Projeto de Laboratórios de Informática 3
Grupo 5

Henrique José Carvalho Faria a82200 José André Martins Pereira a82880
Ricardo André Petronilho a81744

5 de Maio de 2018

Resumo

No âmbito da Unidade Curricular de Laboratórios de Informática 3 (LI3), do Mestrado Integrado em Engenharia Informática da Universidade do Minho, foi nos proposto a elaboração de um projeto que consiste na manipulação e tratamento de uma base de dados armazenada em ficheiros com formato - `.xml` - com o objetivo a responder a interrogações - querries - sobre a base de dados através de uma estrutura de dados programada na linguagem de programação C.

Conteúdo

0.1 Introdução

Este relatório tem o propósito de esclarecer tanto as estratégias aplicadas em cada uma das interrogações resolvidas como as estruturas e tipo de dados utilizadas no desenvolvimento do projeto. Sendo abordados especificamente os seguintes tópicos:

- tipo concreto de dados
- estruturas de dados usada
- modularização funcional
- abstração de dados
- estratégias seguidas em cada uma das interrogações
- estratégias para melhoramento de desempenho

0.2 Tipo concreto de dados

Na evolução do projeto utilizamos os seguintes tipo concretos de dados:

- `TCD_ARRAY_LIST` (1)
- `TCD_PPOST` (2)
- `TCD_DATAS` (2)
- `TCD_ITERATOR` (2)
- `TCD_community` (3)
- `TCD_HTABLE` (4)
- `TCD_POST` (5)
- `TCD_REAL_USER` (6)
- `TCD_TAG` (7)

O `TCD_ARRAY_LIST` (1) foi utilizado na resolução das interrogações e na estrutura interna do programa. Foi especialmente útil uma vez que é uma estrutura que permite armazenar qualquer tipo de dados, sendo por isso genérica e assim possiblita uma gestão e reutilização da mesma de uma forma facilitada. Visto que este projeto tem como objetivo a aplicação da modularidade proveninte do paradigma orientado a objetos, esta estrutura foi baseada precisamente na class ArrayList da linguagem de programação Java.

A `TCD_DATAS` (2) foi utilizado nas interrogações que involvem intrevalos de tempo. As suas utilidades mais vantajosas são a organização dos posts pela data de criação e o acesso direto à memória. O `TCD_ITERATOR` (2) permite de uma forma intuitiva percorrer o `TCD_DATAS` dado um intervalo de tempo, note-se que a implementação deste tipo, foi baseada na *Interface Iterator* da linguagem Java, a utilização deste Iterator, permite um maior modularidade da estrutura `TCD_DATAS`. O `TCD_PPOST` (2) é o tipo usado pelo `TCD_DATAS` para armazenar a informação relativa a cada post, especificamente os posts de cada dia estão armazenados num `TCD_ARRAY_LIST`.

No seguinte exemplo é demonstrado a utilização do `TCD_ITERATOR`:

```
// protótipo do construtor do iterador
TAD_ITERATOR it = datasIterator(TAD_DATAS datas, Date begin, Date end);

while(itHasNext(it)){
    TAD_ARRAY_LIST posts = itNext(datas, it);
    ...
}
```

O `TCD_community` foi criado pela equipa docente com o propósito de todos os grupos terem tipos de dados comuns, de modo a facilitar a avaliação do projeto.

O `TCD_HTABLE` foi desenvolvido com o objetivo de armazenar os diferentes tipos de dados utilizados ao longo do projeto, especificamente o `TCD_POST` e o `TCD_REAL_USER`, possibilitando o aumento do desempenho e velocidade na procura de dados.

O `TCD_POST` foi utilizado para armazenar a informação relativa a cada post da base de dados.

O `TCD_REAL_USER` foi utilizado para armazenar a informação relativa a cada utilizador da base de dados.

O `TCD_TAG` foi utilizado para armazenar a informação relativa a cada tag da base de dados.

0.3 Estruturas de dados usada

Nesta secção apenas explicamos parte das estruturas de dados que achamos de mais difícil compreensão ao utilizador alheio.

O TCD_ARRAY_LIST é implementado com a seguinte estrutura:

```
typedef struct TCD_ARRAY_LIST{

    /**\brief array de apontadores*/
    void** array;
    /**\brief número de elementos contidos no ArrayList*/
    int ocupados;
    /**\brief capacidade do ArrayList*/
    int dim;

} TCD_ARRAY_LIST;
```

É utilizado um array de apontadores do tipo void*, uma vez que é o tipo de dados mais genérico da linguagem C.

O TCD_DATAS, TCD_ITERATOR e TCD_PPOST são implementados com as seguintes estruturas:

```
/**\brief Estrutura que armazena um Post*/
typedef struct TCD_PPOST{
    /**\brief id do post*/
    long post_id;
    /**\brief id do utilizador que criou o post*/
    long user_id;
}*TCD_PPOST;

/**\brief Estrutura que armazena lista de posts consoante a sua data.*/
typedef struct TCD_DATAS{
    /**\brief primeiro ano de informação*/
    int primeiro_ano;
    /**\brief número de anos de informação*/
    int n_anos;
    /**\brief array tri-dimensional que contém os posts, alocados por ano, mês e dia*/
    TAD_ARRAY_LIST*** array_info;
}TCD_DATAS;

/**\brief Estrutura que itera/percorre a Estrutura das Datas.*/
typedef struct TCD_ITERATOR{
    /**\brief arrayList que contém os posts da data cur */
    TAD_ARRAY_LIST arr; // array_info que o iterador aponta neste momento
    /**\brief data atual que o iterator está a apontar*/
    Date cur;
    /**\brief data final do iterador*/
    Date end;

} TCD_ITERATOR;
```

O primeiro ano - primeiro_ano - na estrutura TCD_DATAS, está pré-estabelicido num define (2008), visto que sabemos previamente a data de criação da base de dados.

O número de anos - `n_anos` - é calculado através de uma função que obtém o ano atual da máquina, subtraindo ao ano inicial - `primeiro_ano`.

O `TCD_community` é implementado com a seguinte estrutura:

```
typedef struct TCD_community{

   /**\brief Tabela de Hash que contem os posts*/
   TAD_HTABLE posts;
   /**\brief Tabela de Hash que contem os utilizadores*/
   TAD_HTABLE users;
   /**\brief ArrayList que contem as tags*/
   TAD_ARRAY_LIST tags;
   /**\brief Estrutura que contem as datas*/
   TAD_DATAS datas;

} TCD_community;
```

Esta estrutura armazena toda a informação necessária à execução do programa.

O `TCD_HTABLE` é implementado com a seguinte estrutura:

```
/**\brief célula que armazena informação na Tabela de Hash*/
typedef struct cell{

    /**\brief id da célula*/
    long id;
    /**\brief indica se a célula está a ser usada*/
    long status;
    /**\brief contador de colisões obtidas quando foi armazenada informação na célula*/
    long colisoes;
    /**\brief informação que a célula contém*/
    void *dados;

} HTABLE_CELL;

/**\brief Estrutura que armazena e gere os elementos da Tabela de Hash*/
typedef struct TCD_HTABLE{

    /**\brief array de células que contêm a informação*/
    HTABLE_CELL* celulas;
    /**\brief número de elementos contidos na Tabela de Hash*/
    long ocupados;
    /**\brief capacidade da Tabela de Hash*/
    long dim;

} TCD_HTABLE;
```

Esta estrutura implementa uma tabela de hash com fator de ocupação máximo de 70% e quadratic probbing. Note-se que cada célula - `HTABLE_CELL` - da tabela contém um apontador do tipo void* para a sua respetiva informação - `dados` - o que permite maior versatibilidade uma vez que pode armazenar qualquer tipo de informação, por este motivo a mesma implementação da tabela de hash funciona tanto para alocar as informações dos utilizadores, como a dos posts, em diferentes tabelas de hash.

Foi desenvolvido um sistema de resolução de colisões de forma a aumentar o desempenho na procura de informação na tabela de hash.

O TCD_POST é implementado com a seguinte estrutura:

```
typedef struct TCD_POST{

    /**\brief número de comentários*/
    int commentCount;; // questão 10
    /**\brief */
    int score; // questão 10
    /**\brief Número de respostas ao Post
    int answerCount; // nº de respostas a 1 pergunta - questão 7
    /**\brief tipo de Post, se tipo for 1 é uma questão, se for 2 é uma
    resposta, se for 0 nem é pergunta nem resposta*/
    int tipo; // 1-> question, 2->answer , 0-> neither
    /**\brief ID do utilizador que fez o Post*/
    long user_id; // através da hash function obter onde se localiza
    na tabela de hash o utilizador com este id
    /**\brief ID da pergunta, caso o Post seja uma resposta*/
    long parent_id; // caso seja uma resposta
    /**\brief tags para este Post*/
    char* tags; // array de apontadores para tags -> serve para questão 9
    /**\brief titulo do Post*/
    char* titulo; // questão 8
    /**\brief Número de upvotes dos Posts do utilizador*/
    int upvotes;
    /**\brief Número de downvote dos Posts do utilizador*/
    int downvotes;
    /**\brief Data do Post*/
    Date data;

} TCD_POST;
```

O TCD_REAL_USER é implementado com a seguinte estrutura:

```
typedef struct TCD_REAL_USER{

    /**\brief Reputação do utilizador*/
    int reputacao;
    /**\brief Nome do utilizador*/
    char* nome;
    /**\brief Biografia do utilizador*/
    char* bio;
    /**\brief ArrayList que contém os ID's dos posts do utilizador*/
    TAD_ARRAY_LIST posts;
    /**\brief Número de upvotes dos posts do utilizador*/
    int upvotes;
    /**\brief Número de downvote dos posts do utilizador*/
    int downvotes;

} TCD_REAL_USER;
```

O TCD_TAG é implementado com a seguinte estrutura:

```
typedef struct TCD_TAG{
    /**\brief tagId - id da tag*/
    int tagId;
    /**\brief nome - nome da tag*/
    char* nome;
} TCD_TAG;
```

0.4 Modularização funcional

De forma a facilitar o desenvolvimento do projeto, separamos o mesmo em módulos diferentes e bastante isolados. Assim foi permitido aos diferentes elementos do nosso grupo trabalharem de forma autónoma sem haver conflitos, em partes diferentes do mesmo. A reutilização do código foi simplificada e facilitada. Quando o grupo se deparou com erros de compilação ou execução do código foi mais intuitivo isolar os respetivos erros, assim o desenvolvimento do projeto foi o mais próximo possível do paradigma orientado a objetos.

Na implementação do nosso código, inclui uma API pública e privada, este procedimento permite uma coperação autónoma uma vez que é possível ao longo do desenvolvimento de cada módulo a alteração da implementação de cada função, e no entanto é presevada a sua API.

0.5 Abstração de dados

Durante a evolução do nosso projeto a abstração de dados foi implementada através da identificação - `TAD` - nos diferentes tipos concretos de dados - `TCD`.

Este hábito de produção de software permite ocultar a constituição interna de cada estrutura de dados utilizada, uma vez que torna impossível o acesso direto á mesma. Desta forma apenas se tem conhecimento do resultado fornececido pela função e não da sua implementação. Ou seja, aplica-se em C o que é comum nas linguagens de programação orientada a objetos.

A abstração de dados é implementada da seguinte forma:

```
typedef struct TCD_tipoDeDados *TAD_tipoDeDados;
```

Note-se que esta definição - `typedef` - é realizada no ficheiro com formato `.h` uma ves que a estrutura de dados está implementada no ficheiro `.c` tornando impossivel o acesso á estrutura.

0.6 Estratégias seguidas em cada uma das interrogações

0.6.1 Interrogação 1

O primeiro passo foi verificar se o id dado como argumento corresponde ao id de uma pergunta ou de uma resposta. Caso seja uma pergunta basta obter o nome e título do utilizador que a criou. Tratando-se de uma resposta identifica-se a pergunta correspondente e de seguida procura-se o nome de utilizador da mesma.

0.6.2 Interrogação 2

Foram criadas duas estruturas ArrayList - `ids_users` e `tops` - para armazenar o id de cada utilizador e armazenar o número de posts do utilizador, respetivamente. De seguida obtém-se a lista de posts de cada utilizador e percorre-se a mesma para verificar efetivamente o número de posts (perguntas e repostas) existentes. Assim, insere-se o número de posts contado, ordenadamente no ArrayList `tops`, e simultaneamente adiciona-se no mesmo índice no ArrayList `ids_users`, o respetivo id do utilizador. Ao fim de percorrer todos os utilizadores o ArrayList `ids_users` está ordenado com os ids dos utilizadores com mais posts.

0.6.3 Interrogação 3

Dado um intervalo de tempo arbitário, percorre-se o mesmo tomando partido do iterador, verificando-se o tipo de cada post, incrementando o contador respetivo, caso seja uma pergunta ou uma resposta. No fim é retornado o número de perguntas e respostas total na base de dados.

0.6.4 Interrogação 4

Dado um intervalo de tempo arbitário, percorre-se o mesmo tomando partido do iterador, verificando-se através do tipo do post, se o mesmo é uma pergunta. De seguida verifica-se a existência da tag dada como argumento em cada um dos posts. Atente-se que não foi necessário recorrer a uma função de ordenação, visto que se tirou partido da ordenação natural que o iterador fornece.

0.6.5 Interrogação 5

Dado o id do utilizador, obtêm-se a biografia - `shot_bio` - do mesmo e 10 últimos posts - `post_history` - criados pelo utilizador, tomando partido do algoritmo quickSort, para ordenar o posts por ordem decrescente da sua criação.

0.6.6 Interrogação 6

Dado um intervalo de tempo arbitário, percorre-se o mesmo tomando partido do iterador, para verificarpost a post se se trata de uma resposta. De seguida, é contado o número de votos favoráveis de cada post que passou a verificação e é inserido por ordem decrescente de votação usand o id - `postId`.

0.6.7 Interrogação 7

Dado um intervalo de tempo arbitário, percorre-se o mesmo tomando partido do iterador, para verificar post a post se se trata de uma pergunta. De seguida, são contadas o número de respostas de cada post que passou a verificação e este é inserido por ordem decrescente da quantidade usandi o id - `postId`. Note-se que são usados os ids dos utilizadores para ordenar, também por ordem decrescente posts com o mesmo número de respostas.

0.6.8 Interrogação 8

Dado uma tag, percorre-se a tabela de hash dos posts, verificando-se se o post é um pergunta e se a tag dada está contida no título do post, em caso afirmativo acrescenta-se o id do respetivo post ao arraylist que armazena os ids dos posts. No final tomando partido do algoritmo quickSort, ordena-se o arraylist dos posts por ordem decrescente da sua criação. Atende-se que caso que o número de posts seja inferior ao N dado, apenas se retorna uma lista contendo esse número de posts.

0.6.9 Interrogação 9

Dado os id's - `id1, id2` - de dois utilizadores, percorre-se os posts do utilizador 1, caso estes sejam perguntas, percorre-se as respostas do utlizador 2, comparando-se o id da pergunta - `parent_id` - associada a cada respota, com o id da pergunta do utlizador 1. Caso o post do utlizador 1 seja uma resposta, compara-se o id da pergunta - `parent_id` - associada com o id de todas as respostas do utlizador 2. Realiza-se este procedimento para o utlizador 2. Quando existe uma situação de compartilha de posts entre ambos os utilizadores, ou seja, uma pergunta onde os mesmos interagem através de respostas, ou respostas feitas por um dos utilizadores face à pergunta do outro, o id dessa pergunta adicionada de forma ordenada por cronologia inversa.

0.6.10 Interrogação 10

Dado o id de uma pergunta percorre-se a tabela de hash dos posts procurando-se por respostas à mesma. Caso uma resposta seja encontrada, identifica-se o utilizador que a cirou e obtem-se a reputação e votação desse utilizador, obtem-se também o score e número de comentários da resposta. Aplica-se estes quatro valores obtidos a uma função dada que calcula a respetiva média. Utiliza-se este procedimento para todas as respostas encontradas, guardando-se a maior média e o respetivo id dessa resposta.

0.6.11 Interrogação 11

Dado um intervalo arbitrário e um número também arbitrário(N),é percorrida a estrutura texttttTCD_DATAS, nessse intervalo, para encontrar os N utilizadores mais famosos com publicações entre as datas fornecidas.De seguida percorrem-se os posts de cada utilizador e, a cada um daqueles que são perguntas e foram feitos no intervalo de tempo pretendido retiram-se as tags para contagem. As tags retiradas na forma de string única são individualizadas e de seguida procede-se a um processo de procura numa lista de tags para verificar se já foram registadas, caso a tag tenha sido inserida na lista incrementa-se o contador da mesma, caso contrário a tag é adicionada ao final da lista e o contador inicializado a 1. Por fim procede-se a uma ordenação a partir da cauda para a tag recém adicionada, ordenação que compara os contadores deixando-os por ordem crescente, e para as tags com contadores iguais compara o id das mesmas ficando estas com uma ordenação final tanbém crescente;

0.7 Estratégias para melhoramento de desempenho

A estratégia utilizada na globalStruct foi dividir a informação em duas hash tables diferentes com users e posts, interligadas, um arraylist com a informação das tags, que foi especialmente útil para a elaboração da questão 11 e por fim um array de 3 dimensões, para organizar os posts pela data de criação do mesmo, facilitando as questões que involvem intrevalos arbitrários, obtendo-se assim acesso direto á memória.

Para melhoramento da procura da hash table, decidiu-se aplicar um tratamento de colisões, com *quadratic probbing*. As colisões permitem uma procura mais rápida evitando problemas de procuras desnecessárias. Também foi aplicado uma avaliação da ocupação, caso esta seja superior a 70%, triplicamos o tamanho da hash table, garantindo melhor procura, visto que os elementos da hash ficam mais distribuídos.

Com a finalidade de melhorar a procura de informação decidiu-se interligar as hash tables dos users e posts. A cada post foi adicionado o id - `user_id` - do utilizador que o fez, visto que a função de hash dos users recebe o `user_id`, consegue-se acesso direto à informação do utilizador. Da mesma forma, a cada utilizador está associado um arrayList que contém os id's dos posts que o o utilizador criou.