Projeto de Laboratórios de Informática 3
Grupo 46

Carlos Miguel Lopes Sá Barbosa - A82324
José Pedro Saraiva de Carvalho - A80424
Ricardo Rodrigues Martins - A78914

30 de Abril de 2018

Conteúdo

1 Introdução

Este trabalho, proposto pelos docentes da Unidade Curricular de Laboratórios de Informática 3, tem como propósito o processamento de vários ficheiros .xml fornecidos, proveninentes do site *StackOverflow*, de modo a recolher dados pertinentes sobre quem utiliza este site e como este é utilizado. Para atingir este objetivo, espera-se que os alunos obtenham um trabalho capaz de guardar devidamente os dados recolhidos numa estrutura à sua escolha, e que a partir dela encontrem soluções eficazes e eficientes para onze queries elaboradas pelos docentes.

2 Concepção da Solução

2.1 Estruturas de Dados

O guião fornecido pelos docentes da unidade curricular exige um *TAD COMMUNITY*, apontador para uma estrutura do tipo *TCD COMMUNITY*, que corresponde à estrutura que os alunos devem desenvolver com o intuito de armazenar dados e responder a queries. O TCD COMMUNITY presente neste trabalho foi desenvolvido através da junção das quatro seguintes GHashTables.

```
typedef struct TCD_community{
    GHashTable *HTU;
    GHashTable *HTQ;
    GHashTable *HTA;
    GHashTable *HTT;
}TCD_community;
```

A escolha de implementação de hashtables como estrutura de dados base para o este trabalho deveu-se à baixa complexidade na procura dos seus elementos. A existência das quatro hash tables referidas tem como principal objetivo separar users, questions, answers e tags, sendo que a última hashtable mencionada existe puramente para facilitar a resolução da query 11, bem como melhorar o seu desempenho.

- GHashTable *HTU : consiste numa HashTable cujas keys são os user id's, e cujos values são do tipo *AUser*, apontadores para structs do tipo *auser*, onde serão guardados os seguintes dados:

```
/* Estrutura que armazena as informações essenciais dum user */
struct auser{
    /* Nome do user */
    char* name;
    /* Bio do user */
    char* aboutme;
    /* Reputação do user */
    long reputation;
    /* ID do user */
    long id;
    /* Número de perguntas feitas pelo user */
    int numquestion;
    /* Apontadores das perguntas feitas pelo user */
    GPtrArray *questions;
    /* Número de respostas feitas pelo user */
    int numanswer;
    /* Apontadores das respostas feitas pelo user */
    GPtrArray *answer;
};
```

  - char* name: nome do user;
  - char* aboutme: biografia escrita pelo user;
  - long reputation: reputação do user;
  - long id: identificador do user;
  - int numquestion: nº de questions feitas pelo user;
  - GPtrArray *questions: apontadores para as questions dele;

- int numanswer: nº de answers feitas pelo user;
- GPtrArray *answer: apontadores para as answers dele.

- GHashTable *HTQ : consiste numa HashTable cujas keys são os post id's, e cujos values são do tipo *AQuestion*, apontadores para structs do tipo *question*, onde serão guardados os seguintes dados:

```
/* Estrutura que armazena as informações essenciais duma pergunta */
struct question{
    /* Id da questão */
    long id;
    /* Data de criação da questão */
    long creationdate;
    /* ID do user que fez a questão */
    long owneruserid;
    /* Tags associadas a essa questão */
    char**tags;
    /* Número de tags */
    int numtag;
    /* Título da questão */
    char* title;
    /* Contagem do número de respostas associadas à questão */
    int answercount;
    /* IDs das respostas associadas à questão */
    GArray* answer;
};
```

- long id : id da question;
- long creationdate: data da criação da question;
- long owneruserid: id do user que fez essa question;
- char**tags: tags usadas nessa question;
- char*title: título da question;
- int answercount: nº de answers feitas pelo user;
- GArray* answer: apontadores para as answers dele.

- GHashTable *HTA : consiste numa HashTable cujas keys são os post id's, e cujos values são do tipo *AAnswer*, apontadores para structs do tipo *answer*, onde serão guardados os seguintes dados:

```
/* Estrutura que armazena as informações essenciais duma resposta */
struct answer{
    /* ID da resposta */
    long id;
    /* Data de criação da resposta */
    long creationdate;
    /* ID da pergunta onde foi feita a resposta */
    long parentid;
    /* Score da resposta */
    int score;
    /* ID do user que fez esta resposta */
    long owneruserid;
    /* Número de comentários feitos na resposta */
    int commentcount;
};
```

- long id: id da answer;
- long creationdate: data de criação da answer;
- long parentid: id da question onde foi criada esta answer;
- int score: pontuação da answer;
- long owneruserid: id do user da answer;
- int commentcount: nº de comentários feitos nesta answer.

- GHashTable *HTT : consiste numa HashTable cujas keys são os nomes das tags e cujos values serão os id's dessas tags;

2.2 Parse dos Dados

O Parse dos Dados foi dividido em 3 etapas:

- fromUsers: Dada uma string "path"como argumento, devolve uma GHashTable com estruturas do tipo AUser, que contêm as informações relevantes de cada user. A função começa por verificar se o caminho dado corresponde de facto ao ficheiro Users.xml, e caso tal se verifique, trata então do parsing do ficheiro. Para este efeito, percorre cada nodo "row"do ficheiro e retira informação pertinente para variáveis temporárias recorrendo às funções xmlGetProp, strtol e mystrdup. Estas variáveis são enviadas para a função createUser, que devolve um AUser que é inserido na GHashTable a devolver pela fromUsers depois de percorrer todos os nodos.

- fromPosts: Recebe uma string "path"e três GHashTables, uma para os users, uma para as questions, e outra para as answers. Esta função começa por verificar se o path dado corresponde ao ficheiro Posts.xml, e caso o path esteja correto, faz o parsing do ficheiro encontrado. Para isto, à semelhança da fromUsers, são utilizadas funções e variáveis temporárias que são preenchidas e reutilizadas à medida que cada row é percorrida. Esta função tem a particularidade de distinguir questions e answers, e de acordo com o tipo de post, as informações que retira e o sítio onde as guarda variam. Caso seja uma question, as variáveis utilizadas são enviadas para a createQuestion, cujo output (estrutura do tipo AQuestion) é inserido na GHashTable das questions, e caso seja uma answer, então as variáveis são enviadas à createAnswer que devolve uma AAnswer que é inserida na GHashTable de answers. Tanto as answers como as questions geradas são associadas ao seu criador através da sua inserção num dos dois GPointerArray's que cada user possui, um para questions e outro para answers, que facilitam muito a otimização das queries. Depois de percorridas todas as rows, a função termina.

- fromTags: Dada uma string "path"como argumento e uma GHashTable como local para armazenamento dos ID's Tags, começa-se com a verificação da validação do caminho (o caminho tem de apontar para o ficheiro Tags.xml), e se de facto for válido, ocorre o parsing dos dados. Utilizando funções tais como o xmlGetProp, atoi e mystrdup, percorre-se cada nodo "row"e verifica-se em cada um o "TagName", que será a key para a GHashTable, e o ID, que será o value a assiciar à key na GHashTable. Este parse tem como objetivo organizar as tags de forma eficiente, o que será fulcral para a query 11, facilitando o acesso aos ID's de cada Tag.

2.3 Resolução das queries propostas

Após o parse dos dados, as seguintes queries foram resolvidas da seguinte forma:

2.3.1 Init

Função que aloca memória para um TCD community e devolve um apontador TAD community para o mesmo.

2.3.2 Load

Recebe um TAD community inicializado e uma string com a pasta dos ficheiros .xml. Esta função junta à string recebida o nome de cada ficheiro que será necessário ler, e preenche as diferentes GHashTables do TAD community a devolver com estruturas do tipo AUser, AAnswer, AQuestion e tag ID's com a ajuda das funções de parsing do getter. O bom funcionamento desta função é crucial, visto que todas as queries dependem da estrutura de dados retornada.

2.3.3 Query 1

Recebe como inputs o TAD COMMUNITY e um dado id dum post e o output é o nome do user (se for question é de quem fez essa question e se for uma answer é o id da pessoa que fez a question) e o título da question. Existem 3 tipos de outputs diferentes nesta query:

- O id fornecido ser duma question : basta verificar na GHashTable* HTQ esse id e devolve o nome do user e o título dessa question;
- O id fornecido ser duma answer : temos de ver na GHashTable* HTA o parent id, para verificar esse id na GHashTable* HTQ e assim devolver o nome e o título da question onde foi feita esta answer;
- O id fornecido nem é uma question ou answer : devolve-se vazio.

2.3.4 Query 2

Dado um TAD community e um número de users N, esta query devolve uma LONG list de user id's em ordem decrescente de número de posts. Para isto, é aplicado um foreach à hash table de users da estrutura de dados, ao qual é passado um array bidimensional e a função inserePar, que preenche ordenadamente o array com os id's e a contagem de posts dos users. A primeira posição deste array contém o seu número de linhas, o que permite à função inserePar saber quantas linhas deve utilizar para guardar os id's e evita a passagem de mais parâmetros. No final da query, os id's guardados no array utilizado anteriormente são passados para uma LONG list de tamanho N que é devolvida.

2.3.5 Query 3

Recebe como inputs o TAD COMMUNITY e o intervalo de tempo em que essas questions devem estar (data de início e do fim desse intervalo). São criados dois arrays de longs, ambos com as datas fornecidas pelos docentes alteradas para aumentar a facilidade de comparação destas mesmas. De seguida é utilizada em duas ocasiões distintas a função pertencente à biblioteca glib (g hash table foreach), que recebem as respetivas hashtables (a das answers e a das questions), as respetivas funções de comparação de datas (faux1 e faux2) e os respetivos arrays de longs referidos anteriormente. A última casa desses arrays é um contador que indica num caso quantas questions se encontram no intervalo de tempo pretendido e no outro caso quantas answers se encontram no intervalo de tempo pretendido. No final é retornado um LONG pair, cujo primeiro long é o número de questions naquele intervalo de tempo e cujo segundo long é o número de answers naquele intervalo de tempo.

2.3.6 Query 4

Recebe um TAD community, uma string com uma tag, e duas Dates, uma para o início do intervalo de tempo a testar e outra para o fim, e devolve uma LONG list com os id's das questions organizadas por cronologia inversa em que a tag dada foi encontrada. Para este efeito, todas as questions são passadas da hash table em que se encontram para uma glist que é organizada por cronologia inversa com o auxílio da função comp, e que posteriormente é percorrida por um ciclo while que lhe retira as questions que não se encontram dentro do intervalo de tempo fornecido, ou que não possuem a tag dada, que é comparada com todas as tags da question com a chamada da função searchT. Os id's das questions contidos na glist resultante são de seguida passados para a LONG list que é devolvida como resultado da query.

2.3.7 Query 5

Recebe como inputs o TAD COMMUNITY e o id dum user. Primeiro verifica-se se existe tal user na HTU e se não existir devolve uma bio a dizer *No User* e um post history com id a

zeros. Depois, retira-se o bio desse user e organiza-se todas as questions e answers feitas por esse user, recorrendo a GPtrArray's implementadas no user, onde têm apontadores para as questions e answers, inserindo-as em arrays organizadas por datas. Após isso, foi aplicado o algoritmo merge , para inserção dos ID's das questions e answers, pois são 2 array organizados e serão adicionados alguns dos ID's no post history dependendo das datas.

2.3.8 Query 6

Recebe como inputs o TAD COMMUNITY, o número de answers a ser devolvidas e o intervalo de tempo em que essas questions devem estar (data de início e do fim desse intervalo). Para esta query foi necessário organizar as answers pelo número de votos de cada answer (do maior para o menor) De seguida percorremos as answers e ver se faziam parte daquele intervalo, assim inserindo, por fim devolvendo a LONG list requerida.

2.3.9 Query 7

Recebe como inputs o TAD COMMUNITY, o número de answers a ser devolvidas e o intervalo de tempo em que essas questions devem estar (data de início e do fim desse intervalo). Para esta query as questions todas foram organizadas por datas e foram filtradas todas as questions desnecessáras para esta query (as que não pertenciam ao intervalo, não compremetendo a estrutura). Após tal procedimento, foi efetuada a contagem de todas as answers de todas as questions para verificar as que faziam parte daquele intervalo (pois tanto as questions, tanto as answers tinham de pertencer ao intervalo dado). De seguida foi feita a organização das questions dependendo do novo número de answers, assim finalizando com o retorno da LONG list.

2.3.10 Query 8

Recebe como inputs o TAD COMMUNITY, uma palavra (string), que se irá verificar se está ou não contida no título da question, e um número máximo de questions a que queremos aceder com essa palavra no título. Passamos os valores da hashtable de questions para uma glist (estrutura do glib). De seguida organiza-se conforme a data das questions. Após isto enquanto não se tiver chegado ao fim da lista e o número de questions não superar N, ir-se-à verificar se a palavra está contida no título da question, se estiver então acrescenta-se o Id à lista. No final esta lista é retornada.

2.3.11 Query 9

Recebe como inputs o TAD COMMUNITY, dois ids de dois users e um int que é o número máximo de posts. Se alguns dos users não existir então não poderá como é óbvio haver posts em comum, logo é retornado NULL. Caso ambos existam quatro apontadores são apontados respetivamente aos GPtrArrays, (estrutura da biblioteca glib), às questions de id1, às answers de id1, às questions de id2 e às answers de id2. Após isto são incluídos numa glist, (estrutura do glib), Ids de posts que pertençam a um dos seguintes casos:

- O user com Id id1 fez a question e o user com Id id2 responde;
- O user com Id id2 fez a question e o user com Id id1 responde;
- Ambos os users respondem.

A seguir verifica-se se existem Ids de questions repetidas na glist e se houver remove-se. Passa-se os valores da glist para a LONG list e depois retona-se a LONG list

2.3.12 Query 10

Recebe como inputs o TAD COMMUNITY e o id da question para ver a melhor answer. O procedimento para a resolução desta query foi analisar todas as answers associadas à question fornecida, fazer a média ponderada fornecida pelos docentes e analisar qual delas tinha o valor mais elevado delas todas, sendo essa a devolvida.

2.3.13 Query 11

Recebe um TAD community, um inteiro "N"correspondente ao número de elementos a devolver na lista, e duas Dates, uma para o início do intervalo de tempo e outra para o fim, e devolve uma LONG list com os N id's das tags mais utilizadas pelos N utilizadores com melhor reputação dentro do intervalo de tempo dado. A query começa por copiar para uma glist (new) todos os elementos da hash table de users e organizar essa glist por ordem decrescente de reputação dos users recorrendo à função comprep. Depois disto, copiam-se as questions dos N primeiros users que estejam dentro do intervalo de tempo dado para uma nova glist (questions). Uma vez que a glist de questions está devidamente preenchida, todas as tags aí presentes são copiadas para uma nova glist (tags), e o número de ocorrências de cada tag é contado e colocado juntamente com o id correspondente à tag por ordem decrescente de ocorrências num array bidimensional. Para terminar, os N id's das tags com maior ocorrência do array são colocados na LONG list que é retornada pela query.

2.3.14 Clean

Recebe um TAD community e liberta a memória ocupada por cada elemento que o constitui de modo a evitar memory leaks. Com a ajuda das funções freeElemHTU, freeElemHTQ e freeElemHTA, elimina tanto os campos de cada estrutura user, question e answer, como também elimina estes elementos das respetivas hash tables. No final, retorna a comunidade resultante.

2.4 Estratégias Para Melhoramento do Desempenho

Com o intuito de otimizar o processamento de dados fornecidos a partir do dump, o grupo usou, como foi referido anteriormente em vários pontos deste relatório, a biblioteca glib, em particular funções dos seguintes data types:

- GList;
- GHashTables;
- GPtrArrays;
- GArrays;

O grupo tomou ainda outras medidas para se assegurar da rapidez nomeadamente no âmbito da complexidade das queries, sendo que só numa query, (query 11), se pode verificar uma complexidade cujo Teta seja superior a N*N sendo N o tamanho de uma das hashtables.

2.4.1 Criação de um novo tipo de datas

Ao longo da elaboração deste trabalho, o grupo deparou-se com o facto de que, apesar das datas definidas pelos docentes da unidade curricular terem como unidade de tempo mais pequena o dia, existia uma quantidade exorbitante de posts efetuados com um intervalo de tempo inferior a um dia. Ora, como uma forma de aumentar a precisão da organização por data dos posts, o grupo decidiu definir uma nova maneira de comparar as datas, aproveitando que as datas de criação dos posts são feitas com um grau de exatidão que atinge o milissegundo. Foi para este caso criado um mecanismo de conversão das datas de criação presentes no ficheiro xml dos Posts em long ints. Para isto efetuou-se a seguinte soma: "Creationdate = msec(milissegundos) + seg(segundos) *

1000 + min(minutos) * 100000 + hour(hora) * 10000000 + day(dia) * 1000000000 + month(mês) * 100000000000 + year(ano) * 10000000000000"sendo que depois se comparavam estes long ints, (todas as questions e answers tinham um parâmetro creationdate onde se localizavam estes long ints), e que por conseguinte chegava-se à conclusão de que quanto maior fosse a creationdate do post mais recente este seria.

3 Conclusão

Em suma, este trabalho, elaborado no âmbito desta unidade curricular, no entender de quem o realizou demonstrou-se vital para compreender a utilização de bibliotecas externas e também para a aplicação de conteúdos adquiridos noutras unidades curriculares tais como Algoritmos e Complexidade, Programação Imperativa, entre outras. Serviu também de um teste à capacidade de contornar alguns problemas causados por trabalhar com tipos incompletos. Os alunos que compõem este grupo agradecem a oportunidade de terem podido testar as suas competências.