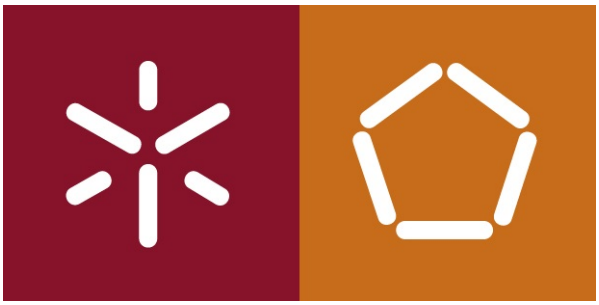
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Laboratórios de Informática III

Trabalho realizado por:	*Número*
Diogo Soares	A74478
João Cabo	A75064
Pedro Almeida	A74301

Conteúdo

1 Introdução

O projeto apresentado pelos docentes da Unidade Curricular de Laboratórios de Informática III resume-se á criação de um programa desenvolvido em C, nesta primeira instância, e cujo objectivo é o armenazamento de informações pertinentes provenientes de snapshots da Wikipedia fornecidos. Os dados armazenados têm que ser capazes de responder a 10 queries previamente disponibilizadas. Para além da resolução que permita uma correta resposta aos problemas estabelecidos pretende-se a construção de um trabalho modular, ou seja, a divisão do código de modo a ficar dividido em unidades separadas,e tambem que seja código bem estruturado de modo a permitir a sua boa manutenção e legibilidade.

2 Estrutura do Relatório

Começamos este relatório por explicar as nossas estruturas principais no projecto, explicando o seu objetivo e o seu conteúdo.

De seguida explicamos a nossa resolução das interrogações a que o programa deve responder.

Apresentamos ainda a nossa makefile, acabando por fim, com a conclusão do relatório.

3 Estrutura de dados

De modo a armazenar toda a informação necessária criamos uma estrutura Registo que contem 2 arrays de apontadores para Avl's sendo que cada array contem 10 apontadores. Um dos arrays trata dos dados relativos aos artigos e o outro array dos dados relativos aos contribuidores. Decidimos usar este tipo de armazenamento pois achamos que era eficiente e simples de implementar. Cada nodo da Avl terá um id, sendo que é consuante o id que se decide em que Avl do array se guarda a informação.Por exemplo um artigo de id 284618 terá a sua informção guardada na Avl da posição 2 do array dos Artigos.Para além do id, o nodo contem também um void* info que será uma estrutura com informações adicionais que varia sendo um artigo ou contribuidor, um interio height que traduz a altura da arvore e um apontador para o nodo esquerdo subsequente e outro para o nodo da direita seguinte.

Estrutura do Registo:

```
struct reg{
    Avl artigos[SIZE];
    Avl contribuidores[SIZE];
};
```

Estrutura do Nodo:

```
struct nodeAvl{
    long id;
    void *info;
    int height;
    struct nodeAvl *left;
    struct nodeAvl *right;
};
```

3.1 Parser

O parser vai ser o instrumento que a partir dos snapshots fornecidos vai captar a informação que queremos e guarda-la na estrutura indicada. O parser lê linguagem Xml e ao guardar será guardada informçação com tipo presente na linguagem C havendo aí uma "moldagem"da informação ao ser lida e armazenada. Esta é uma das partes fundamentais do projecto pois se o parser estiver

mal construido poderá existir uma perda significativa de informção que poderia ser necessária para os objetivos do trabalho. O tempo de execução po parser também é uma parte importante na otimização do código pois como atravessa grande quantidade de dados irá ser dos momentos do trabalho com maior tempo de execução.

3.2 Artigos

Como já foi referido anteriormente cada artigo terá a sua informação guardada numa Avl respetiva ao primeiro digito do seu ID.

Estrutura do Artigo:

```
struct artigo {
        char* titulo;
        int n;
        char** timestamp;
        long* revId;
        long bytes;
        long words;
};
```

- titulo: String onde guardamos o titulo do artigo e que é atualizado a cada revisão.
- n: numero de vezes que o artigo aparece nos backups.
- timestamp: array de char* que vai guardar os timestamps das revisões do artigo.
- revId: array que vai guardar os ID's das revisões nesse artigo.
- bytes: número de bytes do texto do artigo. Guarda o maior valor encontrado nas suas respetivas revisões.
- words: número de palavras do texto do artigo. Guarda o maior valor encontrado nas suas respetivas revisões.

Estas serão as únicas informações adicionais de cada artigo, para alem do id, que serão retiradas e armazenadas visto que estas são as informações necessárias para dar resposta ás questões referentes ao artigo. Optamos por não guardar mais dados relativos ao artigo pois seriam um gasto desnecessário de memória, aumentando assim a otimização do programa.

3.3 Contribuidor

Tal como nos artigos, os contribuidores estão organizados por avl's referentes ao seu ID. Cada nodo destas avl's contêm o id de um contribuidor assim como um apontador para uma estrutura Contribuidor que tem os dados desse mesmo utilizador.

Estrutura do Contribuidor:

```
struct contribuidor{
    char* username;
    int cont;
};
```

- username: string que contém o username do contribuidor.
- cont: número total de contribuições do contribuidor.

Estas são as unicas informações necessárias a reter para responder ás questões sobre contribuidores todas as informações restantes não são guardadas pois seriam apenas uma ocupação de memoria desnesessário .

4 Interrogações

Iremos nesta secção explicitar a nossa resolução das interrogações a que o sistema terá de responder.

1. Retorna todos os artigos encontrados nos backups analisados.

   Para esta função recorremos ao foreach (função que percorre uma árvore, e para cada nodo da mesma, aplica uma função guardando o resultado numa variável que lhe é passada), somando em cada nodo o número de vezes que cada artigo aparece nos backups. Temos assim no final, a soma de todos os artigos.

2. Pretende saber quais os artigos únicos encontrados nos vários backups analisados.

   Para responder a esta questão retornamos a soma de todos os nodos de cada avl do array dos artigos.

3. Pretende saber quantas revisões fora efetuadas naqueles backups.

   Percorremos todas as avl's do array artigos e em cada artigo, comparamos os seus ID's de revisão guardando e somando o total do número de revisões diferentes.

4. Devolve um array com os identificadores dos 10 autores que contribuíram para um maior número de versões de artigos.

   Percorremos as avl's correspondentes aos contribuidores e guardamos num array os ID's dos utilizadores que mais contribuiram assim como o número total das suas contribuições. Para cada nodo verificamos se o número de contribuiçoes é superior aos que estão no array atualizando-o. Devolvemos por fim, o array só com os ID's dos contribuidores com mais contribuições.

5. Devolve o nome do autor com um determinado identificador.

   Procuramos na avl correspondente consoante o ID e quando encontramos o nodo com o identificador pretendido devolvemos o username guardado na sua estrutura contribuidor. Caso esse ID não seja válido ou não exista, a função devolve NULL.

6. Devolve um array com os identificadores dos 20 artigos que possuem textos com um maior tamanho em bytes.

   Percorremos todas as árvores de artigos e guardamos num array os ID's dos artigos com maior tamanho em bytes assim como o seu valor. Para cada artigo comparamos o seu tamanho com os presentes no array, e inserimos no array aqueles que tiverem valores superiores. Devolvemos por fim um array com os ID's dos top 20 artigos de maior tamanho.

7. Devolve o título do artigo com um determinado identificador

   Procuramos na avl correspondente consoante o ID e quando encontramos o nodo com o identificador pretendido devolvemos o titulo guardado na sua estrutura artigo. Caso esse ID não seja válido ou não exista, a função devolve NULL.

8. Devolve um array com os identificadores dos N artigos que possuem textos com um maior número de palavras.

   Percorremos as avl's correspondentes aos artigos e guardamos num array os ID's dos artigo que contêm mais palavras assim como o seu valor.. Para cada nodo verificamos se o número de palavras é superior aos que estão no array atualizando-o. Devolvemos por fim, o array com os ID's do top N de artigos com mais palavras.

9. Devolve um array de títulos de artigos que começam com um prefixo passado como argumento da interrogação.

   Guardamos num array dinâmico todos os titulos com o prefixo dado. Para isto percorremos todas as avl's do array e comparamos cada titulo com o prefixo (guardado na primeira posição do array o prefixo necessário para a comparação). Devolvemos no fim, um array com os titulos ordenados com esse prefixo.

10. Devolve o timestamp para uma certa revisão de um artigo.

    Procuramos na arvore correspondente pelo ID do artigo. Uma vez encontrado, comparamos a revisao dada com as revisões do artigo, devolvendo o timestamp correspondente. Caso o ID do artigo ou da revisão nao exista, a função devolve NULL.

5 Makefile

```
CC = gcc
CFLAGS = -Wall -g `pkg-config --cflags libxml-2.0` `pkg-config --cflags glib-2
LIBS = `pkg-config --libs libxml-2.0` `pkg-config --libs glib-2.0`

program:
	$(CC) $(CFLAGS) *.c -o program $(LIBS)

clean:
```

```
rm program
```

6 Conclusão

O desenvolvimento deste projecto permitiu um grande desenvolvimento de práticas que serão de extrema necessidade para o futuro tal como o contacto com um grande volume de dados e até a maneira como se organiza e se desenvolve o código. O projecto permitiu uma maior proximidade com estruturas de dados, algumas das quais relativamente novas para elementos do grupo o que fez com que o nivel de conhecimento da linguagem e das suas capaciddades sofresse um aumento.