Universidade do Minho
Escola de Engenharia

Wikipédia

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III
2º Semestre
2016-2017

A70565 Bruno Manuel Borlido Arieira
A70938 Diogo Meira Neves
A70824 Luís Miguel Bravo Ferraz

1 de Maio de 2017
Braga

Conteúdo

Capítulo 1

Introdução

No âmbito da Unidade Curricular de Laboratórios de Informática III, foi proposto a realização de um projeto prático, utilizando a linguagem de programação C, que tivesse em conta os vários aspetos do desenvolvimento dum programa de software de média/larga escala.

Este projeto consiste na construção de um sistema que permita analisar os artigos presentes em backups da Wikipédia, realizados em diferentes meses, e na extração de informação útil destes. Cada um destes backups está num ficheiro.xml, disponibilizado pela equipa docente, e para cada um deles o programa deve fazer o parse e guardar os dados em memória. Deste modo, para guardar os dados de modo a garantir o encapsulamento dos mesmos, usaram-se três módulos: parse, interface e estrutura, e ainda recorremos ás biblioteca AVL da GNU. Depois dos ficheiros serem percorridos e os dados serem carregados para as respetivas estruturas, desenvolveram se as Queries propostas no enunciado do projeto prático. Para responder às diferentes Queries foram utilizadas as funções definidas na API dos diferentes módulos já referidos.

Neste relatório apresentamos as decisões que foram tomadas pelo grupo na implementação do projeto, nomeadamente as estruturas utilizadas para criar cada um dos módulos e as suas API's.

Capítulo 2

Módulos de Dados

Nesta secção apresenta-se o desenho da estrutura de dados todos os typedef e a API comentada função a função.

2.1 Módulo Estrutura

Para armazenar os dados extraídos dos ficheiros, decidimos criar uma Árvore Balanceada "grupo_titulos", que possui um long "artigos", que corresponde ao número de artigos analisados, e uma Árvore Balanceada(AVL) Títulos. Esta AVL "titulos", em cada nodo, possui um long "id_titulo(fator de comparação para o balanceamento da árvore) , que corresponde ao identificador de cada artigo, e uma Árvore Balanceada de Revisões. Esta AVL "revisoes" possui, em cada nodo, um char* "titulo" que contém o Titulo do artigo, um long id_revisao(fator de comparação para o balanceamento da árvore) que é o que identifica cada revisão, um char* "texto" que contém o texto do artigo, um char* "timestamp" que contém informações relativamente a data, um long "id_contribuidor"que identifica a pessoa que fez a revisão, um char* "nome_contribuidor" que é o username de quem fez a revisão e por último um long "num_palavras" que corresponde ao número de palavras que o texto do artigo tem. Concluindo, para cada artigo é criado um nodo com o id do seu titulo e para cada nodo é criada uma AVL com as informações sobre a revisões.

2.1.1 Typedef

- ```
  typedef struct grupo_titulo{
      AVL titulos;
      long artigos;
  }grupo;
  ```
- ```
  typedef struct titulos{
      long id_titulo;
      AVL revisoes;
  }tit;
  ```
- ```
  typedef struct revisoes{
      char* titulo;
      long id_revisao;
      char* texto;
      char* timestamp;
      long id_contribuidor;
      char* nome_contribuidor;
      long num_palavras;
  }rev;
  ```
  ```

- `typedef struct grupo_titulo* GT;`
- `typedef struct titulos* Titulo;`
- `typedef struct revisoes* Revisao;`

2.1.2 Funções da API

- `int compara_Revisao(const void *revisao_a, const void *revisao_b, void *param);`

  Função que compara os "id_revisao", de modo a que na inserção de um nodo na árvore, esta fique balanceada.

- `int compara_Titulo(const void *titulo_a, const void *titulo_b, void *param);`

  Função que compara os "id_titulo", de modo a que na inserção de um nodo na árvore, esta fique balanceada.

- `GT in_grupo_titulo();`

  Função que inicia uma AVL do tipo GT.

- `Titulo in_titulo();`

  Função que inicia uma AVL do tipo Titulo.

- `Revisao in_Revisao;`

  Função que inicia uma AVL do tipo Revisao.

2.1.3 Desenho da estrutura

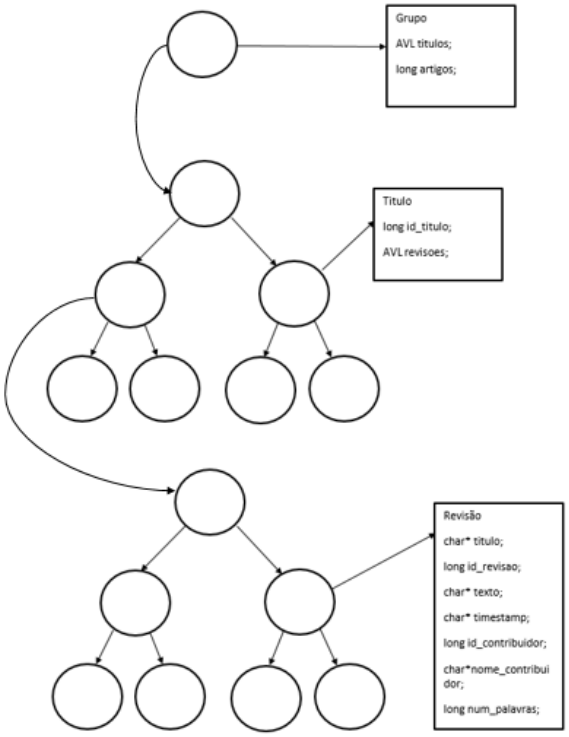

Figura 2.1: Representação da estrutura e respetivo conteúdo.

2.2 Módulo Parser

Depois de definido o esqueleto do projeto, ou seja, a estrutura, foi necessário obter e guardar os dados nesta. Para isso, tivemos de correr os ficheiros e analisar o conteúdo de cada campo do xml, que ao longo do tempo fomos descobrindo que possui algumas caraterísticas especiais, nomeadamente na tag “text”.

2.2.1 Typedef

Não implementamos nenhum.

2.2.2 Funções da API

- `char* parseTitle (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”title”do xml presente nos snapshots.

- `char* parsePageId (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”id”dentro da tag ”page”do xml presente nos snapshots.

- `char* parseTimeStamp (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”timestamp”, que se encontra dentro da tag ”revision”, do xml presente nos snapshots.

- `char* parseRevisionId (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”id”, que se encontra dentro da tag ”revision”, do xml presente nos snapshots.

- `long contaPal(char* s);`

  Função auxiliar que conta o número de palavras de um determinado texto passado como parametro.

- `char* parseText (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”text”, que se encontra dentro da tag ”revision”, do xml presente nos snapshots. Esta foi uma das funções que tivemos de ter um cuidado especial, porque o ”text”vai buscar todo o texto dos campos, e os que não tem devolve NULL. Para isso tivemos de verificar se a string presente no nodo era NULL, e caso não fosse entao podiamos guarda-la.

- `char* parseUsername (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”username”, que se encontra dentro da tag ”contributor”, do xml presente nos snapshots.

- `char* parseUsernameId (xmlDocPtr doc, xmlNodePtr cur);`

  Função auxiliar da função ”parseDoc”que retira o conteúdo do campo ”id”, que se encontra dentro da tag ”contributor”do xml presente nos snapshots.

- `GT parseDoc(char *docname, GT res);`

  Esta a a função principal do módulo ”Parser”, esta é a função que utiliza todas as anteriores para retirar o conteúdo dos campos do xml, e insere na estrutura(depende do módulo ”estrutura”). Na realização desta função tivemos de ter alguns cuidados, como por exemplo, a criação de variáveis auxiliares(de modo a garantir o encapsulamento dos dados,tivemos de fazer algumas comparações, nomeadamente se o ”id_titulo”já existia,para não criarmos dois nodos com o mesmo id, verificar se os artigos tinham texto ou não, também verificamos que certos artigos não continham os campos

"username"e respetivo "id", e por isso também tivemos de fazer essa verificação. Depois tivemos de fazer as inserções na estrutura, e por fim libertar a memória que criamos com as variáveis auxiliares, fazendo free's.

2.3 Módulo Interface

Depois de os dados estarem guardados, foi nos proposta a realização de algumas Queries, este módulo contém a realização dessas Queries. Também é neste módulo que nos deparamos com o tipo abstrato de dados, que foi é um conceito novo para nós e que verificamos que traz vantagens na realização do projeto.

2.3.1 Typedef

- ```
  typedef struct TCD_istruct{
      GT grupo;
  }TAD;
  ```
- `typedef struct TCD_istruct* TAD_istruct;`

2.3.2 Funções da API

- `TAD_istruct init();`

  Função que inicia a estrutura, ou seja, esta está vazia, apenas foi alocado o espaço para receber os dados.

- `TAD_istruct load(TAD_istruct qs, int nsnaps, char* snaps_paths[]);`

  Função que carrega os dados de vários snapshots para a estrutura. Esta função utiliza o "parseDoc", ou seja, a função "parseDoc"é chamada "nsnaps"vezes. Os snapshots são passados através do array "snaps_paths[]".

- `long all_articles(TAD_istruct qs);`

  Função que retorna todos os artigos que foram analisados. Apenas é necessário aceder à variável "artigos"de GT, visto que esta já conta o número de artigos na inserção dos dados na estrutura.

- `long unique_articles(TAD_istruct qs);`

  Função que retorna todos os artigos unicos, para isso é apenas necessário contar os nodos da AVL "titulos", usando o avl_contador(AVL a), da biblioteca AVL da GNU.

- `long all_revisions(TAD_istruct qs);`

  Função que nos devolve o número de revisões totais dos dados analisados. Recorremos ao iterador da biblioteca AVL da GNU, iteramos a AVL "titulos"e para cada nodo fazemos a contagem de revisões, através do função avl_contador(AVL a) que conta os nodos de cada AVL "revisoes"(neste caso), guardando o resultado numa variavel auxiliar do tipo long.

- `long* top_10_contributors(TAD_istruct qs);`

  Começamos por iterar a AVL "titulos", depois para cada nodo iteramos a respetiva AVL "revisoes"guardamos o "id_contribuidor"num array auxiliar depois inserimos na struct auxiliar criada, e guardamos noutro array auxiliar o numero de ocorrencias. Não conseguimos associar o array das ocorrencias ao respetivo array contribuidores.

- `char* contributor_name(long contributor_id, TAD_istruct qs);`

  Esta função devolve o nome de um autor dado um determinado identificador, ou seja iteramos, recorrendo ao iterador da biblioteca AVL da GNU, a AVL "titulos", depois para cada nodo vamos ter de iterar a respetiva AVL "revisoes"e verificar se existe algum nodo em que o "id_contribuidor"seja
  ```

igual ao "contribuidor_id"caso exista, acedemos ao nome do contribuidor presente nesse nodo.Caso não exista, retorna NULL.

- `long* top_20_largest_articles(TAD_istruct qs);`

  Começamos por iterar a AVL "titulos", depois para cada nodo iteramos a respetiva AVL "revisoes"guardamos num array auxliar o comprimento do texto do artigo. Em seguida o objetivo era através desse comprimento achar o id do artigo mas não conseguimos associar o id do artigo ao comprimento do texto.

- `char* article_title(long article_id, TAD_istruct qs);`

  Esta função devolve o titulo do artigo dado um determinado identificador, ou seja iteramos, recorrendo ao iterador da biblioteca AVL da GNU, a AVL "titulos", e verificamos se existe algum nodo em que o "id_titulo"seja igual ao "article_id"caso exista, iteramos a AVL "revisoes"até ao último nodo e devolvemos o titulo presente nesse nodo.Caso não exista, retorna NULL.

- `long* top_N_articles_with_more_words(int n, TAD_istruct qs);`

  Começamos por iterar a AVL "titulos", depois para cada nodo iteramos a respetiva AVL "revisoes"guardamos num array auxliar o numero de palavras do texto do artigo. De seguida fizemos a ordenação do array auxiliar, depois tentamos ir a AVL revisoes e procurar pelo id correspondente a esse numero de palavras.

- `int Pref (char* s1, char* s2);`

  Função auxiliar da char** titles_with_prefix(char*prefix, TAD_istruct qs) que nos indica se uma palavra é ou não prefixo de outra.

- `char** titles_with_prefix(char*prefix, TAD_istruct qs);`

  Esta função devolve nos um array de apontadores, cujo o conteúdo saõ os titulos com o mesmo prefixo. Primeiro iteramos, recorrendo ao iterador da biblioteca AVL da GNU, a AVL "titulos", depois a AVL "revisoes". De seguida, verificamos se o titulo de cada nodo contém o prefixo passado como parametro, através da função auxiliar "Pref". Guarda-mos os que forem num array auxiliar e depois "eliminamos"os repetidos transformando os em strings compostas apenas por "xxx". EXEMPLO: A string "João,João, Fernando, João,Fernado"passa a "João,xxx,xxx,Fernado,xxx,xxx". Por fim retiramos tudo que for diferente de "xxx"para um novo array,e por fim ordenamos por ordem alfabética.

- `char* article_timestamp(long article_id, long revision_id, TAD_istruct qs);`

  Esta função devolve o "timestamp"do artigo dado o identificador do artigo e da revisão, ou seja iteramos, recorrendo ao iterador da biblioteca AVL da GNU, a AVL "titulos", e verificamos se existe algum nodo em que o "id_titulo"seja igual ao "article_id", caso exista, iteramos a AVL "revisoes"e verificamos se existe algum nodo em que o "id_revisao"seja igual ao "revision_id", caso exista devolvemos o "timestamp"presente nesse nodo.Caso não exista, retorna NULL.

- `TAD_istruct clean(TAD_istruct qs);`

  Função que liberta a memória, que foi alocada para guardar a informação toda dos snapshots. Para isso, mais uma vez,iteramos, recorrendo ao iterador da biblioteca AVL da GNU, a AVL "titulos", e para cada nodo, iteramos a AVL "revisoes"e para cada nodo libertamos a memoria ocupada por este, depois dessa AVL "revisoes"estar vazia, libertamos a memoria ocupada pelo repetivo nodo da AVL "titulo"e no final libertamos a memoria ocupada pela AVL "grupo_titulo".

2.4 Biblioteca AVL(GNU)

As bibliotecas da GNU de AVL foram uma grande ajuda para o nosso trabalho, visto que tinham métodos que acabaram por ser bastante úteis para todos os módulos devido a métodos, como por exemplo,

quando necessitamos de fazer a tavessia de sobre as AVL usando a função avl_T_alooc e posteriormente para iterar, while(avl_t_next(b)!=NULL){...}. Outras funções também bastante importantes foram as de criação, avl_create(), de inserção avl_insert() e de procura avl_find(). No entanto, decidimos acrescentar a função avl_contador que conta os nodos de uma AVL e dois typedefs(apresentados em baixo) de modo a simplificar o código.

2.4.1 Typedef (acrescentados)

- `typedef struct avl_table * AVL;`
- `typedef struct avl_traverser * TVS;`

2.4.2 Funções da API(acrescentadas)

- ```
  long avl_contador(AVL a){
  return a->avl_count
  }
  ```

  Função que devolve o número de nodos de uma AVL.
  ```

Capítulo 3

Makefile

A nossa Makefile foi estruturada de modo a compilar primeiro todos os ficheiros .c para os seus respetivos .o. De seguida, conecta os .o criando um executável program. A compilação é realizada com as flags -Wall -std=c11-g","‘xml2-config –cflags‘"e "‘xml2-config –libs‘"como era pedido no enunciado. Introduzindo o comando make são compilados todos os .c que foram alterados ou que ainda não tenham sido compilados.

Possui também o comando make clean que elimina todos os .o gerados.

```
INCLUDES = interface.o parser.o avl.o estrutura.o
CC = gcc
CFLAGS = -Wall -std=c11 -g
XMLCFLAGS = ‘xml2-config --cflags‘
LIBS2 = ‘xml2-config --libs‘
all: includes program
includes: avl.o estrutura.o parser.o interface.o
interface.o:
$(CC) $(XMLCFLAGS) $(CFLAGS) -c  interface.c

avl.o:
$(CC) $(XMLCFLAGS) -c $(CFLAGS) avl.c

parser.o:
$(CC) $(XMLCFLAGS) -c $(CFLAGS) parser.c

estrutura.o:
$(CC) -c $(CFLAGS) estrutura.c $(XMLCFLAGS)

program: $(INCLUDES)
$(CC)  $(XMLCFLAGS) $(CFLAGS) $(INCLUDES) program.c -o program $(LIBS2)

run: program
./program

clean:
rm *.o
rm program
```

Capítulo 4

Conclusão

Neste projeto foi-nos possível perceber que o encapsulamento, a modularidade dos dados, e os tipos abstratos de dados num projeto de média/grande escala são bastante importantes.

O encapsulamento e os tipos abstratos de dados permitem ocultar elementos da estrutura interna do programa. Isto leva a que os utilizadores possam usá-lo sem ter acesso ao funcionamento das várias funções definidas.

A modularidade permitiu a elaboração de um código muito mais simples. Ou seja, dividindo a estrutura do programa em módulos, reparamos num elevado número de vantagens, como por exemplo, uma melhor organização e estruturação do código, em caso de erro uma maior facilidade em encontrar o sitio do código onde este está (menos linhas por módulo), melhor reutilização e ainda diferentes pessoas poderem trabalhar em módulos diferentes ao mesmo tempo.

As nossas maior dificuldades foram a realização das Queries
"long* top_10_contributors(TAD_istruct qs)","long* top_20_largest_articles(TAD_istruct qs)"e
"long* top_N_articles_with_more_words(int n, TAD_istruct qs)", pelo facto de termos de trabalhar com dois campos ao mesmo tempo, por exemplo,criar uma lista com o número de palavras dos artigos, e associar o id de cada artigo a esse número de palavras. Também tivemos alguma dificuldade em otimizar os tempos de execução das Queries, mas achamos que estes são aceitaveis.