Universidade do Minho

Relatório – Laboratórios de Informática III

MIEI – 2º ANO – 2º SEMESTRE

UNIVERSIDADE DO MINHO

STACK OVERFLOW

GRUPO 9

André Costa	Carlos Gomes	Fábio Silva	Nuno Silva
A80824	A77185	A82331	A78156

3 de Maio de 2018

Conteúdo

Introdução

Este projeto foi nos 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 posts presentes em backups do Stack Overflow, realizados em diferentes datas, e extrair informação útil para esse período de tempo como, por exemplo, os top N utilizadores com maior número de posts de sempre, obter o número total de posts, entre outros.

Outros objetivos estão também associados a este, como a consolidação de conhecimentos adquiridos em unidades curriculares anteriores, dentro das quais se incluem Programação Imperativa, Algoritmos e Complexidade e Arquitetura de Computadores.
O principal desafio do projeto seria a programação em larga escala, uma vez que pelo nosso programa passam milhares de dados, aumentando assim a complexidade do trabalho. Para que a realização deste projeto fosse possível, foram-nos introduzidos novos princípios de programação, com especial relevo para a Modularidade e encapsulamento de dados.

Ficheiros

Os ficheiros que usamos para obter a informação e guardar o seu respetivo conteúdo foram Posts, Users e Tags.

Cada um desses ficheiros continha informação mais detalhada de diferentes partes que formam um post.

Esses ficheiros estavam escritos em XML, o que nos obrigou a realizar um parser para a verificação se cada ficheiro era valido.

TAD_community

A estrutura TAD_community é o modulo aonde vão ser armazenadas todas as informações retiradas dos ficheiros que possam vir a ser uteis para responder as interrogações. Como se trata de uma larga quantia de informação, necessitamos de a guardar em sub estruturas independentes de forma uma melhor organização de dados.

```
typedef struct TCD_community{
        head posts;
        usuario users;
        respostas comentarios;
}*TAD_community;
```

O tipo de estrutura para guardar e manipular a informação contida nos dump da Stack Overflow, foram as listas ligadas. Pois ao nosso ver, era o tipo de estrutura que melhor se adequava ao nosso conhecimento e por ser de fácil manipulação e integração.

Modularização funcional

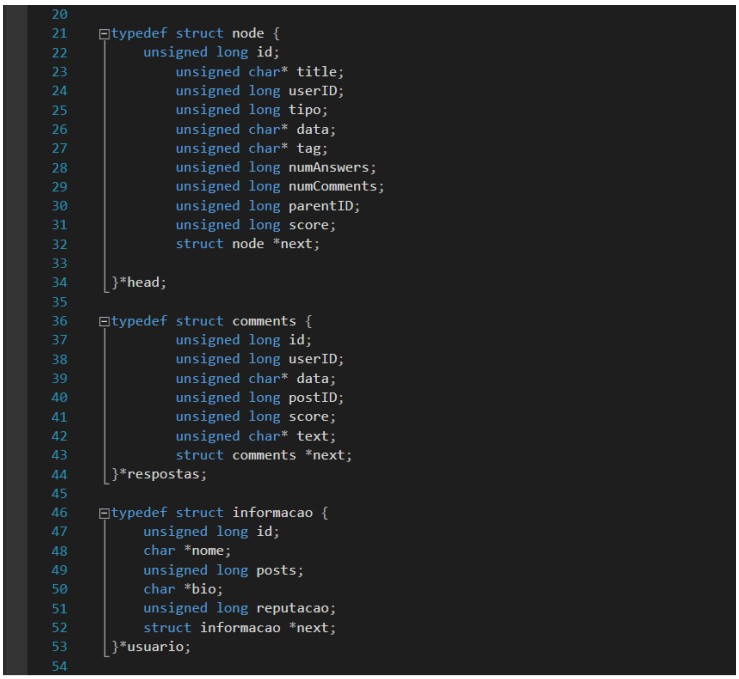
Para a concessão do trabalho as diferentes partes do código foram divididas em vários ficheiros.

Ficheiro esses que serão importados se a necessidade assim o exigir, foram criados os ficheiros funcoes.c (funções auxiliares), parser.c (parser que ira verificar a legibilidade de cada ficheiro XML), comentarios.c (funções que inserem toda a informação relativa do ficheiro comments.xml na sua estrutura correspondente), posts.c (funções que inserem toda a informação relativa do ficheiro posts.xml na sua estrutura correspondente) e usuarios.c (funções que inserem toda a informação relativa do ficheiro users.xml na sua estrutura correspondente).

Em relação à resolução das queries, a equipa optou por resolver na main.c, e ao mesmo tempo, fomos desenvolvendo uma interface, um menu de modo a facilitar a interação entre o programa e o user, que está localizado no menu.c. Ficheiro esse, menu.c, onde se encontra também as funções init, que inicializa todas as estruturas e a função load, que carrega os dados para as estruturas.

Abstração de dados

Tal como referido acima, a estrutura TAD_community é formada por três sub estruturas independentes que garantem uma melhor organização dos dados guardados e evita-se assim um conflito de informação.



```c

typedef struct node {
    unsigned long id;
        unsigned char* title;
        unsigned long userID;
        unsigned long tipo;
        unsigned char* data;
        unsigned char* tag;
        unsigned long numAnswers;
        unsigned long numComments;
        unsigned long parentID;
        unsigned long score;
        struct node *next;

}*head;

typedef struct comments {
        unsigned long id;
        unsigned long userID;
        unsigned char* data;
        unsigned long postID;
        unsigned long score;
        unsigned char* text;
        struct comments *next;
}*respostas;

typedef struct informacao {
    unsigned long id;
    char *nome;
    unsigned long posts;
    char *bio;
    unsigned long reputacao;
    struct informacao *next;
}*usuario;

```

Tal como verificado na imagem acima, cada estrutura é independente uma da outra. Cada estrutura recebe toda a informação de um ficheiro, por exemplo, a estrutura node recebe do ficheiro Posts, comments recebe de comments e informação recebe de Users.

Estratégias

Conceção de cada querie

- **Querie 1**: Dado o identificador de um post, a função deve retornar o titulo do post e o nome (não o ID) de utilizador do autor.

  Começou-se por verificar qual era o tipo de post que era pedido
  Caso fosse 1, percorríamos a estrutura até esse ID de post.
  Caso fosse 2, íamos ter que percorrer a estrutura até ao parentID desse post.
  E assim que chegássemos ao ID que se pretendia, tiramos a informação toda que era necessária.

- **Querie 2**: Pretende obter o top N utilizadores com maior número de posts de sempre.

  Começamos por inicializar um array com MAX_USERS (Variável que vai ser um limite que definimos) tudo a 0.
  E a partir daí, fomos percorrendo a estrutura toda e contando o nº de posts de cada utilizador.
  De notar, cada ID do user era a sua posição no array.

- **Querie 3**: Dado um intervalo de tempo4 arbitrário, obter o número total de posts (identificando perguntas e respostas separadamente) neste período.

  Das queries mais simples, ao nosso ver, onde bastava termos 2 variáveis, um que contasse os posts que fossem de tipo 1, e outra variável que contasse os posts que fossem do tipo 2. Após

definirmos essas variáveis, só foi preciso percorrer a estrutura e incrementar as variáveis de acordo com o tipo de post.

- **Querie 4**: Dado um intervalo de tempo arbitrário, retornar todas as perguntas contendo uma determinada *tag*. O retorno da função devera ser uma lista com os IDs das perguntas ordenadas em cronologia inversa.

  Após uma intensa pesquisa, encontramos uma solução para compararmos data com facilidade. Multiplicar o ano por 10000, o mês por 100 e o dia ficava normal. Também se definiu uma função chamada comparaTag em que vai receber a linha do tag e a tag que pretendemos procurar e vai devolver 0 ou 1.
  Com isto tudo, percorreu-se a estrutura e foi-se verificando as condições.

  https://www.portugal-a-programar.pt/forums/topic/13172-comparar-datas/

- **Querie 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.

  Começamos por definir um array dinâmico, pois não sabíamos dizer qual seria o tamanho final. Após isso, percorremos a estrutura de users até chegarmos ao ID do user que é pretendido e retirou-se a biografia. Após feito isso, percorremos a estrutura de posts e guardamos o ID de cada post do user até cumprir 10 ou chegasse ao fim.

- **Querie 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 devera ser obtido pela diferença entre *Up Votes* (UpMod) e *Down Votes* (DownMod).

  De novo, utilizou-se a mesma estratégia da data, após isso, inicializamos os arrays de votos e ids a 0. Depois de termos inicializado, percorreu-se a estrutura de posts e verificávamos primeiro se cumpria a condição. Se caso cumprisse, fazíamos um cicle for onde ia percorrer o array de votos e verificava se os votos desse post era maior do que estava guardado no array. E a medida que se ia fazendo isso, também se alterava o array dos ids.

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

  Mais uma vez, utilizou-se a tática das datas e da query 6, pois eram semelhantes.

- **Querie 8**: Dado uma palavra, devolver uma lista com os IDs de N perguntas cujos títulos a contenham, ordenados por cronologia inversa.

  Definiu-se uma função que chamava comparaTitulo, onde vai receber o título e a palavra que pretendemos verificar se existe no título. Se caso existir, devolve 1, senão 0. E fez-se isso enquanto percorria a cada post e verificava se o tipo de post era 2.

- **Querie 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.

  Das que mais deu desafios à equipa, começou-se por definir, de novo, um array dinâmico e um array que guarda o resultado de tamanho N.

  Começamos por percorrer a estrutura de posts e guardamos o ID/ParentID de acordo com a situação, de notar que primeiro verificava-se se o userID era o ID que pretendíamos. Após isso, tivemos que percorrer de novo a mesma estrutura, e fomos procurar pelo 2º ID e a medida que íamos encontrando, corríamos um ciclo for , para percorrermos o array do 1º userID.

  A medida que se ia achando, ia-se guardando no array resultado.

- **Querie 10**: Dado o ID de uma pergunta, obter a melhor resposta.
  Para isso, devera usar a função de media ponderada abaixo:
  (*Scr* × 0,45) + (*Rep* × 0,25) + (V*ot* × 0,2) + (*Comt* × 0,1) onde,
  Scr - *score* da resposta;
  Rep - reputação do utilizador;
  Vot - número de votos recebidos pela resposta7;
  Comt - número de comentários recebidos pela resposta;

  Percorremos a estrutura de posts e a estrutura de users, e para isso, tivemos que definir uma função chamada reputacaoUser, onde vai percorrer a estrutura e devolve o número da reputação desse user que fez o comentário. Depois de se definir essa função, corremos a estrutura e verificamos se o parentID correspondia ao ID da pergunta.
  Caso respondesse, aplicávamos a fórmula fornecida pelos professores.
  Feito isso, comparamos os valores para poder retornar o maior valor obtido.

- **Querie 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.

Otimização do desempenho

Que enquanto grupo, estamos conscientes, que o nosso projeto não deve ser dos melhores em termos de rapidez, uma vez que escolhemos por trabalhar com lista ligadas. Mas, no entanto, responde quase a todas queries pedidas com uma velocidade bastante aceitável.

Utilizamos táticas como array e breaks quando fosse necessário de modo a otimizar a velocidade, bem como o reaproveitamento de variáveis definidas para guardar outras informações quando fosse necessária.

Conclusão

Este foi o projeto que ate ao momento gerou um maior número de dificuldades a todos elementos do grupo. Ao longo da realização do mesmo fomos confrontados com situações novas que puxaram pelo nosso espírito de autonomia e também desenvolvemos uma maior capacidade avaliativa de quando tomamos uma decisão passando pelos pros e pelos contras da mesma.

Tivemos necessidade de aprender como funciona a libraria do XML de modo a conseguirmos efetuar o parse que verificava a legibilidade de cada ficheiro XML a ser utilizado.

Todo este processo de realização do projeto serviu para o grupo consolidar matéria lecionada em outras unidades curriculares e ganhar uma maior experiência na realização de trabalhos em equipa.