Projeto de Laboratórios de Informática III
Grupo 18

André Vieira a78322 Carlos Pedrosa a77320
David Sousa a78938 Manuel Sousa a78869

April 29, 2017

Abstract

Este projeto tem como principal objetivo a criação de um sistema que permita analisar os artigos presentes em backups da Wikipedia, realizados em diferentes meses, e extrair informação útil para esse período de tempo como, por exemplo, o número de revisões, o número de novos artigos, etc.

Assim, este relatório pretende sumarizar todos os esforços efetuados para alcancar o objetivo proposto. Nesse sentido, serão apresentadas as soluções feitas nas diversas tarefas propostas pelos docentes.

Contents

1 Introdução **2**

2 Descrição do Problema **2**

3 Concepção da Solução **2**
3.1 Estruturas de Dados 2
3.2 Carregamento dos dados para a Estrutura de Dados 3
3.3 Desenvolvimento das Interrogações 4
3.4 Dificuldades na concepção da solução 5

4 Conclusões **5**

1 Introdução

Este trabalho foi-nos proposto no âmbito da unidade curricular de Laboratórios de Informática III e tem como principal objetivo fazer com que haja, por parte dos alunos, um maior contacto com a linguagem de programação imperativa C bem como a sensibilização destes para a escolha de código rápido e eficiente quando se trata do manuseamento de grandes quantidades de dados.

Nesse sentido, foi-nos apresentado um projeto dividido em duas tarefas fundamentais, no qual o fim único da sua realização é a consulta rápida dos dados, usando para isso certas querys que foram fornecidas pelos professores. Assim, neste relatório, vamos descrever todos os passos dados para alcancar o objetivo proposto, bem como os problemas que foram surgindo aquando a realização desses mesmos passos.

Em suma, pretendemos falar do desenvolvimento e da concepção do código, explicando, deste modo, a forma de como efetuamos cada uma das tarefas sugeridas pelos docentes.

2 Descrição do Problema

O objetivo geral do projeto apresentado é o desenvolvimento de um sistema de pesquisa rápido. Para esse efeito, os problemas foram abordados conforme o enunciado apresentado.

Num primeiro momento, deparamo-nos com a biblioteca xml, pelo que o primeiro problema que tivemos foi o manuseamento desta.

Por último, os snapshots disponibilizados pelos professores são de tamanho relativamente elevado. Nesse sentido, é necessária uma estrutura rápida e concisa que permita um acesso aos dados quase imediato. Sendo assim, foi outra das dificuldades que tivemos de ultrapassar.

3 Concepção da Solução

Ao longo do trabalho efetuado fomos testando as nossas soluções num site que nos foi disponibilizado. Assim, desenvolvemos o código necessário para dar resposta a todas as interrogações que foram propostas.

3.1 Estruturas de Dados

As *estruturas de dados* são um modo de armazenamento e organização de dados, de modo, que podem ser usadas de forma eficiente, facilitando assim a busca e modificação.

Nesse sentido, como era necessária uma estrutura de dados relativamente eficiente optamos por utilizar uma biblioteca do sistema, a *GLib*. Esta pareceu-nos uma biblioteca bastante adequada pois contém uma série de estruturas já implementadas. Assim, dentro de uma grande panóplia de estruturas oferecidas por esta biblioteca usamos a *GTree*(Balenced Binary Tree), uma vez que, considerarmos ser a que, em geral, mais se adequava ao trabalho proposto. Esta estrutura é em tudo semelhante a uma AVL, acrescentando o pormenor de que em cada nodo da árvore existe uma chave e um valor, semelhante também a uma tabela de hash.

As nossas estruturas principais consistem, em duas Balenced Binary Trees, cujo valor é um apontador para outro tipo de estrutura. Ambas as estruturas baseiam-se numa lista ligada com diferentes variáveis, as quais servem para guardar diferente tipo de informação sobre cada usuário, artigo e respetivas revisões. Desta forma podemos aceder a cada artigo (ou revisão do mesmo) e retirar de lá a informação pretendida para respondermos de forma rápida e concisa a certas interrogações. A primeira estrutura trata da informação dos artigos e suas revisões, a segunda contém o número de vezes que um certo usuário contribui para diferentes artigos. Este usuário é identificado pelo seu ID (key da árvore) e pelo seu Username.

```
typedef struct cont {
    char* userName;
    int rep;
} *User;

typedef struct node {
    char* title;
    int idRevisao;
    char* timestamp;
    long tamanho;
    long words;
    struct node *next;
} *ValueNode;
```

3.2 Carregamento dos dados para a Estrutura de Dados

O carregamento dos dados foi feito através do parser de ficheiros da biblioteca xml, o qual nos facilitou a extração da informação que pretendíamos para dar resposta às diferentes interrogações. Apesar do uso da Balanced Binary Tree, é aqui que o programa tem o maior consumo de tempo. O load é feito através da pesquisa sobre os diferentes campos do ficheiro xml, até chegarmos ao campo pretendido, onde extraímos os dados pretendidos através de funções fornecidas pela biblioteca da libxml. Posto isto, é chamada uma função a qual cria um nodo para a árvore com todos os dados necessários, para depois ser adicionado à sua respetiva estrutura. Após diversos testes em todos as máquinas dos alunos responsáveis por este trabalho, vimos que o tempo de execução do load e o carregamento dos dados, rondava os 15 segundos. Também nos foi fornecida uma plataforma de testes sobre as nossas interrogações, onde aqui o tempo de load foi menor pois as máquinas usadas pelos docentes terão melhores arquiteturas.

```
gcsousa@gcsousa-port:~/LI3$ time ./program

real    0m14.655s
user    0m12.744s
sys     0m1.892s
```

```
real    0m10.823s
user    0m9.628s
sys     0m1.176s


Diff between group results and Server results: If no output appears then there are no differences
diff ../grupo18/results.txt ../Server/results.txt
```

3.3 Desenvolvimento das Interrogações

O desenvolvimento das interrogações foi feita após todas as implementações das estruturas de dados, bem como o carregamento da informação para estas. Posto isto, algumas das interrogações foram automaticamente respondidas pois os dados necessários estavam na estrutura e foram calculados na função de load. Como a nossa estrutura se baseava numa árvore, efetuamos diferentes travessias sobre a estrutura de dados com o intuito de calcular e obter certa informação para ser retornada. Aqui, usamos diversas funções de travessia para serem aplicadas a cada nodo da árvore, pois era um requisito da função de procura da biblioteca GLib, a qual retorna sempre um valor booleano que permitirá continuar a travessia. A título de exemplo, mostraremos a função que calcula o número total de artigos, bem como as revisões dos snapshots analisados. É enviado um contador a todos os nodos, a qual é somado o total de cada artigo e revisões, estando estas guardadas numa lista ligada.

```
int tamanho(ValueNode revisoes) {
    int res = 0;
    ValueNode aux = revisoes;
    while(aux!=NULL){
        res++;
        aux = aux->next;
    }
    return res;
}

gboolean iter_all(gpointer key, gpointer value, gpointer data) {
    *(int*)data += tamanho((ValueNode)(value));
    return FALSE;
}

long all_revisions(TAD_istruct qs) {
    long n_revisoes = 0;
    g_tree_foreach(qs->avlIDs, (GTraverseFunc)iter_all, &n_revisoes);
    return n_revisoes;
}
```

Os testes e tempos sobre todas as interrogações foi nos dada pela plataforma cedida pelos docentes. Seguem-se os resultados obtidos:

```
init() -> 0.024000 ms
load() -> 10629.255000 ms
all_articles() -> 0.000000 ms
unique_articles() -> 0.001000 ms
all_revisions() -> 2.158000 ms
top_10_contributors() -> 1.125000 ms
contributor_name(28903366) -> 0.000000 ms
contributor_name(194203) -> 0.000000 ms
contributor_name(1000) -> 0.000000 ms
top_20_largest_articles() -> 6.361000 ms
article_title(15910) -> 0.002000 ms
top_N_articles_with_more_words(30) -> 6.464000 ms
article_title(25507) -> 0.001000 ms
article_title(1111) -> 0.003000 ms
titles_with_prefix(Anax) -> 2.507000 ms
article_timestamp(12,763082287) -> 0.002000 ms
article_timestamp(12,755779730) -> 0.000000 ms
article_timestamp(12,4479730) -> 0.000000 ms
clean() -> 1.267000 ms
```

3.4 Dificuldades na concepção da solução

No decorrer do trabalho deparamo-nos com diversas dificuldades, sendo que, na sua grande maioria foram ultrapassadas. No entanto, apesar de tudo, tivemos alguns problemas que apenas foram parcialmente resolvidos. Assim, mesmo usando a estrutura de dados que nos pareceu ser a mais adequada, continuamos a obter tempos aquém do que seria esperado. Tal problema, pode dever-se não só à estrutura que usamos mas também ao modo como as funções foram implementadas. No caso da função load, sendo esta a que tem um maior impacto em todo o programa, usamos listas ligadas que podem influenciar negativamente o tempo de execução do ficheiro quando a sua utilização se torna demasiada elevada. De modo a resolver estes problemas pensamos no uso de uma tabela de Hash, também ela implementada na *GLib*, pois em termos de tempo de execução, certas queyrs pediam o uso desta. Contudo, a escolha desta estrutura era um pouco arriscada pois tal implementação requeria uma função de Hash bastante precisa para evitar colisões, e optámos por não implementar devido à relação tempo-trabalho.

4 Conclusões

De acordo com o objetivo do projeto, sendo este o acesso rápido a um conjunto elevado de dados, podemos concluir que este foi razoavelmente concluído.

De facto, apesar das nossas tentativas, não nos foi possível obter um tempo de execução menor. Consideramos, assim, que o grande objetivo foi alcançado.

Efetivamente, e apesar de tudo, o projeto no que diz respeito à primeira fase foi satisfatoriamente resolvido pelo que o código necessário para o funcionamento das queries foi desenvolvido.

Em jeito de conclusão, o presente trabalho serviu para um aprimoramento das nossas aptidões relativamente à linguagem de programação imperativa C. Nesse sentido, este trabalho formatou a nossa maneira de pensar aquando da resolução de um problema, tornando-nos mais objetivos e racionais.