Trabalho prático de LI 3

Versão 1.2

Fevereiro, 2018

1 Introdução

O Stack Overflow[1] é, actualmente, uma das comunidades de perguntas e respostas mais utilizadas por desenvolvedores em todo o mundo. Nesta plataforma, qualquer utilizador pode colocar questões que serão depois respondidas por outros utilizadores. Estas respostas estão sujeitas a um sistema de votações que tende a favorecer as melhores respostas dado que as respostas são apresentadas por ordem decrescente do número de votos.

Cada utilizador acumula pontos de reputação que são ganhos sempre que uma das suas respostas é favoravelmente votada (voted up) e perdidos sempre que uma das suas respostas é votada desfavoravelmente (voted down).

Segundo informação oficial, a plataforma tem cerca de 8.4 milhões de utilizadores e já se colocaram cerca de 15 milhões de questões, às quais se responderam 24 milhões de vezes. A informação resultante da utilização da plataforma é extremamente útil e valiosa, não só devido ao conteúdo das perguntas e respostas mas também aos metadados originados a partir dos mesmos. Através deles é possível inferir, por exemplo, quais são os utilizadores mais ativos ou quais são os temas (identificados por tags) mais comuns. A análise destes metadados pode, no entanto, ser um processo bastante demorado e custoso devido ao grande volume de dados com os quais se tem de lidar e às operações necessárias para cruzar as diferentes informações disponíveis.

[1] https://www.stackoverflow.com

2 Objetivo

Este trabalho tem como objetivo o desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as várias informações utilizadas pelo Stack Overflow. Uma vez processada essa informação, pretende-se que seja possível executar um conjunto de interrogações específico (apresentado posteriormente) de forma eficiente. Esta aplicação deve ser desenvolvida obrigatoriamente em C e a escolha das estruturas de dados utilizadas deve ser devidamente justificada.

3 Requisitos

O sistema a desenvolver deve ser implementado em C e terá de suportar as funcionalidades apresentadas na Figura 1:

```c
#include "date.h"
#include "pair.h"
#include "list.h"
#include "user.h"

typedef struct TCD_community * TAD_community;

TAD_community init();

TAD_community load(TAD_community com, char* dump_path);

STR_pair info_from_post(TAD_community com, int id);

LONG_list top_most_active(TAD_community com, int N);

LONG_pair total_posts(TAD_community com, Date begin, Date end);

LONG_list questions_with_tag(TAD_community com, char* tag, Date begin, Date end);

USER get_user_info(TAD_community com, long id);

LONG_list most_voted_answers(TAD_community com, int N, Date begin, Date end);

LONG_list most_answered_questions(TAD_community com, int N, Date begin, Date end);

LONG_list contains_word(TAD_community com, char* word, int N);

LONG_list both_participated(TAD_community com, long id1, long id2, int N);

LONG_list better_answer(TAD_community com, int id);

LONG_LIST most_used_best_rep(TAD_community com, int N, Date begin, Date end);

TAD_community clean(TAD_community com)
```

Figura 1: Ficheiro interface.h

Esta interface será importada por um ficheiro main.c que deverá poder chamar qualquer uma das funções referidas na Figura 1. É esperado que a primeira função a ser chamada seja a função *init()* , que irá inicializar a estrutura TAD_istruct. Em seguida, a função *load()*[2] carregará a estrutura de dados definida pelo grupo, sobre a qual todas as interrogações produzirão os respectivos resultados.

4 Compilação

Deverá existir uma Makefile que permitirá gerar um executável "program" através do comando "make program". Este comando deverá compilar um ficheiro "main.c" que irá incluir o ficheiro "interface.h".

Respeitar esta nomenclatura é importante para garantir que o projeto é validado pelos scripts que irão executar nos repositórios GIT. O ficheiro "main.c" será substituído por uma versão alternativa para correr os testes de validação. Este ficheiro irá executar todas funções definidas no ficheiro "interface.h" pela ordem correta (init, load, interrogações, clean). As interrogações poderão ser feitas mais de uma vez com parâmetros diferentes.

O único requisito é que o ficheiro "main.c" possa incluir o ficheiro "interface.h" e que não necessite de incluir outros ficheiros do projeto ou bibliotecas externas para executar as funções corretamente.

O projeto deverá ser sempre compilado com a flag -Wall e -std=c11. Para utilizar bibliotecas externas (exemplo: libxml, glib, etc). A compilação deve sempre utilizar o comando pkg-config com os argumentos –cflags (para declarar o caminho para os ficheiros .h das bibliotecas) e –libs (para declarar o caminho para as bibliotecas).

Exemplo (muito simples) de uma Makefile para o projeto:

CC = gcc CFLAGS = -Wall -std=c11 -g ‘pkg-config –cflags libxml-2.0‘ ‘pkg-config –cflags glib-2.0‘ LIBS = ‘pkg-config –libs libxml-2.0‘ ‘pkg-config –libs glib-2.0‘

$(CC) $(CFLAGS) main.c -o program $(LIBS)

[2]Note que um dos parâmetros da função é o caminho para a diretoria onde os ficheiros a carregar estarão. Tais ficheiros seguirão a nomenclatura do backup exemplo (ver Secção 5).

5 Carregamento dos dados

O passo seguinte do programa será chamar a função *load(TAD_community com, char* dump_path)*. Que recebe como argumento extra o caminho onde os ficheiros estão armazenados.

Serão disponibilizados dois backups de diferentes comunidades e volume de posts (*i.e.,* android e askubuntu). Cada backup é composto por oito ficheiros .XML e podem ser baixados a partir do link *https://goo.gl/mXY9t3*. Este link também contém a documentação fornecida pela stack overflow que pode ser melhor interpretado recorrendo ao link:

https://ia800107.us.archive.org/27/items/stackexchange/readme.txt.

A leitura dos dados e extração de informação relevante de cada backup deve considerar todos os ficheiros (*i.e.*, Votes, Tags, Users, PostLinks, Posts, PostHistory, Comments e Badges). Vale notar que a seleção e estruturação desta informação deve ser justificada.

6 Interrogações

O ficheiro "*interface.h*" define as seguintes interrogações:

1. *STR_pair info_from_post(TAD_community com, int id);*
2. *LONG_list top_most_active(TAD_community com, int N);*
3. *LONG_pair total_posts(TAD_community com, Date begin, Date end);*
4. *LONG_list questions_with_tag(TAD_community com, char* tag, Date begin, Date end);*
5. *USER get_user_info(TAD_community com, long id);*
6. *LONG_list most_voted_answers(TAD_community com, int N, Date begin, Date end);*
7. *LONG_list most_answered_questions(TAD_community com, int N, Date begin, Date end);*
8. *LONG_list contains_word(TAD_community com, char* word, int N);*
9. *LONG_list both_participated(TAD_community com, long id1, long id2, int N);*

10. *long better_answer(TAD_community com, int id);*

11. *LONG_list most_used_best_rep(TAD_community com, int N, Date begin, Date end);*

Interrogação 1: Dado o identificador de um post, a função deve retornar o título do post e o nome (não o ID) de utilizador do autor[3]. Se o post for uma resposta, a função deverá retornar informações (título e utilizador) da pergunta correspondente;

Interrogação 2: Pretende obter o top N utilizadores com maior número de posts de sempre. Para isto, devem ser considerados tanto perguntas quanto respostas dadas pelo respectivo utilizador;

Interrogação 3: Dado um intervalo de tempo[4] arbitrário, obter o número total de posts (identificando perguntas e respostas separadamente) neste período;

Interrogação 4: Dado um intervalo de tempo arbitrário, retornar todas as perguntas contendo uma determinada *tag*. O retorno da função deverá ser uma lista com os IDs das perguntas ordenadas em cronologia inversa[5];

Interrogação 5: Dado um ID de utilizador, devolver a informação do seu perfil (short bio) e os IDs dos seus 10 últimos posts (perguntas ou respostas), ordenados por cronologia inversa;

Interrogação 6: Dado um intervalo de tempo arbitrário, devolver os IDs das N respostas com mais votos, em ordem decrescente do número de votos; O número de votos deverá ser obtido pela diferença entre *Up Votes* (UpMod[6]) e *Down Votes* (DownMod).

Interrogação 7: Dado um intervalo de tempo arbitrário, devolver as IDs das N perguntas com mais respostas, em ordem decrescente do número

[3]Caso o post não possua título ou nome de utilizador, o resultado correspondente deverá ser *NULL*.

[4]Para todas as interrogações que usam tempo como parâmetro de consulta, a granularidade deverá ser o dia.

[5]A pergunta mais recente no início da lista.

[6]Detalhes em https://ia800107.us.archive.org/27/items/stackexchange/readme.txt

de respostas;

Interrogação 8: Dado uma palavra, devolver uma lista com os IDs de N perguntas cujos títulos a contenham, ordenados por cronologia inversa;

Interrogação 9: Dados os IDs de dois utilizadores, devolver as últimas N perguntas (cronologia inversa) em que participaram dois utilizadores específicos. Note que os utilizadores podem ter participado via pergunta ou respostas;

Interrogação 10: Dado o ID de uma pergunta, obter a melhor resposta. Para isso, deverá usar a função de média ponderada abaixo:

$$(Scr \times 0.45) + (Rep \times 0.25) + (Vot \times 0.2) + (Comt \times 0.1)$$

onde,

- Scr - *score* da resposta;
- Rep - reputação do utilizador;
- Vot - número de votos recebidos pela resposta[7];
- $Comt$ - número de comentários recebidos pela resposta;

Interrogação 11: Dado um intervalo arbitrário de tempo, devolver os identificadores das N *tags* mais usadas pelos N utilizadores com melhor reputação. Em ordem decrescente do número de vezes em que a *tag* foi usada.

[7]Deverá ser usada a mesma abordagem da Interrogação 6