da estrutura dos contribuidores

## 2.8 Set

Como referido acima, esta estrutura foi criada como uma adição a uma das vantagens do uso de **AVL**. Este módulo efetua a passagem de qualquer estrutura para um array de identificadores, organizados através de uma função de comparação passada como parâmetro nas funções de construção do array. Ao ser inicializado o **SET** (**SET** **initSet**(**int** **cap**, **boolean** **lck**, **Funcao** **f**)) há hipótese de escolha se este vai ter um carácter dinâmico ou estático, se **lck** tiver como valor **TRUE**, o **SET** resultante é estático. Se **lck** tomar o valor **FALSE**, o **SET** resultante é dinâmico. Esta opção foi tomada como vantajosa na medida em que possibilita a reutilização do **SET** em diversas interrogações.

## 2.9 Utils

Este ficheiro contém algumas funções auxiliares, assim como a definição das funções passadas como argumento.

## Capítulo 3

# Interrogações

Após o desenho e a implementação das estruturas que servem de suporte ao desenvolvimento do programa pretendido, surge agora o momento de tirar partido de tais estruturas e começar a fazer interrogações ao programa, com o intuito de demonstrar o seu nível de praticidade para o problema em questão e também perceber o nível de fiabilidade que ele proporciona.

Sendo assim, podemos definir 4 momentos cruciais que englobam toda a dinâmica de resposta às interrogações:

- Ainda antes de interrogar o programa, é feita uma inicialização de algumas estruturas referenciadas anteriormente neste relatório. Esta inicialização é, essencialmente, o alocamento de memória para o *TCD\_istruct* que, no fundo, corresponde a alocar memória para um catálogo de artigos (*CatArtigos*) e para um catálogo de contribuidores (*CatContribuidores*).

```
TAD_istruct init(){
    TAD_istruct estrutura = malloc(sizeof(struct TCD_istruct));
    estrutura->artigos = initCatArtigos();
    estrutura->contributors = initCatContribuidores();
    return estrutura;
}
```

Figura 3.1: Inicialização do *TCD\_istruct*

- Após a fase de inicialização das estruturas, surge o momento de "carregar" a informação presente nos snapshots para as estruturas. Para que tal feito seja possível, é necessário, numa primeira instância, retirar o formato XML da informação dos snapshots e depois, carregar os dados que lá se encontram para as estruturas adequadas (que já se encontram inicializadas).

```
TAD_istruct load(TAD_istruct qs, int nsnaps, char *snaps_paths[]){
    int i;
    for(i = 0 ; i < nsnaps ; i++){
        parseFile(qs->artigos, qs->contributors, snaps_paths[i]);
    }
    return qs;
}
```

Figura 3.2: Load dos snapshots para as estruturas

- Depois de as estruturas estarem inicializadas e carregadas corretamente com a informação dos backups (ou seja, após o parse dos snapshots), o programa

está apto a responder às interrogações especificadas. A resposta às interrogações envolve a procura de informação nas estruturas criadas através de funções definidas, como por exemplo: *getArtigosRepetidos(...)*, *getRevisoesUnicas(...)*, *getContentCatArtigos(...)*, etc.

- Por fim, quando cessam as interrogações ao programa, surge o momento de libertar o espaço em memória ocupado pela estrutura do catálogo de artigos, pela estrutura do catálogos dos contribuintes e pelo próprio tipo abstrato de dados.

```
TAD_istruct clean(TAD_istruct qs){
    if (!qs)
        return qs;
    freeCatArtigos(qs->artigos);
    freeCatContribuidores(qs->contributores);
    free(qs);
    return NULL;
}
```

Figura 3.3: Free das estruturas

Todos estes momentos do programa enunciados que são necessários para a resposta às interrogações, e também a resposta às próprias interrogações encontram-se implementados no ficheiro *interface.c*.

Neste ficheiro, encontram-se também codificadas todas as interrogações definidas, as quais abordaremos de seguida.

### 3.1 Interrogação Número 1

Esta interrogação procura contar os artigos presentes em todos os snapshots analisados, sabendo que os artigos duplicados em snapshots sucessivos e novas revisões de artigos serão também contabilizados.

**Estratégia:** guardamos o número de artigos repetidos no load dos ficheiros, e posteriormente adicionamos o número de artigos únicos. Este número de repetidos é guardado no *catArtigos*.

### 3.2 Interrogação Número 2

Esta interrogação procura contar os artigos distintos presentes em todos os snapshots analisados, sabendo que apenas são contabilizados os artigos com identificadores distintos, sendo ignorados os artigos duplicados ou revisões dos mesmos.

**Estratégia:** contar o número de elementos presentes na estrutura *catArtigos*, já que este corresponde ao número de artigos únicos uma vez que não são guardados repetidos.

### 3.3 Interrogação Número 3

Esta interrogação procura contar as revisões presentes nos snapshots analisados, retornando apenas o número de revisões únicas.

**Estratégia:** guardamos o número de revisões únicas no *catArtigos* aquando o load dos ficheiros (para aumentar eficiência da resposta), sendo este o número retornado.

### 3.4 Interrogação Número 4

Esta interrogação conta as revisões dos contribuidores, devolvendo os 10 identificadores daqueles que mais revisões fizeram. Note-se que caso hajam contribuidores com o mesmo número de revisões, ordena-os pelo seu identificador de forma crescente.

**Estratégia:** Nesta interrogação utilizamos a estrutura auxiliar *SET* para se poder percorrer toda a nossa estrutura. Para tal é passado como argumento da função de inserção *moveCatContribuidoresToSet(qs-> contributors, contributors, (comp\_func)maxContribuicoes)* a função *maxContribuicoes* que compara o número de contribuições de dois contribuintes. Note-se que não conseguimos corretamente implementar esta interrogação, sendo que consideramos que não é o desenho da solução o problema, mas possivelmente a maneira como se está a fazer a contabilização do número de revisões feita por cada Contribuidor.

### 3.5 Interrogação Número 5

Esta interrogação retorna o nome de um contribuidor correspondente a um dado identificador de contribuidor, sendo que caso não haja um contribuidor com o identificador explicitado, é retornado o valor NULL.

**Estratégia:** Para a resolução desta interrogação, foi feita uma procura do identificador do contribuidor pela estrutura do *catContribuidores*, sendo retornado o apontador para a estrutura com os dados referentes ao contribuidor que apresenta tal identificador (através da função *getContentCatContribuidores(qs->contributors, contributor\_id)*). Depois é apenas retirar o nome do contribuinte da estrutura através da função *getNome(contributor)*.

### 3.6 Interrogação Número 6

Esta interrogação identifica os artigos com maior texto (tamanho em bytes), devolvendo os 20 identificadores daqueles com mais tamanho. Caso hajam artigos com o mesmo tamanho, ordena-os pelo seu identificador de forma crescente.

**Estratégia:** Para a resolução desta interrogação, recorremos novamente à estrutura *SET*, passando no entanto, a função de comparação correspondente, neste caso, a função *compareTextSize*. Esta função compara entre duas revisões, qual das duas tem presente entre as varias revisões do proprio artigo, o maior numero de texto em bytes. Após termos os 20 artigos com maior texto (em bytes) no *SET*, percorremo-lo e vamos passando para um array os identificadores de tais artigos. Note-se que a função que coloca o nosso catálogo de artigos no *SET* é a seguinte *moveCatArtigosToSet(qs-> artigos, artigos, (comp\_func)compareTextSize)*

### 3.7 Interrogação Número 7

Esta interrogação retorna o título de um artigo correspondente a um dado identificador de artigo. Note-se que, caso não haja um artigo com o identificador explicitado, é retornado o valor NULL e caso um artigo tenha várias revisões, é considerado o título da revisão mais recente do artigo.

**Estratégia:** Dado o identificador do artigo, procuramos na nossa estrutura *catArtigos* o artigo com o identificador dado. Obtendo depois o apontador para a estrutura com as informações desse artigo, verificamos o nome do artigo em si.

### 3.8 Interrogação Número 8

Esta interrogação identifica os artigos com maior número de palavras, devolvendo os 'n' identificadores daqueles que apresentam maior contagem de forma crescente. Note-se que caso hajam artigos com o mesmo número de palavras, ordena-os pelo seu identificador de forma crescente.

**Estratégia:** Esta interrogação tem um método de resolução parecida com as outras interrogações de tops. É novamente utilizado um *SET* para guardar as respostas à interrogação, utilizando neste caso a função de comparação *compareWords* que faz a verificação de qual dos artigos passados contém um maior número de palavras, entre todas as revisões do artigo. A função que faz a inserção no *SET* é a seguinte `moveCatArtigosToSet(qs->artigos,artigos,(comp_func)compareWords);`

### 3.9 Interrogação Número 9

Esta interrogação devolve os títulos dos artigos que contêm o prefixo especificado, ordenados alfabeticamente. Note-se que caso os títulos tenham sido alterados em algumas revisões, é considerado o título da revisão mais recente do artigo e que o array de títulos tem de terminar com o elemento NULL.

**Estratégia:** Para esta interrogação foram necessários alguns pequenos ajustes na estrutura *SET*. Para a realização desta interrogação era necessário passar à função de comparação o prefixo que pretendemos testar. Foi portanto necessária a criação das funções auxiliares para a inserção no set, tais que estas recebem de extra o prefixo a testar. Posteriormente, o teste da inserção é feito através da função *isPrefix* que recebe como parâmetros um prefixo e uma *Revisão*, verificando se o título da mesma tem presente o prefixo dado.

### 3.10 Interrogação Número 10

Esta interrogação devolve o timestamp para uma certa revisão/versão de um dado artigo, sendo que caso não haja a revisão para o artigo específico, é retornado o valor NULL.

**Estratégia:** Dado um identificador de artigo e um identificador de revisão, procuramos na estrutura *catArtigos* a existência do artigo com o identificador dado. Se o mesmo não existir é retornado o valor NULL. Pelo contrario, se o artigo estiver presente na estrutura é posteriormente verificado se esse artigo contem nas suas revisões alguma com o identificador dado. Se o mesmo existir é então retornado o timestamp da revisão, caso contrario é retornado o valor NULL.

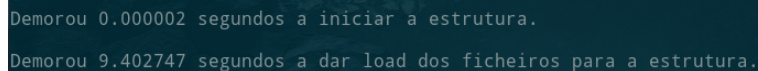
## Capítulo 4

# Interface do utilizador

A execução do programa parte do ficheiro *program.c*, sendo que é neste ficheiro que se demarca o *flow* do programa. De forma a tornar o programa mais perceptível, decidimos criar um menu simplista que tem o objetivo de guiar o utilizador na execução.

Depois de se executar a makefile o programa toma a seguinte execução:

- Em primeiro lugar, o utilizador é informado acerca de quanto tempo o programa demorou a inicializar as estruturas referidas no capítulo anterior, e também quanto tempo o programa levou a fazer o parsing correto da informação contida nos snapshots para as estruturas.

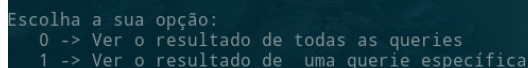


```
Demorou 0.000002 segundos a iniciar a estrutura.  
Demorou 9.402747 segundos a dar load dos ficheiros para a estrutura.
```

Figura 4.1: Tempo de iniciar as estruturas e load

- Depois o utilizador tem 2 opções: ou pretende ver a resposta do programa para as 10 interrogações definidas; ou então pretende ver a resposta para uma interrogação específica.

Note-se que, caso o utilizador deseje ver o resultado de todas as interrogações, estão pré-definidos alguns valores que necessitam de ser usados em algumas interrogações, enquanto que se for vista apenas uma interrogação, a interação com o programa é maior pois o utilizador tem oportunidade de escolher os seus próprios valores: por exemplo, na interrogação número 5, é necessário que seja dado um identificador de contribuinte e, caso o utilizador deseje ver o resultado de todas as interrogações, este identificador tem o valor por defeito de **dwd**, enquanto que se pretender ver apenas esta interrogação, tem a possibilidade de escolher qual o identificador de contribuinte que deseja usar.



```
Escolha a sua opção:  
0 -> Ver o resultado de todas as queries  
1 -> Ver o resultado de uma query específica
```

Figura 4.2: Menu de escolha

```
1 -> Contar os artigos presentes em todos os snapshots analisados
2 -> Contar os artigos distintos presentes em todos os snapshots analisados
3 -> Contar as revisões presentes em todos os snapshots analisados
4 -> Contar as revisões dos contribuidores, devolvendo os 10 identificadores daqueles que mais revisões fizeram
5 -> Retornar o nome de um contribuidor correspondente a um dado identificador de contribuidor
6 -> Devolver os 20 identificadores de artigo com maior texto(tamanho em bytes)
7 -> Retornar o título de um artigo, correspondente a um dado identificador de artigo
8 -> Devolver os 'n' identificadores dos artigos com o maior número de palavras de forma crescente
9 -> Devolver os títulos dos artigos que contêm o prefixo especificado, ordenados alfabeticamente
10 -> Devolver o timestamp para uma certa revisão/versão de um dado artigo

Escolha a sua opção: █
```

Figura 4.3: Menu caso escolha ver uma interrogação específica

- Por fim, independentemente da opção anterior que o utilizador tenha tomado, a execução do programa termina com o *free* das estruturas que foram inicializadas, mostrando o tempo que tal processo demorou.

```
Demorou 0.020711 segundos a dar free da estrutura.

[marco : grupo61]
→ █
```

Figura 4.4: Free das estruturas

## Capítulo 5

# Performance

Serão agora apresentados pequenos testes de performance realizados para verificação de resultados.

Para fazer a verificação dos tempos acima apresentados, foi utilizada a livreria disponível em *c time.h*, através do uso de *clock\_t*.

Para poder verificar se não ocorrem problemas no uso incorreto da memória dinâmica, foi utilizado um programa externo denominado por *Valgrind*. Este programa deteta erros na alocação e libertação de memória, vazamentos de memória e verifica se existem acessos a locais de memória inválidos.

O print posterior tem a informação do output do respetivo programa, indicando que foram realizados 7,745,358 alocações e foram realizadas 7,745,321 libertações. Podemos então verificar que em certos locais do programa existem alocações de memória que não são depois libertadas. Foi tentada a correção destes problemas, tendo no entanto obtido resultados negativos em relação à mesma.

```
==31851==
==31851== HEAP SUMMARY:
==31851==   in use at exit: 20,891 bytes in 37 blocks
==31851== total heap usage: 7,745,358 allocs, 7,745,321 frees, 6,650,649,193 bytes allocated
==31851==
==31851== 640 bytes in 5 blocks are definitely lost in loss record 26 of 29
==31851==   at 0x4C2AF1F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==31851==   by 0x401B5C: titles_with_prefix (in /home/bruno/Projetos/grupo61/program)
==31851==   by 0x4015E4: main (in /home/bruno/Projetos/grupo61/program)
==31851==
==31851== 681 (41 direct, 640 indirect) bytes in 1 blocks are definitely lost in loss record 27 of 29
==31851==   at 0x4C2AF1F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==31851==   by 0x401B30: titles_with_prefix (in /home/bruno/Projetos/grupo61/program)
==31851==   by 0x4015E4: main (in /home/bruno/Projetos/grupo61/program)
==31851==
==31851== LEAK SUMMARY:
==31851==   definitely lost: 681 bytes in 6 blocks
==31851==   indirectly lost: 640 bytes in 5 blocks
==31851==   possibly lost: 0 bytes in 0 blocks
==31851==   still reachable: 19,570 bytes in 26 blocks
==31851==   suppressed: 0 bytes in 0 blocks
==31851== Reachable blocks (those to which a pointer was found) are not shown.
==31851== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==31851==
==31851== For counts of detected and suppressed errors, rerun with: -v
==31851== Use --track-origins=yes to see where uninitialised values come from
==31851== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```

Figura 5.1: Output do programa Valgrind



## Capítulo 6

# Makefile

```
#Compilador a ser utilizado
CC = gcc

#Ficheiros a serem compilados
OBJ_FILES := $(patsubst src/%.c, obj/%.o, $(wildcard src/*.c))

#Flags de compilação passadas ao compilador
CFLAGS = -Wall -std=c11 -O2
PFLAGS = `pkg-config --cflags libxml-2.0`
LIBS = `pkg-config --libs libxml-2.0` `pkg-config --libs glib-2.0`

#Realizar a compilação e linkagem
program: setup $(OBJ_FILES) interface.o
    $(CC) $(CFLAGS) $(PFLAGS) $(LIBS) -o program program.c interface.o $(OBJ_FILES)

#Criar uma pasta obj onde serão guardados todos os ficheiros .o
setup: mkdir -p obj

#Compilar o programa para fazer debug
debug: CFLAGS := -g -O0
debug: program

obj/parser.o : src/parser.c
    $(CC) $(PFLAGS) $(CFLAGS) -o $@ -c $<

interface.o : interface.c
    $(CC) $(PFLAGS) -o $@ -c $<
```

```

#Gerar os ficheiros .o para a pasta obj
obj/%.o: src/%.c
    $(CC) $(CFLAGS) -o $@ -c $<

#Compilar o programa e executá-lo de seguida de forma automática
run: setup $(OBJ_FILES)
    $(CC) $(CFLAGS) $(PFLAGS) -o program $(OBJ_FILES)
    ./program

#Remove a pasta obj, juntamente com os ficheiros .o e o executável
clean:
    -@rm -rf obj
    -@rm program
    -@rm *.o

#Remove os ficheiros .o gerados, a pasta obj, o executável e toda a documentação do código
cleanAll: clean
    -@rm -rf doc
    -@rm -rf html

#Gera a documentação do código
.PHONY: doc
doc: setup $(OBJ_FILES)
    doxygen Doxyfile

```

## Capítulo 7

# Conclusão

As dificuldades encontradas na realização do programa foram o facto do volume de dados analisado ser bastante elevado, levando a um duplo cuidado com a forma como o código é estruturado para permitir uma maior rapidez de execução do programa, o facto da linguagem de programação utilizada permitir a alocação dinâmica de memória, o que embora permita uma melhor gestão da memória utilizada pelo programa, crie problemas adicionais no modo em como se aloca e liberta memória ao longo da execução do mesmo. Como já referido anteriormente não foi possível resolver o problema referente à interrogação 4, ficando no entanto a nota que o problema possivelmente se deve à maneira como se está a fazer a contabilização do número de revisões feita por cada Contribuidor. Após uma análise global acerca das dimensões do problema proposto e da maneira como o programa desenvolvido lida com ele, acreditamos veemente que desenvolvemos uma boa solução, sendo essa capaz de processar o volume de dados apresentado e produzir respostas a interrogações que lhe são colocadas num tempo útil. Por fim, acrescenta-se também que este projeto foi positivo na medida em que nos permitiu continuar a desenvolver a nossa capacidade de programação numa linguagem imperativa que é o *C*, assim como implementar e aprofundar o conhecimento de algoritmos/estruturas abordados noutras unidades curriculares.