Projeto de Laboratórios de Informática 3
Grupo 26

José Pinto (A81317) Luís Correia (A81141) Pedro Barbosa (A82068)

5 de Maio de 2018

Resumo

Neste projeto da disciplina de Laboratórios de Informática 3 (LI3) do Mestrado Integrado em Engenharia Informática da Universidade do Minho foi-nos proposto desenvolver um sistema em C capaz de processar ficheiros XML que armazenam informações utilizadas pelo Stack Overflow, podendo depois executar um conjunto de querys de forma eficiente.

Conteúdo

1 Introdução

Neste relatório vamos primeiramente falar das estruturas de dados utilizadas e dos vários módulos criados, terminando com a explicação das estratégias usadas em cada query.

2 Dependências externas

Este projeto tem duas dependências externas, as bibliotecas libxml2 e glib.

3 Módulos

Nesta secção vamos falar dos módulos criados para auxiliar o desenvolvimento deste projeto.

3.1 Post

Este módulo corresponde à implementação de um post do Stack Overflow.

3.1.1 Estrutura de dados

A informacão relativa a um utilizador é guardada na estrutura *post*:

```
struct post {
        //Identifica o tipo de post (Pergunta ou Resposta)
        char postTypeId;
        QUESTION q;
        ANSWER a;
        //ID do post
        int id;
        //ID do utilizador que fez o post
        char * ownerUserId;
        //Data da criacao do post
        Date creationDate;
};
```

Esta estrutura é a implementação concreta do tipo de dados abstrato POST declarado no header:

```
typedef struct post * POST;
```

Como a informação de um post varia conforme seja uma pergunta ou uma resposta, foram criadas as estrututras QUESTION e ANSWER:

```
typedef struct question{
        //Titulo
        char *title;
        //Numero de respostas
        int nanswers;
        //Tags
        GSList *tags;
        //Ultima atividade do post
        Date lastActivityDate;
}*QUESTION;

typedef struct answer{
        // ParentID (ID do post a que a resposta pertence)
        int parentId;
        // Numero de comentarios de uma resposta
```

```
        int comments;
        // Score de uma resposta
        int score;
}*ANSWER;
```

O membro *tags* é uma lista ligada que guarda os ids das tags de um post.

3.1.2 Funções

Devido à dicotomia dos posts, existem duas funções distintas para a criação de uma instância de um POST. Estas criam cópias de todos parâmetros recebidos com tipo não primitivo, para garantir o encapsulamento dos dados.

O acesso exterior aos membros da estrutura é realizado exclusivamente através de funções. Todas as variáveis podem ser consultadas, sendo o resultado da consulta de tipos não primitivos uma cópia destes. Nenhum membro da estrutura pode ser alterado após a iniciaização.

Existe também uma função que liberta a memória alocada para a estrutura.

3.2 Posts

O objetivo deste módulo é permitir guardar e processar a informação de todos os posts de uma comunidade do Stack Overflow

3.2.1 Estrutura de dados

A estrutura utilizada é *TCD_posts*:

```
struct TCD_posts {
        GSList *list;
        GHashTable *hash;
        GHashTable *months_hash;
};
```

O membro *list* é uma lista ligada que contêm todos os posts por ordem cronológica inversa. *hash* é uma hashtable que associa o id de um *POST* à posicão deste na lista. *months_hash* é uma hashtable que faz a associação entre um mês(e o ano) e o primeiro post desse mês. Assim, quando um dos parâmetros de uma interrogação é um intervalo de tempo, é possível começar a percorrer a lista dos posts num ponto já muito próximo do intervalo pedido, evitando percorrer a lista desnecessariamente. Desta forma o tempo de execução dessas interrogações é muito menor. Poderíamos ter optador por guardar o primeiro post de cada dia no entanto a melhoria no tempo de execução não compensaria o tamanho acrescido da hashtable.

Esta estrutura é a implementação concreta do tipo de dados abstrato *Posts* declarado no header:

```
typedef struct TCD_posts * Posts;
```

Como é necessário percorrer a lista dos posts em várias interrogações também foi definido o tipo *PostsList* para manter a abstração:

```
typedef GSList * PostsList;
```

3.2.2 Funções

Depois de uma instância de *Posts* ser inicializada com *init_posts*, podem ser utilizadas funções para adicionar/remover posts. Tal como no módulo anterior existe uma função de adição para as perguntas e outras para as respostas. Estas funções recebem a informação através de vários parâmetros e usam internamente as funções do módulo Post para assegurar o encapsulamento.

Para garantir o correto funcionamento do módulo é necessário que o a função *finalize_posts* seja invocada após todos os posts serem adicionados. O bom funcionamento do módulo estar

dependente de um fator externo pode não ser ideal mas foi um compromisso realizado para garantir um tempo de execução aceitável. É mais eficiente ordenar a *list* e preencher a *months_hash* depois de todos os posts serem inseridos.

Existem várias funções que permitem procurar um post, ou a localização deste na lista, de acordo com certos parâmetros (id, data, etc). Também existem funções que permitem percorrer a lista de posts e obter o *POST* dessa posição. Como estas últimas funções recebem/devolvem o tipo abstrato *PostsList* o encapsulamento é garantido.

Também está definida uma função para libertar a memória alocada.

3.3 MyUser

Este módulo corresponde à implementação de um utilizador do Stack Overflow, contendo a informação relativa a um *user*.

3.3.1 Estrutura de dados

A informacão relativa a um utilizador é guardada na estrutura *user_ht*:

```
struct user_ht{
        //ID do user
        int id;
        //Nome do utilizador
        char *name;
        // ShortBio do utilizador
        char *shortBio;
        // Numero de posts do utilizador
        unsigned short nr_posts;
        // Ultimo post do utilizador
        int lastPost;
        // Reputacao do utilizador
        int reputation;
};
```

O último post do utilizador corresponde ao id do post mais recente deste e a sua utilidade será esclarecida na secção sobre a interrogação 5. Os restantes membros da estrutura contêm informação necessária para responder às interrogações.

Esta estrutura é a implementação concreta do tipo de dados abstrato MY_USER declarado no header:

```
typedef struct user_ht * MY_USER;
```

3.3.2 Funções

A criação de uma instância do MY_USER é feita através de um método de construção parametrizado. Esta função tem o cuidado de criar cópias das strings recebidas para garantir o encapsulmanento dos dados.

O acesso exterior aos membros da estrutura é realizado exclusivamente através de funções. Todos as variáveis podem ser consultadas, sendo o resultado da consulta de tipos não primitivos uma cópia destes. As únicas que podem ser modificadas após a inicialização são o *lastPost* e o *nr_posts* devido ao facto de que a informação destes não se encontrar no ficheiro Users.xml e ser obtida através do posterior processamento do ficheiro Posts.xml

Por último, existe uma função que liberta a memória alocada para a estrutura.

3.4 Users

O objetivo deste módulo é permitir guardar e processar a informação de todos os utilizadores de uma comunidade do Stack Overflow

3.4.1 Estrutura de dados

A estrutura utilizada é *users*:

```
struct users {
        GHashTable* hash;
        GSList* users_by_nr_posts;
        GSList* users_by_reputation;
};
```

hash é uma hashtable que associa o id a um *USER*. *users_by_nr_posts* e *users_by_nr_reputation* são listas ligadas de *USER*, ordenadas em ordem decrescente de acordo com o número de posts e a reputação de cada *USER* respetivamente.

Esta estrutura é a implementação concreta do tipo de dados abstrato MY_USER declarado no header:

```
typedef struct users * USERS;
```

3.4.2 Funções

Tal como no módulo *Posts*, depois de uma instância de *Users* ser inicializada com *init_users*, podem ser utilizadas funções para adicionar/remover users. A função de adição recebe a informação através de vários parâmetros e usam internamente as funções do módulo User para assegurar o encapsulamento.

Existem duas funções para alterar o número de posts e o último post de um dado utlilizador. Assim, é possível percorrer a lista de posts contida numa instância de *Posts* para alterar estas variáveis mesmo após as instâncias de *USER* já terem sido criadas.

Por razões semelhantes às explicadas no módulo *Posts* (neste caso é a ordenação das duas listas), é necessário que o a função *finalize_users* seja invocada após todos os utilizadores serem adicionados/alterados.

A funcionalidade principal deste módulo é a função *find_user*, que permite obter a informação de um utilizador dado o seu id, e duas funções que percorrem uma das listas da estrura *users* e criam e retornam uma *LONG_list* com os N utilizadores com mais posts ou mais reputação.

Também está definida uma função para libertar a memória alocada.

3.5 My Date

O módulo *mydate* é um pequeno módulo que possui algumas funções relacionadas com datas que são utlilizadas por vários módulos.

3.6 Tags

Este módulo é responsável por guardar e processar informação sobre tags de posts

3.6.1 Estrutura de dados

A estrutura de dados deste módulo é apenas uma hash table que associa o nome de uma tag ao seu id. Para o propósito da abstração esta tabela é representada pelo tipo *TAGS*:

```
typedef GHashTable * TAGS;
```

3.6.2 Funções

As únicas funções presentes neste módulo são as de inicialização, inserção, procura e libertação de memória. Existe apenas mais uma função utilizada para converter a informação sobre tags presente no Posts.xml numa lista para passar à função *create_question* do módulo *Post*

4 Interrogações

Nesta secção vamos explicar os métodos usados para cada uma das interrogações e as estratégias para melhoramento de desempenho.

4.1 Interrogação 1

Nesta interrogação usa-se primeiramente uma função para devolver o post com o ID dado. Caso o post seja uma pergunta, procura-se o user que fez esse post para que se devolva o nome do autor junto com o título do post. Se o post for uma resposta, devolve-se o nome do user que fez a resposta e procura-se pela pergunta a que essa resposta diz respeito para se retornar também o título da pergunta.

4.2 Interrogação 2

A struct dos users, formada por duas listas e uma hashtable, já possui uma lista com todos os users ordenados pelo número de posts, pelo que apenas é necessária uma função que percorra essa mesma lista e devolva uma nova com os N users pretendidos.

4.3 Interrogação 3

No módulo post_list temos a lista de posts ordenada da mais recente para a mais antiga. Procuramos com a ajuda da hashtable months_hash o post anterior ao intervalo de tempo dado, retornando a lista de posts a partir desse post. Com isto, percorre-se a lista dentro do intervalo de tempo contabilizando o número de perguntas e respostas.

4.4 Interrogação 4

Inicialmente utiliza-se a hashtable months_hash para encontrar o último post que ainda faz parte do intervalo de tempo em questão. De seguida, percorre-se a lista de posts desde esse mesmo post até chegar à data do begin. À medida que se percorre cada post verifica-se se é uma pergunta e se contém a tag. Se estas condições se verificarem adicionamos o ID do post a uma lista ligada. Por fim, copiam-se todos os IDs da lista ligada para uma LONG_list e retornamos a mesma.

4.5 Interrogação 5

A partir do ID dado, procura-se na hashtable dos users pelo user com esse ID. Com isto obtém-se a informação do seu perfil (short bio), o número de posts e o último post do user. Começa-se a percorrer a lista dos posts desde o último post até serem encontrados 10 ou já se tiver percorrido todos os posts do user. Cada post pertencente ao user é adicionado a um array Caso não sejam encontrados 10 posts, o resto do array fica a 0.

4.6 Interrogação 6

Utiliza-se a hashtable months_hash para encontrar o último post que ainda faz parte do intervalo de tempo em questão. De seguida, percorre-se a lista de posts desde esse mesmo post até chegar à data do begin. Para cada post, caso o post seja uma resposta é adicionado a uma lista ligada temporária. Após percorrer todos os posts, ordena-se a lista temporária segundo a score de cada

resposta. Por fim, copiamos as primeiras N respostas para uma LONG_list e retornamos a mesma. Caso não sejam retornadas N respostas no intervalo dado, preenchemos o resto da lista com zeros.

4.7 Interrogação 7

No módulo post_list existe a lista de posts ordenada da mais recente para a mais antiga. Procuramos com a ajuda da hashtable months_hash o post anterior ao intervalo de tempo dado, retornando a lista de posts a partir desse post. Percorre-se a lista de posts dentro do intervalo de tempo dado e caso um post seja uma pergunta, adiciona-se a uma lista ligada temporária. No fim de se percorrer a lista de posts, ordena-se a lista ligada temporária de acordo com o número de respostas, para que se possa retornar os IDs das N perguntas com mais respostas. Se não foram encontrados N perguntas no intervalo de tempo dado, o resto da LONG_list fica a 0.

4.8 Interrogação 8

Para esta interrogação começamos por percorrer a lista dos posts, verificar se é uma pergunta e se a palavra pertence ao seu título, caso pertença adiciona-se o ID da pergunta a uma LONG_list e devolve-se a mesma. Caso não sejam retornadas N perguntas com a palavra no título, o resto da lista é preenchida com zeros.

4.9 Interrogação 9

Para resolver esta interrogação foi preciso criar três queues, na queue1 vamos guardar respostas do utilizador com ID1 e na queue2 vamos guardar respostas do utilizador com ID2 e na queuef guardamos o ID dos posts onde participam ambos os utilizadores. À medida que percorremos a lista dos posts, quando se encontra uma resposta verificamos quem a publicou, se for o ID1/2 verificamos o parentOwnerId, se este for o ID2/1 adicionamos o ID da pergunta à queuef, se o parentOwnerId não for o ID2/1, vamos à queue2/1 verificar se tem alguma respostas com o mesmo parentOwnerId, se tiver adicionamos à queuef. Caso nenhuma das condições se verifique adicionamos à queue1 ou à queue2. Quando o post que encontramos é uma pergunta se o parentOwner for o ID1/2, vamos à queuef verificar se o post está lá, se estiver adicionamos à LONG_list, caso contrário não faz nada.

4.10 Interrogação 10

Primeiramente, utiliza-se a hashtable hash da estrutura dos posts, para obter a lastactivityDate do post. De seguida, utiliza-se a hashtable months_hash para encontrar o primeiro post que antecede o post cuja data é a lastActivityDate. Por fim percorremos a lista até à creationDate da pergunta, verificando se o post encontrado é uma resposta à pergunta. Por fim é calculado a score de cada post e comparam-se todas as scores até encontrar a melhor retornado o ID dessa resposta.

4.11 Interrogação 11

Inicialmente, fazemos uma lista com os N utilizadores com melhor reputação. Seguidamente utilizamos a hashtable months_hash para encontrar o último post que ainda faz parte do intervalo de tempo em questão e percorremos a lista de posts desde esse mesmo post até chegar à data do begin. À medida que se percorre a lista dos posts, caso seja uma pergunta e o owner seja um dos N utilizadores com melhor reputação vamos adicionar a tag a uma lista. Caso a tag já esteja presente na mesma, incrementamos o número de vezes que ocorre. No final, ordenamos a lista das tags, copiamos para uma LONG_list e retiramos as N primeiras. Caso não sejam contadas N tags, enchemos o resto da lista com zeros.