Laboratórios de Informática III

Projecto em C

Relatório de Desenvolvimento

João Gomes	Tiago Fraga	Ricardo Silva
a74033	a74092	a60995

30 de Abril de 2017

Conteúdo

Capítulo 1

Introdução

Este trabalho prático tem como objetivo construir um programa que permite analisar artigos presentes em *backups* da *Wikipédia*.

Estes *backups* foram realizados em diferentes meses e com a construção deste programa, pretendemos extrair e organizar informação pertinente para esses períodos de tempo.

Os principais requisitos que nos foram propostos, baseavam-se na obrigatoriedade que este projeto fosse desenvolvido na linguagem de programação **C**, e que e que o código teria de conter um par de ficheiros '.c' e '.h' relativos a uma interface, onde estaria definida a Estrutura principal do trabalho, bem como algumas queries que têm de ser resolvidas pelo nosso grupo.

Nos capítulos seguintes vamos descrever a forma como abordámos o problema, bem como a solução implementada para o resolver.

Capítulo 2

Analise do Problema

De maneira a iniciar o projeto, enfrentamos logo uma situação fulcral para a realização do mesmo, isto é, a escolha das estruturas que íamos utilizar para guardar a informação dos *backups*.

De forma a poder realizar as queries que estão implementadas no ficheiro "interface.c", e ao mesmo tempo ter um bom desempenho no *load* da informação pretendida para as nossas estruturas, e ao mesmo tempo ter uma boa organização e rapidez na procura dessa mesma informação decidimos utilizar **Árvores Balanceadas de Procura**.

Resumidamente, cada nodo destas *Árvores* vai conter a informação que pretendemos para a realização das queries, que de maneira prática, por exemplo, vai ser: o titulo do artigo, o seu id, o numero de caracteres que esse artigo possui bem como o numero de palavras, todas as revisões a que foi submetido esse artigo, onde guardamos, o id da revisão, a data (*TimeStamp*) a que foi feita essa revisão, e por fim o *username* de quem a fez e o seu id.

Analisamos tambem outro tipo de estruturas que podiamos utilizar para guardar esta informação principal. Pensamos em **Tabelas de Hash**, **Listas Ligadas**, ou **Heap's**. As duas ultimas colocamos logo de parte para guardar a informação principal, pois tinham pouca eficiência tanto na inserção como na procura, principalmente as *Listas Ligadas*, no entanto poderiam ser úteis como estruturas auxiliares na realização das queries, como vamos explicar mais á frente.

Por fim, ficamos reduzidos a duas possibilidades bastante boas, as **Árvores Balanceadas de Procura** e as **Tabelas de Hash**. Em termos de procura ambas são bastante viáveis, visto que nas *Tabelas de Hash* usamos a função de *Hash* e podemos aceder logo ao campo pretendido, enquanto que nas *AVL's* nos piores casos, temos de fazer uma travessia completa á árvore que demoraria *log N*, sendo que N é o tamanho da árvore, no entanto da maneira que idealizamos para implementar a árvore, poucas ou quase nenhumas travessias completas a árvore tínhamos de fazer, logo este era um problema minoritário. Em relação á inserção de elementos na estrutura, como não sabemos do tamanho que vamos precisar para fazer uma pré seleção de memória, visto que íamos inicializar a *Tabela de Hash* com um valor e caso não chegasse teríamos de voltar a realocar a memória, ou então alocar inicialmente um valor muito alto de memoria, onde poderia sobrar espaços por preencher, o que denotava muito pouca eficiência, portanto, posto isto, achamos que a implementação da *Árvore Balanceadas de Procura* adequava-se melhor á resolução deste problema. No entanto deixamos aqui uma nota que as *Tabelas de Hash* são igualmente uma solução bastante boa, e tanto uma como outra tem aspetos negativos e positivos.

Capítulo 3

Árvores Balanceadas de Procura

Neste capitulo, vamos abordar e explicar a implementação das estruturas principais do nosso projeto. Para organizar a informação criamos duas árvores balanceadas, uma relativa aos artigos e outra relativa aos contribuidores.

3.1 Avl de Artigos

Nesta *AVL* definimos uma estrutura para a cabeça da árvore, que engloba um apontador para o primeiro nodo da árvore, e três variáveis do tipo *long*, sendo que uma é para guardar o numero total de artigos, a outra para guardar o total de artigos únicos, e por fim o total de revisões únicas. Estas variáveis são incrementadas á medida que inserimos ou atualizamos informação na árvore.

Cada nodo da árvore de artigos é constituído por uma variável do tipo *char** que guarda o titulo do artigo, três variáveis do tipo *long* uma que guarda o id do titulo, outra o numero de caracteres do texto do artigo, e outra referente ao numero de palavras do mesmo. Por fim o nodo tem também um apontador para a cabeça de uma *AVL* de Revisões, onde vão ser guardadas todas as revisões referentes ao artigo.

3.1.1 Avl de Revisões

Como foi dito em cima, cada nodo da *AVL* de artigos vai conter uma *AVL* de revisões, logo o numero total de *AVL's* de revisões que vai existir vai ser igual ao numero total de artigos únicos que existir na árvore de artigos. Assim como na *AVl* de artigos, também definimos uma estrutura para a cabeça da árvore de revisões, que é constituída por uma variável do tipo *long* e um apontador para o primeiro nodo da árvore. Cada nodo da árvore de revisões, é composto por duas variáveis, uma do tipo *char** referente á data da revisão e uma variável do tipo *long* que se refere ao id da revisão.

3.2 Avl de Contribuidores

Esta *AVL* diz respeito a todos os contribuidores que realizaram revisões nos artigos existentes nos *backups* da *Wikipédia*. A estrutura da cabeça da árvore de contribuidores é parecida á cabeça da árvore de revisões, ou seja contem uma variável do tipo *long* que corresponde ao numero total de contribuidores que existe na árvore, e um apontador para o primeiro nodo da árvore. E cada nodo é constituído por três variáveis, uma do tipo *char** que se refere ao *Username* do contribuidor, e outras duas do tipo *long* que se referem ao id do utilizador, e ao numero total de contribuições que aquele utilizador executou.

Capítulo 4

Interface

Neste capitulo vamos abordar todos os aspetos da interface do projeto, sendo que teríamos de implementar as estruturas pré-definidas bem como todas as queries.

4.1 Estrutura

4.1.1 Definição

ISTRUCT, assim definida, é composta por dois apontadores, uma para a cabeça da *AVL* de artigos e outro para a cabeça da *AVL* de contribuidores.

4.1.2 init()

Nesta função, chamamos para cada um dos parâmetros da *ISTRUCT*, ou seja o apontador para a cabeça da árvore de artigos, e o apontador para a cabeça da árvore de contribuidores, as funções que inicializam e alocam memoria para os campos das cabeças das árvores. Estas funções estão definidas no ficheiro ".h"relativo a cada *AVL*,

4.1.3 load()

Esta função trabalha essencialmente com o modulo *PARSER*. A função principal do modulo acima descrito é chamada tantas vezes, quantos os números de *backups* que queremos inserir nas nossas estruturas. A função denomina-se **parsedoc()** e recebe como parâmetros as cabeças das árvores onde vai ser feita a inserção dos elementos e o documento de onde vai ser extraída a informação.

4.1.4 clean()

No clean da estrutura principal **ISTRUCT**, chamamos para cada uma das duas árvores que a compõem as funções definidas nos respetivos ".h"que fazem o clean das mesmas. Processa-se de uma maneira fácil, visto que é feito o clean de todos os espaços alocados para variáveis do tipo *char** e depois o clean de cada nodo, e essa função é chamada recursivamente para limpar todos os nodos das árvores.

4.2 Queries

4.2.1 Querie 1, 2 e 3

Nestas três *queries* são chamadas funções auxiliares que estão no modulo *AVL Artigos*, e apenas é feito o retorno dos três parâmetros que estão na cabeça da árvore, que são o total de artigos, o total de artigos únicos, e o total de revisões únicas. Como estes parâmetros já foram incrementados nas inserções não é preciso fazer nenhum tipo de procura.

4.2.2 Querie 4

Esta função está definida na interface e invoca a função **querie4** do ficheiro *avlContribuidores*. Esta querie interage com a AVL de Contribuidores pois incide sobre informação relativa a cada um deles. Esta *querie* utiliza um *array* de uma estrutura auxiliar denominada **CAC** que guarda a informação relativa a um determinado id e o a quantidade de artigos para o qual o dono desse id contribuiu. Assim para responder a esta interrogação é inicializado um array de 10 posições (que serão os 10 maiores contribuidores) e ainda um array de 10 posições para o tipo long que será o resultado da querie. De seguida é chamada uma função auxiliar responsável por obter os 10 maiores contribuidores, querie4_aux. Esta função navega sobre toda a AVL e para cada nodo verifica se ele deve entrar nos 10 maiores contribuidores. Para verificar isso a função apenas compara se o contribuidor do nodo em que se encontra contribuiu para mais artigos do que a posição 0 do array de CAC. Só é necessário verificar a posição 0 pois o array encontra-se sempre ordenado crescentemente pelo numero de artigos. Caso o contribuidor do nodo entre no array com os maiores contribuidores, o array é reordenado. E é feito este processo para todos os nodos da AVL.

4.2.3 Querie 5

A função que esta *querie* chama esta definida no modulo dos contribuidores uma vez que o objetivo é retornar o username de um contribuidor dado um id. É usada uma função auxiliar que se chama **avl_find_contributor** que faz a procura ordenadamente pelo id que pretende encontrar. Uma vez que, as árvores estão ordenadas consoante os id's esta querie foi fácil de implementar.

4.2.4 Querie 6

Esta querie tem como objetivo devolver os id's dos vinte artigos com maior numero de caracteres. De forma a poder fazer esta tarefa, criamos uma estrutura auxiliar denominada **idchar**, que contem dois parâmetros do tipo *long*, em que é para guardar o id do artigo e outro para guardar a quantidade de caracteres presentes nesse artigo. Criamos então, um array com 2 posições desta estrutura. Numa função auxiliar, corre-se a árvore toda, e faz se a comparação com o nodo em causa e os artigos que já fazem parte desta nossa estrutura, caso a quantidade de caracteres seja maior que o mínimo da quantidade dos artigos presentes no array, faz se a substituição e respetiva ordenação utilizando o *quickSort*. Utiliza-se este método para todos os nodos da árvore, recursivamente. Por fim, ao chegar ao fim da pesquisa da árvore, faz-se a copia dos *id's* dos artigos do array para uma estrutura que vai ser devolvida a função implementada na interface.

4.2.5 Querie 7

Esta *querie* é muito semelhante a *querie5*. Do mesmo modo esta implementada uma função noutro modulo, desta vez na *AVL Artigos*. O objetivo desta função é devolver o ultimo titulo de um artigo dado o seu id. É usada uma função auxiliar chamada **avl_find_artigo**, que recebe o id de um artigo e vai procura-lo ordenadamente e recursivamente. Caso o id do artigo pretendido seja maior que o nodo atual a função entra no filho da direita, caso contrario entra no filho da esquerda.

4.2.6 Querie 8

Esta função está definida na interface e invoca a função **querie8** do ficheiro *avlArtigos*. Esta *querie* interage com a AVL de Artigos pois incide sobre informação relativa a cada um deles. Aqui é utilizado um array de uma estrutura auxiliar denominada **AAI** que guarda a informação relativa a um determinado id e o comprimento do artigo relativo a esse id. Assim para responder a esta interrogação é inicializado um array de N posições (que serão os N maiores artigos) e ainda um array de N posições para o tipo *long* que será o resultado da querie. O *N* é o parâmetro passado na invocação desta interrogação. De seguida é chamada uma função auxiliar responsável por obter os N maiores artigos, **querie8_aux**. Esta função navega sobre toda a AVL e para cada nodo verifica se ele deve entrar nos 10 maiores artigos. Para verificar isso a função apenas compara se o artigo do nodo em que se encontra é mais longo que o artigo que esta na posição 0 do array de AAI. Só é necessário verificar a posição 0 pois o array encontra-se sempre ordenado crescentemente pelo tamanho do artigo. Caso o artigo do nodo entre no array com os maiores artigos, o array é reordenado. E é feito este processo para todos os nodos da AVL.

4.2.7 Querie 9

Esta função está definida na interface e invoca a função **querie9** do ficheiro avlArtigos. Esta querie interage com a AVL de Artigos pois incide sobre informação relativa a cada um deles. Aqui é utilizado um array de uma estrutura auxiliar, uma lista ligada, denominada **linkedList_t** que guarda a informação relativa a um determinado titulo. Assim para responder a esta interrogação é inicializada uma lista ligada do tipo descrito acima que ficará com os títulos que cumpram um determinado prefixo. O prefixo será passado por argumento na invocação desta interrogação. De seguida é chamada uma função auxiliar responsável por obter os títulos que possuem o prefixo indicado, **querie9aux**. Esta função navega sobre toda a AVL e para cada nodo verifica se o titulo do artigo representado no nodo começa com determinado prefixo. A função responsável por verificar se um titulo começa com um prefixo é a **startsWith**. Caso a função determine que um titulo começa com o prefixo esse titulo é inserido na lista ligada e a variável que contem o numero de títulos na lista ligada é incrementada. No fim da travessia sobre a AVL o resultado (ou seja, a lista ligada preenchida com os títulos) é devolvida para a função **querie9**. Assim, é alocado um array com tantas posições quantas o numero de títulos existentes na lista ligada acrescido de uma unidade. É feita uma travessia sobre a lista ligada e cada titulo presente na lista ligada é copiado para o array. Por fim, é colocado na ultima posição do array o valor *NULL* para indicar o fim dos elementos.

4.2.8 Querie 10

Esta querie tem duas funções auxiliares importantes, sendo que estas duas, uma esta contida na outra. A primeira situa-se no modulo da *AVL* de artigos e, com a ajuda da função **avl_find_artigo** já descrita na *querie7* vai procurar o nodo de um artigo dado o seu id. Assim que esta tarefa ta completa, o programa mergulha noutra função auxiliar que esta definida no modulo da avl de revisões. Esta função recebe a cabeça da avl que o nodo acima descrito continha e um id da revisão. Com a ajuda de uma função de procura semelhante á do modulo dos artigos denominada **avl_find_revisão**, vai procurar a revisão pretendida através do seu id. Por fim, a função retorna a data a que foi feita essa revisão.

Capítulo 5

Encapsulamento

Neste capitulo vamos abordar de que forma foi feito o encapsulamento dos módulos no nosso programa, visto que é dos aspetos mais importantes deste projeto.

De forma a garantir o encapsulamento de todo o programa criamos vários módulos onde definimos os cabeçalhos das funções num ficheiro ".h"e a sua implementação num ficheiro ".c"de forma a criarmos as nossas próprias bibliotecas.

Deste modo, de forma a importar as bibliotecas de forma correta, apenas importamos em cada modulo, o modulo que vamos necessitar de forma a não fazer um *loop* na compilação dos vários módulos.

Em suma, os únicos módulos que não dependem dos outros é ficheiro da *AVL de Revisões*, pois apenas é feito o *include* do seu próprio ".h", e o ficheiro da *AVL de contribuidores*. No ficheiro da *AVL de artigos*, é feito o include da *AVL de Revisões*, pelo que já foi explicado no capitulo 3.1.

O ficheiro do *Parser* vai ter incluído as bibliotecas da *AVL de Artigos* e da *AVL de contribuidores*. Por fim o ficheiro do *program*, que é onde esta a *main* só vai incluir a *Interface* que por sua vez tem incluído as *AVL's* de artigos e contribuidores e o modulo do *parser*.

Todas estas dependências são devido a chamadas de funções, ou não reconhecimento de estruturas, que pertencem a outros módulos que não os que tão a ser chamados no momento.

Capítulo 6

Conclusão

A elaboração da primeira fase do trabalho prático desta unidade curricular foi um bom modo de aprofundarmos os conhecimentos obtidos acerca da linguagem de programação em *C* ao nível da modularidade,ou seja, da estruturação do código, ajudo-nos a perceber que este conceito é muito importante ao desenvolver-mos um projeto, pois permite a sua fácil evolução,manutenção e legibilidade.

As principais dificuldades encontradas foram sobretudo na forma como estruturar os módulos criados, de modo a interligarmos todos os nossos ficheiros. Assim que ultrapassado este obstáculo, a resolução do problema proposto tornou-se mais fácil.

O trabalho embora que completo e funcional, a nível de estruturação de código, apercebemos-nos também que as nossas funções poderiam e deviam ter tido nomes mais sugestivos e concordantes entre si,e que poderia-mos ter criado alguns módulos adicionais de modo a facilitar a leitura do código desenvolvido.

Não obstante, o trabalho cumpre todos os requisitos pedidos de forma eficaz, concluindo então que, embora pudesse estar mais completo e estruturado trata-se dum trabalho competente.