Projeto de Laboratórios de Informática III
Grupo 59

João Nunes (A82300) Luís Braga (A82088) Luís Martins (A82298)

1 de Maio de 2018

Conteúdo

1 Introdução

Este projeto foi solicitado pelos docentes da unidade curricular de Laboratórios de Informática III e tem como principal objetivo a realização de um programa que permita analisar os dados presentes em dumps do Stack Overflow e extrair a partir daí a informação útil tal como a reputação dos utilizadores, o número de posts, etc.

Outros objetivos estão também associados a este, tais como a consolidação de conhecimentos adquiridos em unidades curriculares anteriores tais como Programação Imperativa e Algoritmos e Complexidade.

O principal desafio deste projeto seria a programação em larga escala, uma vez que o programa passa por milhares de dados aumentando assim a complexidade do trabalho. Para tornar possível a realização deste projeto foi nos introduzidos novos princípios de programação tais como, a modularidade e encapsulamento de dados.

2 Descrição dos Módulos

2.1 Ficheiros

Os ficheiros que usamos para obter a informação e guardar o seu respetivo conteúdo foram os três dumps relativos ao Posts ao Users e ao Tags.

Os três dumps continham informações relevantes para a eventual resposta de cada uma das queries.

Como estes ficheiros estavam escritos em XML, levou-nos a criar um parser de modo a poder "traduzir"e guardar a informação pertinente nas nossas estruturas.

2.2 Estruturas

2.2.1 TAD_community

```
typedef struct TCD_community{
        Posts *posts;
        Users *users;
        LTag tags;
        LDate datas;
        long sizeP;
        long sizeU;
        long sizeT;
}*TAD_community;
```

A estrutura TAD_community é o módulo onde vão ser armazenadas todas as informações retiradas dos dumps que possam vir a ser úteis para responder as queries. Como se trata de uma larga quantia de informação, é necessário guardá-la numa estrutura adequada, estrutura essa que é possivél verificar em cima.

A estrutura corresponde a um misto de estruturas, irá conter uma Hash Table para os users, uma Hash Table para os posts, listas ligadas para as tags e para as datas e os tamanhos correspondentes para as Hashs e para a lista ligada.

Métodos principais:

- `TAD_community initMyEstrutura();`

Inicia a TAD_community, colocando-a a NULL.

- `TAD_community loadMyEstrutura(long sizep, long sizeu);`

Inicia as estruturas Users e Posts associadas a TAD_community alocando espaço e colocando a NULL todos os elementos das Hashs dos Posts e dos Users.

- `void LItoLDL(LONG_list r, LInt *l);`

Converte uma lista ligada numa LONG_list.

- `void cleanPosts(TAD_community *estrutura);`

Liberta o espaço alocado a estrutura Posts.

- `void cleanUsers(TAD_community *estrutura);`

Liberta o espaço alocado a estrutura Users.

- `void cleanTags(TAD_community *estrutura);`

Liberta o espaço alocado a estrutura Tags.

- `void cleanDatas(TAD_community *estrutura);`

Liberta o espaço alocado a estrutura Datas.

Os restantes métodos são (gets e sets) são para garantir ainda mais o encapsulamento.

2.2.2 Posts

```
struct posts{
      xmlChar* parentId;
      xmlChar* titulo;
      xmlChar* userId;
      xmlChar* tags;
      xmlChar* date;
      long scr;
      long ncom;
      int nresp;
      LInt resp;
};
```

A estrutura Posts é passada no TAD_community e foi criada com o intuito de guardar as informações relativas ao ficheiro Posts.xml, guardando numa Hash Table. A estrutura Posts possui 5 xmlChars correspondentes ao parent ID (sendo que esta componente diferencia se é uma pergunta ou resposta), título, user ID, as tags e a data. A estrutura também guarda um long scr de modo a ser guardado o score da resposta, um long ncom referente às respostas e que é o número de comentários, o int nresp é o número de respostas que respondem a essa pergunta e por fim uma lista ligada que guarda as suas respostas.

Métodos principais:

- `void initPost(Posts *p);`

Inicia a estrutura Posts, colocando-a a NULL.

- `LInt getResp(Posts p);`

Retorna a cópia lista ligada das respostas .

- `void setPostPer(Posts *p, xmlChar *t, xmlChar *tag, xmlChar *date);`

Preenche a estrutura posts alocando espaço onde necessário segundo o método de uma pergunta.

- `void setPostRes(Posts *p, xmlChar *s, long scr, long ncom);`

Preenche a estrutura posts alocando espaço onde necessário segundo o método de uma resposta.

- `void cleanPost(Posts *p);`

Liberta o espaço alocado a estrutura Posts.

Os restantes métodos são reservados para garantir ainda mais o encapsulamento (gets e sets).

2.2.3 Users

```
struct users{
      xmlChar* shortbio;
      xmlChar* displayname;
      xmlChar* reputation;
      int nposts;
      LDate posts;
};
```

A estrutura Users é passada no TAD_community e foi criada com o intuito de guardar as informações relativas ao ficheiro Users.xml guardando numa Hash Table. A estrutura possui 3 xmlChars correspondentes ao short bio, ao display name e à reputation, a estrutura possui também um int nposts onde é guardado o número de posts do User e por fim uma lista ligada com os posts do User ordenados temporalmente por cronologia inversa.

Métodos principais:

- `LDate getPosts_user(Users u);`

Retorna a cópia da lista dos posts do User ordenado por data.

- `void set_Post_User(Users *u, long i, xmlChar *d);`

Altera a estrutura Post associando um User, colocando a data na lista ligada LDate por cronologia inversa.

- `void set_User(Users *u, xmlChar* a, xmlChar* b, xmlChar* c);`

Carrega os novos parâmetros para a estrutura Users.

- `void cleanUser(Users *u);`

Liberta o espaço alocado a estrutura Users.

Novamente os restantes métodos ficam reservados para garantir ainda mais o encapsulamento (gets e sets).

2.2.4 Tags

```
struct tag{
      xmlChar* tagName;
      long id;
      struct tag *prox;
};
```

A estrutura Tag é passada no TAD_community e foi criada com a finalidade de guardar as informações relativas ao ficheiro Tags.xml guardando numa lista ligada. A estrutura possui um xmlChar que corresponde ao nome da tag, um long id e um apontador para a próxima casa sendo que esta é ordenada segundo o valor dos ids.

Métodos principais:

- `void clean_Tags(LTag l);`

Liberta o espaço alocado a estrutura LTag.

- `void insertOrd_tagused(LTag *resultado, LTag tag, int N);`

Insere ordenadamente uma tag na lista ligada LTag incrementando se já lá estiver.

- `void insert_Tag(LTag *t, char *aux);`

Insere uma tag numa lista ligada LTag.

- `void procura_insert_ID(LTag tags, LTag maisusadas, int N, LONG_list *r);`

Procura os elementos da LTag maisusadas na LTag tags, e devolve os ids ta LTag maisusadas na forma de uma LONG_list.

Os restantes métodos estão reservados para garantir ainda mais encapsulamento (gets e sets).

2.2.5 Datas

```
struct datelist{
       long id;
       xmlChar *date;
       struct datelist* prox;
};
```

No que respeita a estrutura datelist este passa pelo TAD_community cuja a criação tem o objetivo de facilitar a resposta as queries que contêm intervalos temporais. A estrutura possui um long id, um xmlChar *date que representa data e um apontador para a próxima casa.

Métodos principais:

- `void initDate(LDate *data);`

Inicia a estrutura LDate, colocando-a a NULL.

- `LDate* GetheadID(LDate *data, long *id);`

Guarda o ID da cabeça e devolve o resto da LDate.

- `LDate datecpy(LDate d);`

Cria uma cópia da LDate d,retornando a cópia da lista ligada original.

- `long* dateToArray(LDate beg, long size);`

Retorna o array dos IDs da estrutura LDate.

- `long* getIntervalo(LDate d, char* begin, char* end, long *size)`

Função cujo objetivo é dado um intervalo caraterizado por begin e end e um tamanho, retornar um array de ids da estrutura LDate nesse intervalo de tempo.

- `void freeLDate(LDate posts);`

Liberta o espaço alocado a estrutura LDate.

Os restantes métodos visam a completar o encapsulamento (gets e sets).

3 Queries

3.0.1 Query 1

```
STR_pair info_from_post(TAD_community com, long id){
      STR_pair resp = query1aux(com,id);
      return resp;
 }
```

Tendo em conta a estrutura Posts criada e todos os seus métodos associados, na função query1aux é testado se o post com o dado long id existe, em caso afirmativo, testa se é uma resposta, se for, guarda o id da pergunta a que responde, caso seja pergunta, vai à estrutura e carrega todas as informações dessa pergunta, título, o user id e, por fim, o display name.

3.0.2 Query 2

```
LONG_list top_most_active(TAD_community com, int N){
        LONG_list resp = query2aux(com,N);
        return resp;
}
```

Na query 2 foi utilizada uma estrutura que irá ser referida posteriormente, a LDLong, que é na sua natureza uma lista ligada. Esta irá guardar o id, e no campo common irá guardar o número de posts ordenadamente, sendo que serão feitas comparações e, por fim, inserções na lista, sendo retornada a lista que no fim será convertida para o tipo pretendido.

3.0.3 Query 3

```
LONG_pair total_posts(TAD_community com, Date begin, Date end){
        LONG_pair r = query3aux(com,begin,end);
        return r;
}
```

Como a query 3 envolve datas, será usada com maior foco a estrutura LDate, onde, usando esta juntamente com a função referida no sub capítulo 2.2.5 *getIntervalo* , irá extrair todos os ids dos posts feitos no intervalo dado, bastando verificar se o post é uma resposta e incrementando a variável associada ao número de respostas, caso contrário incrementa a varíavel do número de perguntas, retornando o número total de posts nesse período.

3.0.4 Query 4

```
LONG_list questions_with_tag(TAD_community com, char* tag, Date begin, Date end){
          LONG_list r = query4aux(com,tag,dateConvert(begin), dateConvert(end));
          return r;
}
```

A query 4 envolve tags pelo que será usada maioritariamente a estrutura LTag, e novamente, como involve um intervalo de tempo arbitrário, é usada a função *getIntervalo* para devolver todos os ids nesse intervalo dado. De seguida basta comparar as tags desse intervalo com a tag dada, verificando que se trata de uma pergunta, inserindo na cauda à medida que se avança e retornando a lista dos ids ordenada por cronologia inversa.

3.0.5 Query 5

```
USER get_user_info(TAD_community com, long id){
         USER r = query5aux(com,id);
         return r;
}
```

Nesta query, devido aos métodos criados na estrutura Users, basta verificar se o user existe na estrutura no dado id, e utilizar os métodos para retirar o shortbio e os 10 últimos posts por ordem cronológica inversa.

3.0.6 Query 6

```
LONG_list most_voted_answers(TAD_community com, int N, Date begin, Date end){
       LONG_list r = query6aux(com,N,begin,end);
       return r;
}
```

Como é dado um N arbitrário será usado predominantemente a estrutura LDLong. É também dado um intervalo de tempo e tal como nas outras queries, usa-se a função *getIntervalo* para devolver a lista de ids nesse intervalo de tempo. Apenas é necessário verificar de seguida se retrata uma pergunta,e caso seja o caso, é retirado o score das respostas dessa pergunta e é preenchida a estrutura LDLong respeitando a ordenação, sendo esta depois convertida no tipo pretendido com os ids das respostas.

3.0.7 Query 7

```
LONG_list most_answered_questions(TAD_community com, int N, Date begin, Date end){
        LONG_list r = query7aux(com,N,begin,end);
        return r;
```

Tal como na query 7, nesta também é dado um N e um intervalo de tempo devido a este facto. Será usada a estrutura LDLong juntamente com a função *getIntervalo* para retirar o array de ids dentro desse intervalo, de seguida usando a estrutura Posts, é verificado se é uma pergunta, retirando o número de respostas a essa pergunta, sendo devolvida a estrutura LDLong com o common e tem-se que o número de respostas, ordenada, por fim, por ordem decrescente a estrutura LDLong é convertida no tipo pretendido.

3.0.8 Query 8

```
LONG_list contains_word(TAD_community com, char* word, int N){
        LONG_list r = create_list(N);
        query8aux(com,word,N,&r);
        return r;
}
```

Na query 8 é usada a estrutura Users, onde é verificado se é uma pergunta e, usando uma adaptação de uma função da libraria C *strstr* , verifica-se se o Title contém a dada palavra e nesse caso é inserido o id na lista respeitando a cronologia inversa.

3.0.9 Query 9

```
LONG_list both_participated(TAD_community com, long id1, long id2, int N){
       LONG_list r = create_list(N);
       query9aux(com,id1,id2,&r);
       return r;
}
```

Neste caso como mistura utilizadores com posts são usadas fundamentalmente as estruturas Users e Posts, onde no ínicio retiramos o número de posts do utilizador 1 e do utilizador 2, e usando métodos já estabelecidos criou-se um array com os posts do primeiro e segundo utilizador. De seguida é feita a "interseção"com a utilização dos dois arrays, sendo que se irão percorrer de modo a comparar o conjunto de perguntas e respostas, testando se os posts existem e o tipo

(pergunta/resposta), e se ambos já responderem a alguma pergunta entre os 2 ou se responderem a alguma pergunta em comum.

3.0.10 Query 10

```
long better_answer(TAD_community com, long id){
     long r = GetBetterAnswer(com,id);
     return r;
}
```

Será usada a estrutura Posts neste caso, onde é verificado inicialmente se na estrutura o id existe e se é uma pergunta, neste caso são retiradas as respostas para uma LInt, onde de seguida são analisadas as respostas utilizando uma função de média ponderada e comparações para obter a melhor resposta.

3.0.11 Query 11

```
LONG_list most_used_best_rep(TAD_community com, int N, Date begin, Date end){
       LONG_list r = create_list(N);
       query11aux(com, N, dateConvert(begin), dateConvert(end), &r);
       return r;
}
```

Por fim, o problema foi abordado tendo sido começado por seleccionar os top N users do intervalo de tempo dado. Seguidamente, beneficiando da estrutura, nomeadamente do facto de cada user[index] conter a sua lista de posts, foi retirada dessa lista os posts contidos no intervalo de tempo dado como argumento. Destes posts são seleccionadas as perguntas e destas são discriminadas as tags, que serão ulteriormente carregadas numa lista ligada para serem depois ordenadas por ordem descrecente de ocorrência, a fim de dar uma resposta final à query. No entanto, na implementação deste algoritmo a maior dificuldade suscitou no processamento das tags que resultou no insucesso da realização desta tarefa.

4 Desempenho

Como o desempenho é um dos principais aspetos em ter em consideração na realização do trabalho, foi necessário ter em conta diversos aspetos para alcançar esse objetivo, nomeadamente:

- A estrutura principal usada, ou seja, Hash Tables, faz com que o tempo de acesso e resposta seja mais rápido, e como o número de dados a ser processados é grande, isso faz com que a estrutura escolhida seja ainda mais eficiente;
- Nas subestruturas que têm por base o conceito de lista ligada, são guardados os apontadores para a última casa, devido ao facto de, posteriormente, não ser necessário percorrer de novo a lista ligada de início, simplesmente pode-se recomeçar nesse ponto.
- A criação de uma estrutura auxiliar common ligada permite ajudar na resposta a várias queries, com foco especial para as queries que pedem um top N, sendo que o long common da estrutura guarda tudo o que seja essencial para ordenar pode ser tanto o score como o número de respostas, entre outros, com o objetivo de, assim, diminuir ainda mais o número de acessos à memória.

```
struct commonligada{
        long id;
        long common;
        struct commonligada* prox;
};
```

- Por fim, sempre que se faz mais de um acesso ao mesmo elemento, existe uma variável auxiliar permite poupar os acessos à memória central.

5 Conclusão

Ao longo da realização do projeto, o grupo foi confrontado com situações novas que puxaram pelo espírito de autonomia e também permitiu desenvolver um maior espírito auto crítico, passando pelos prós e contras de uma decisão.

Todas as estruturas usadas na realização do projeto, desde as Hashs até às listas ligadas requeriam um conhecimento prévio de conceitos abordados noutras unidades curriculares.

Foi imprescendível, também, aprender o funcionamento da libraria do XML, de efetuar o parse das dumps e guardar as informações úteis em estruturas.

O fato de haver necessidade de usar tipos abstratos foi também um grande stumbling block, sendo que, novamente, foi algo novo para o grupo e que obrigou por mais que uma vez a reestruturar o trabalho duma maneira completamente diferente do que o nomal face à experiência atual do grupo, como, por exemplo, criar uma cópia antes de a retornar, caso contrário o utilizador poderia ter acesso à mesma através do apontador.

Embora nem todas a queries terem sido respondidas com sucesso, todo este processo de realização do trabalho permitiu ao grupo consolidar tanto estas novas técnicas, como matéria lecionada em unidades curriculares anteriores. Permitindo também ganhar uma maior experiência na realização de trabalhos em equipa.