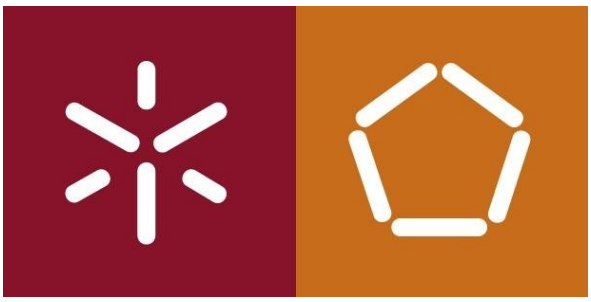
UNIVERSIDADE DO MINHO

Mestrado Integrado em Engenharia Informática



Laboratórios de Informática III

Lucas Ribeiro Pereira - 68547

Tiago João Fernandes Baptista - 75328

Ricardo Ribeiro Pereira - 73577

Conteúdos Abordados

1. Introdução

O projeto de Laboratórios de Informática de 2017/2018 resume-se no desenvolvimento de um sistema capaz de processar ficheiros XML. Estes armazenam vários dados que são utilizadas pela famosa plataforma Stack Overflow. Após o processamento dos documentos fornecidos é pretendido que o projeto desenvolvido seja capaz de usar estruturas de dados auxiliares para resolver uma série de interrogações.

Para tal foram sugeridas pelos docentes que usássemos algumas bibliotecas já feitas, de maneira a facilitar a realização do trabalho, entre elas o libxml2 e a glib. A primeira foi usada na parte inicial do trabalho enquanto a glib foi fulcral para o desenvolvimento do nosso código, visto que usamos estruturas predifinidas na mesma, como hash table, arvores binárias e listas ligadas.

2. Tratamento de Dados

2.1 Tipo concreto de dados

Como estruturas para armazenamento e manipulação de dados usámos tabelas de hash, árvores binárias e em algumas situações, listas ligadas.

Mais especificamente temos:

- Uma tabela de hash para armazenamento de todos os utilizadores;
- Uma tabela de hash para armazenamento de todos os posts;
- Duas tabelas de hash para diferenciação entre perguntas e respostas a uma dada pergunta, sendo que cada elemento de uma hash vai ter uma árvore de posts(com uma determinada data);
- Uma estrutura auxiliar (iteratoraux) utilizada maioritariamente para guardar informação retirada nos foreach.

Ainda na nossa estrutura de dados temos uma lista ligada que é construída no parser, de modo a facilitar a resposta à interrogação 2.

2.2 Parser

O nosso parser é do tipo DOM, este tipo de abordagem permite-nos trabalhar com o documento como se este se tratasse de uma árvore.

Ao iniciarmos o nosso parser começamos por garantir que é possível efetuá-lo, ou seja se o nosso documento não é vazio (NULL). No nosso caso, o documento só tem um children node, depois apenas é usado o xmlGetProp para retirar os elementos que precisamos para povoar as estruturas auxiliares(hashs e trees). É efetuada a iteração ao longo dos próximos nodos através do campo next, até atingir o fim do ficheiro.

Dentro dos parsers, temos algumas variantes no código, dependendo se estamos a processar,users ou posts. Colocamos algumas condições (no parse dos posts) como a verificação do post type, para povoarmos as estruturas que nos interessam assim como a verificação do proprietário de cada post para incrementar a contagem de posts na hash table de users.

```
struct iteratoraux{
   Date begin;
   Date end;
   long questions, answers;
   char* tag;
   GSList* q4;
   long q4_cont;

   TAD_community com;

};
```

```
iterator_aux new_aux(){

   iterator_aux aux = malloc(sizeof(struct iteratoraux));

   aux->questions = 0;
   aux->answers = 0;
   aux->q4 = NULL;
   aux->q4_cont = 0;
   aux->com = NULL;
   return aux;
}
```

```
struct TCD_community{

  GHashTable* users;
  GHashTable* posts;
  GHashTable* questions;
  GHashTable* answers;

  GSList* top_users;
};
```

```
TAD_community new_community(){

  TAD_community com = malloc(sizeof(struct TCD_community));

  com->users = g_hash_table_new((GHashFunc)hashtable_key, (GEqualFunc)hashtable_equal);
  com->posts = g_hash_table_new((GHashFunc)hashtable_key, (GEqualFunc)hashtable_equal);
  com->questions = g_hash_table_new((GHashFunc)hashtable_key, (GEqualFunc)hashtable_equal);
  com->answers = g_hash_table_new((GHashFunc)hashtable_key, (GEqualFunc)hashtable_equal);

  return com;
}
```

2.3 Modularização funcional

Para garantir que o código fosse mais fácil de trabalhar, legível e de fácil deteção e correção de erros foram usados vários ficheiros .c e os seus correspondentes headers(.h). Esta modularização permitiu-nos o desenvolvimento de partes diferentes do trabalho por cada elemento do grupo, o que tornou a execução do trabalho muito mais rápida.

Para garantir um código que respeite as “regras de modularização” foram declaradas as funções de cada módulo nos ficheiros .c e colocados os cabeçalhos dessas mesmas funções nos .h, ao mesmo tempo quando são necessárias funções de um determinado módulo apenas incluíamos os header do módulo correspondente, desta maneira, tudo o que é relativo ao módulo fica encapsulado, isto é restrito ao scope de onde as funções são declaradas.

2.4 Abstração de dados

A abstração de dados tem como objetivo a definicao de um tipo de dado que seja independente de qualquer implementação específica.Desta forma separou-se a definição, da utilização que os dados vão ter. No caso do trabalho começou-se por ter uma TCD_community que é nomeada como *TAD_community , dada pelos docentes, que representa o tipo concreto de dados(no ficheiro interface.h).De seguida passou-se à definição da estrutura que efetivamente se usou e os respetivos campos, a TAD_community,ou seja, o tipo abstrato de dados(no structures.c), onde foram criadas funções que podessem trabalhar sobre elas de maneira a garantir o encapsulamento .

Neste caso definiram-se para os vários campos das estruturas, as funções de get (para obter os dados presente num determinado campo) e de set (para alterar os dados de um determinado campo). Ao garantir esta abstração existe afastamento da maneira de como as estruturas estão implementadas, ficando apenas com as operações/funções definidas sobre elas. De maneira sintética,este tipo de implementação permite usar a TAD sem conhecer os detalhes da sua implementação.

3. Estratégias para resolução das interrogações

3.1 Interrogação 1

Na Interrogação 1 é dado um identificador de um determinado posts e é pedido o seu título e nome de utilizador do autor. Caso o identificador seja de um post que é uma resposta, é necessário retornar o título e o nome de utilizado do autor da pergunta em que essa resposta é dada.

Como existem duas hash tables, uma para os users e outra para os posts em geral, é feito um lookup à hash table dos posts, se esta for uma pergunta é retirada logo a informação pedida. Caso seja uma resposta, é retirado o identificador da pergunta à qual foi dada essa resposta e é feita outra lookup à hash table para retirar a informação pretendida.

3.2 Interrogação 2

Na Interrogação 2 é dado um número N e é pedido o top N de utilizadores com maior número de posts de sempre. Como no parser, à medida que é percorrido o ficheiro xml, vai sendo incrementando o número de posts do respetivo utilizador que criou o posts, o código que está relacionado à interrogação 2 é só uma travessia à hash dos users. A cada user é feita um prepend a uma lista ligada. No fim da travessia é realizado um sort da lista ligada correspondente e esta fica com um “top geral” dos utilizadores com mais posts de sempre. Para finalizar, é truncada a lista em N e feita a conversão da informação para esta ser correspondente com o tipo de retorno desejado pela API.

3.3 Interrogação 3

Na Interrogação 3 é dado um intervalo de tempo com granularidade de dia. É feita uma travessia às nossas hash tables de perguntas e respostas, respeitando o intervalo de tempo dado, e a cada nodo é feito o incremento dos contadores, retornando a soma.

3.4 Interrogação 4

Na Interrogação 4 é dado um intervalo de tempo e uma tag, iniciamos a resolução com uma travessia à hash table de perguntas e é procurado verificar se essa tag ocorre. Se ocorrer, é feito um prepend dessa pergunta a uma lista ligada. No final da travessia, procede-se à organização da nossa lista ligada por cronologia inversa e a respetiva conversao para o tipo de retorno desejado.

3.5 Interrogação 5

Na interrogação 5 é dado um identificador de um utilizador e quer como retorno a informação do seu perfil e os identificadores dos seus últimos 10 posts (ordenados por cronologia inversa). A função parser já faz grande parte do trabalho aqui porque vai atualizando os users à medida que um novo post é adicionado às estruturas de dados (neste caso hash de posts,answers ou questions) portanto fazemos um acesso direto à nossa hash table dos user e retornamos de acordo com os tipos desejados.

3.6 Interrogação 6

Na Interrogação 6 é dado um intervalo de tempo e um número N. Para a resolução é feita uma travessia à hash table das respostas respeitando o intervalo de tempo, de seguida é feito um prepend a uma lista ligada auxiliar. No final, procede-se à organização da lista ligada. Para terminar trunca-se a lista ligada em N e retorna-se a lista de resposta (no tipo respetivo após conversão).

3.7 Interrogação 7

Na Interrogação 7 é dado um intervalo de tempo e um número N. Para a resolução é feita travessia à hash table das perguntas e das respostas e são guardados todos os posts numa lista ligada.De seguida é efetuada a ordenação da lista ligada pelo numero de respostas, que é um campo da estrutura dos posts.De seguida efetuamos o corte (truncamos) de acordo com o N dado, fazendo a respetiva conversão para o tipo desejado.

3.8 Interrogação 8

Na Interrogação 8 é dada uma palavra e um número N. Para a resolução é feita uma travessia à hash table das perguntas e feita a comparação para verificar se a palavra dada está contida no título da pergunta. Caso esteja, é feito um prepend a uma lista ligada auxiliar da pergunta. No final, é realizado uma ordenação dessa lista e efetuamos o corte (truncamos) de acordo com o N dado, fazendo a respetiva conversao para o tipo desejado.

3.10 Interrogação 10

Na interrogação 10 é dado o ID de uma pergunta para a qual é necessário obter a melhor resposta, para determinar a melhor resposta foi usada a seguinte formula : 0.65*score(do post fornecido)+0.25*reputação do user + nºde comentarios(do post fornecido).

Para a resolução é feito um foreach à hashtable das respostas em que apenas guardamos as respostas à pergunta para a qual pretendemos determinar a melhor resposta e guardamo-la numa lista ligada auxiliar.Depois vão ser percorridos todos os elementos desta lista ligada de repostas a aplicar a formula enunciada em cima, é usada uma variável auxiliar de maneira a guardar o maior valor encontrado e uma outra para guardar o id dessa resposta.Após isto apenas falta retornar a variável auxiliar que contem o id da respota que procuravamos.

4. Problemas de otimização

Ao longo da realização do projeto foram aparecendo vários desafios.Dentro deste capitulo do relatório são destacados os maiores problemas que apareceram em termos de escolha de estruturas de dados e também de métodos encontrados para otimizar o código que ia sendo produzido.

Ao nível das estruturas de dados, começou-se por guardarmos os users e os posts em arvores binárias, sendo que o conceito assentava em ordenar a arvore dos post por data de publicação dos mesmos...Algo que se demonstrou impossível pois existiam datas iguais de publicação até ao milissegundo.De maneira a contornar este problema foi pensado em usar, nos posts, arvores que em cada nodo possuiam hash tables mas através de experiências foi verificado que não era o mais eficaz, rápido e prático que poderia ser usado.Passou-se a a dividir os posts em perguntas e respostas e para cada uma delas foram criadas 1 hash ordenada por data em que em cada campo da hash ficaram arvores com os posts usando como key os id dos posts, sendo que todos os elementos destas arvores tem a mesma data até ao dia.

Acabámos por ficar com esta solução pois parece-nos ser a melhor, sendo que ao aumentar o volume de dados, o tamanho da hash apenas vai variar com o numero de dias (a granularidade serão os dias) enquanto se optassemos por exemplo pelos ids teriamos uma hash muito maior.

No caso dos Users optou-se por usar hash tables, pois a maioria dos acessos a informação de um user era direta, por exemplo ser dado o id e ser pedido a short bio.Enquanto se a estrutura usada fosse uma arvore como pensado inicialmente ia haver necessidade de percorrer a arvore... Enquanto com as hash apenas é passado o id do user e obtem-se logo o User pretendido através de um lookup.

Para garantir que apenas se percorre os ficheiros dos dump uma vez, foram usadas algumas estratégias.Verificando o tipo de post que estava a ser parsado e inserindo-o ou na hash das perguntas ou na Hash das respostas e tambem na hash dos posts – uma hash geral que criamos, em que inserimos os posts ordenados por id para nos ajudar na resolução de algumas queries em que não existe intervalo de tempo a condicionar.Outra destas estratégias foi verificar o user que criou o post que estava a ser parsado e incrementar na hash dos users o numero de posts feitos pelo mesmo e acrescentar a uma lista ligada o endereço desse mesmo post, o que nos permitiu responder facilmente à Interrogação 5.

5. Conclusão

Este foi o projeto que até agora criou mais dificuldades a todos os elementos do nosso grupo, tanto a nível de programação como de raciocínio para estruturação do trabalho.

Para começar deparamo-nos com um volume de informação muito superior ao que estamos habituados, houve necessidade de pensar e otimizar o código para garantir que este executasse em tempo mais ou menos aceitável.

A utilização da glib foi outro dos pontos que suscitou grandes dúvidas, principalmente para perceber como utilizamos algumas das funções predifinidas que nos são disponibilizadas, como por exemplo o foreach. Onde houveram dificuldades na definição das funções de travessia e em perceber que estrutura auxiliar teriamos que utilizar.Por outro lado, o facto dos nossos dados serem em xml fez-nos explorar a libxml2 e os tipos de parser que poderiam ser usados, fruto da nossa pesquisa decidimos usar a estrutura DOM , que será a mais indicada para o volume de dados do input.

Por último, a aplicação de regras de “boa programação” como a abstração de dados e encapsulamento, pareceu-nos um pouco confusa ao início mas acabamos por admitir que facilitava bastante o nosso trabalho a nivel individual, visto que cada elemento podia facilmente desenvolver um modulo independentemente dos outros e depois juntá-lo ao código já desenvolvido. Para além disto apercebemo-nos que encontrar e resolver erros foi muito mais rápido e fácil.