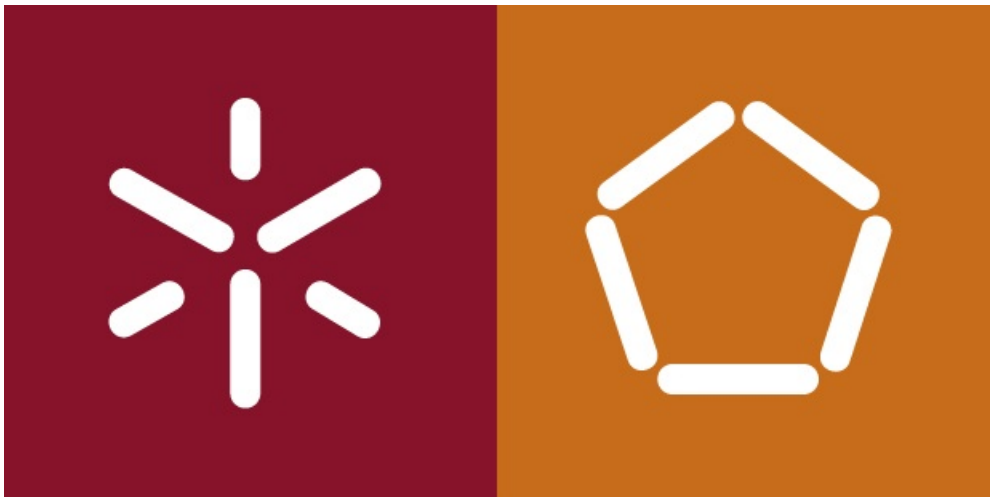
Universidade do Minho

Mestrado Integrado em Engenharia Informática



Relatório - Laboratórios de Informática III

Grupo 19

Trabalho realizado por:	*Número*
Adriana Gonçalves	A75119
Marco Gonçalves	A75480
Ricardo Canela	A74568

Conteúdo

1 Introdução

Este projeto foi-nos solicitado pelos docentes da unidade curricular Laboratórios de Informática III e, esta fase, tem como principal objetivo a realização de um projeto em C sobre o Stack Overflow. Através do processamento de ficheiros XML do Stack Overflow, o programa terá de ler e guardar todas as informações necessárias para responder ás interrogações propostas. O Stack Overflow é, atualmente, uma das comunidades de perguntas e respostas mais utilizadas em todo o mundo e, devido à grande utilização desta plataforma e de forma a obter informações como o top N utilizadores com maior número de posts de sempre, o número total de posts num determinado intervalo de tempo, o título de um post e o nome do respetivo autor (dado o id do post) ou a informação do seu perfil pessoal (short bio), como iremos ver mais à frente, é necessário o desenvolvimento de um código legivel, bem estruturado e a utilização de estruturas para fluxo dos respetivos dados. Neste relatório, iremos ainda apresentar e fundamentar as opções de desenho e desempenho do trabalho realizado bem como eventuais limitações/superações dos objetivos propostos.

2 Descrição do enunciado do projeto

O projeto tem como principal objetivo responder a um total de onze interrogações sobre a plataforma de perguntas e respostas Stack Overflow. E para este efeito deve ser desenvolvido um programa em C bem estruturado e a executar no menor tempo possivel de maneira a obter-se uma melhor qualidade de resultados. As informações relativas aos posts, users e votes desta plataforma num determinado instante podem ser acedidos através de ficheiros XML que fornecem toda essa informação. Através da análise das queries e do enunciado do projeto percebemos que apenas temos a necessidade de definir os dados das tags, users, quests, answers e days, sendo que estes possuem caracteristicas comuns mas tambem um conjunto de caracteristicas específicas. Utilizamos ainda o diferentes estruturas principais para armazenar todo o conteudo do xml e outras estruturas auxiliares para para o fluxo de dados de modo a otimizar o desempenho do programa.

3 Tipo concreto de dados - TCD

O tipo concreto de dados (TCD) contém todas as estruturas principais, que foram definidas para a realização do projeto, nomeadamente uma HashTable para as tags, outra para os users, para as quests e para as answers; e uma Tree para os days, sendo que esta última terá associado duas HashTables, uma para as quests e outra para as answers.

4 Tipo abstrato de dados - TAD

O tipo abstrato de dados (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados sem referência a detalhes da implementação. A especificação das TADs no nosso programa é possível devido aos ficheiros .h através da diretiva #include dos .h onde se define apenas os cabeçalhos das operações e nos .c implementa as operações usando as mesmas assinaturas definidas no .h. As TADs conferem ainda ao programa um encapsulamento de dados permitindo manter os dados de cada classe com funções bem delimitadas e a criação de módulos onde cada classe cumpra o seu objetivo, tendo total controle sobre as operações correspondentes.

5 Modularização funcional

Visto que este projeto trabalha com uma enorme quantidade de dados, o mais usual é dividir o código em diversos ficheiros. Desta forma, o nosso projeto é dividido em ficheiros com extensão .h (headers) e ficheiros .c. Os ficheiros com extensão .h são utilizados para especificar funções, constantes e tipos de dados necessários na utilização dos ficheiros .c. Os ficheiros .c contêm tudo o que é interno a esse módulo e usam os ficheiros .h necessários para a definição das funções do seu ficheiro. A modulalaridade do código permite assim que a estrutura do programa fique mais clara (agrupando funções e variáveis relacionadas num mesmo ficheiro, conferindo encapsulamento ao código), permite que se compile cada módulo individualmente (sendo mais fácil a correção de erros) e a manutenção mais fácil (reutilização de funções). Assim sendo, como verificamos que nos ficheiros .h é onde estão definidas as funções a serem usadas nos ficheiros .c, no nosso trabalhos esses ficheiros estão definidos da seguinte maneira:

- **tag.c:** este ficheiro .c usa as funções definidas no ficheiro tag.h onde está definida a função necessária para inicializar esta estrutura para inicializar esta estrutura, as funções responsáveis por onde está, as funções responsáveis por recolher os dados necessários para a mesma e a função responsável por imprimir os resultados desse módulo.
- **users.c:** este ficheiro .c usa as funções definidas nos ficheiros common.h e users.h. No ficheiro users.h está definida a função responsável por inicializar a estrutura dos users, as funções necessárias para recolher os dados necessários para a mesma e a função responsável por imprimir

os resultados desse módulo. No ficheiro common.h está definida a função mystrdup (mystrdup (const char *s)). Esta é usada para proteger o nosso código através do encapsulamento de dados uma vez que esta função recebe um char* (string), dá malloc do size da string e copia a string para dentro da memória alocada, devolvendo o apontador para onde está guardada a string na memória.

- **quest.c:** este ficheiro .c usa as funções definidas no ficheiro quest.h onde está definida a função necessária para inicializar esta estrutura, as funções responsáveis por recolher os dados necessários para a mesma e a função responsável por imprimir os resultados desse módulo.
- **answer.c:** este ficheiro .c usa as funções definidas no ficheiro answer.h onde está definida a função necessária para inicializar esta estrutura, as funções responsáveis por recolher os dados necessários para a mesma e a função responsável por imprimir os resultados desse módulo.
- **day.c:** este ficheiro .c usa as funções definidas no ficheiro day.h onde está definida a função responsável por recolher os dados necessários para a mesma e a função responsável por imprimir os resultados desse módulo.
- **tcd.c:** este ficheiro .c usa as funções definidas nos ficheiros tcd.h, tag.h, users.h, quest.h, answer.h e day.h. Estes ficheiros é onde estão definidas as estruturas base do trabalho.

6 Estruturas

Neste capítulo iremos abordar as estruturas que utilizamos no desenvolvimento do nosso projeto, fundamentando o porque da escolha dessas estruturas bem como as vantagens das mesmas na funcionalidade e eficácia na resolução das queries do trabalho proposto.

6.1 Tags

Para as Tags, optamos por criar uma Hashtable visto que esta possui a vantagem de associar keys a valores. O seu objetivo é, a partir de uma key, fazer uma pesquisa rápida e obter o valor desejado. Assim sendo, a Hashtable das Tags tem como par key,value o nome da tag e o módulo de dados da tag respetivamente. O value contém o id da tag, o nome de uma tag e um inteiro correspondente ao número de vezes que uma tag é usada num intervalo de tempo (definido pela query 11).

```
Tag create_tag(long id,char* s,int n);
```

- **Tag create_tag:** é a função responsável por criar a estrutura Tag.

6.2 Users

Para a estrutura dos Users, optamos pela criação de uma Hashtable pelas características que esta possui, já mencionadas na secção anterior (Tags). Nesta estrutura, cada nodo (User) irá guardar uma lista com o Id do user, o DisplayName, a short bio (AboutMe), a reputação (Reputation), o número de posts em que um user participou (nr_posts) e uma lista das quests e das answers que um user fez.

```
User init_user(long id, char* dn, char* am, long r);
```

- **User init_user:** é a função responsável por inicializar a estrutura dos Users.

6.3 Quests

Para a estrutura dos Quest, optamos pela criação de uma Hashtable pelas características que esta possui já mencionadas anteriormente. Através desta estrutura, cada módulo quest irá guardar uma lista com o Id da quest, a data de criação de uma pergunta (creationDate), o score da quest, o ownerUserId de um user que elaborou uma determinada quest, o título, uma tag, o número de anwsers, comentários e quests com mais votos e uma lista ligada das answer que uma quest tem.

```
Quest init_quest(long id, Date cd, int s,long ouid, char* ti,
char* ta, int ac, int cc, int fc);
```

- **Quest init_quest:** é a função responsável por inicializar a estrutura das Quest.

6.4 Answers

Para a estrutura das Answers, optamos pela criação de uma Hashtable pelas características que esta possui já mencionadas anteriormente. Através desta estrutura, cada answer irá guardar uma lista com o Id da answer, o parentId que indica o id da quest à qual uma answer responde, a data de criação de uma resposta (creationDate), o ownerUserId de um user que elaborou uma determinada answer, o número de comentários que uma quest teve e o número de answer com mais votos, e a average (indica a qualidade da answer de uma quest entre duas datas).

```
Answer init_answer(long id,int pid,Date cd,int s,long ouid,
int cc,int fc);
```

- **Answer init_answer:** é a função responsável por inicializar a estrutura das Answer.

6.5 Days

Para a estrutura dos Day, optamos por criar uma Tree visto que é mais útil para responder a queries relacionadas com intervalos de tempo, sendo que a pesquisa em árvores será executada de forma mais rápida e eficaz. Esta Tree contém uma data (Date day), o número de quest e answer que ocorreram num dia e ainda tem associado duas HashTables, uma para as quest e outra para as answer. Optamos por cada nodo da árvore representar um dia onde todas as participações de um dia estão lá guardadas separadamente em answers e quests.

```
Day init_day(Date day);
```

- **Day init_day:** é a função responsável por inicializar a estrutura dos Day.

7 Parser

Sendo o parser uma componente essencial no bom funcionamento do projecto, é necessário decidir a melhor solução possível. Assim, na existência de vários tipos de parser, optamos pelo DOMParser (Document Object Model) para implementação do projeto, pois é bom para ser utilizado em grandes documentos não consumindo tanta memória como no caso de outros tipos de parsers. Neste caso, o parser executará documentos XML e gerará um DOM para a aplicação usar. Ao longo do desenvolvimento do projeto apercebemo-nos que apenas temos a necessidade de parsar três tipos de dados:

as Tags, os Users e os Posts. Assim, teremos três funções auxiliares com o objetivo de pre-processar os ficheiros contendo os dados anteriormente referidos. A função processTag irá criar uma tag com a informação respetiva e insere na hashtable das tags, esta é chamada pela streamTags que itera ao longo do ficheiro com as rows que contem a informação do elemento tag. Para os Users, o processo é análogo ao das Tags. Para os Post, a processPost terá a função de criar um novo post (com o id, posttypeid, parentid, creationdate, score, owneruserid, tittle, tags, answercount e commentcount) e inseri-lo em todas as estruturas que lhe dizem respeito. Começa por verificar se o post é do tipo pergunta ou resposta, no primeiro caso é inicailizado uma quest com os valores recolhidos e é inserida ma hashtable das quest e numa hashtable de um dos dias (caso o dia com a data do post nao exista é criado um novo dia com a mesma), para além de incrementar a variavel nr_post do user que o criou e de o adicionar à lista de respostas desse mesmo user. Caso o post seja uma resposta é tomado o mesmo procedimento para o caso das perguntas.

8 Queries

Após todas as estruturas estarem bem definidas e os dados devidamente armazenados, estão assim reunidas as condições necessárias para começarmos a responder ás onze queries referidas no enunciado.

8.1 Info-From-Post

Para a resolução desta interrogação, teremos inicialmente de saber se o id do post fornecido é relativo às quest ou às answer. Caso se trate de uma quest, basta aceder à Hash das quests e retirar o título do post correspondente e o ownerUserId do user que elaborou a quest. Seguidamente, acedemos à estrutura dos users e, através do ownerUserId do post, obtemos o DisplayName do user que criou a quest. Caso o post seja relativo a uma answer, teremos de obter o parentId do post sendo que este retornará o id da quest à qual esta answer responde. Depois disto, a forma de obter o título do post e o nome do utilizador autor do post retorna-se de forma análoga ao caso das quests.

8.2 Top-Most-Active

Para a resolução desta interrogação é necessário receber um inteiro N correspondente ao número de utilizadores pretendidos com mais posts de sempre (tanto perguntas como respostas). Para isso, basta iterar a estrutura dos Users e utilizar a variável nr_post para criar uma lista ligada ordenada por este fator. Esta é guardada na estrutura de dados principal, pois quando a execução da query for efetuada mais do que uma vez será somente necessário truncar a lista pelos N elementos necessários.

8.3 Total-Posts

Para a resolução desta interrogação é necessário receber duas datas arbitrárias (Date begin e Date end) e esta irá retornar o total de posts ocorridos nesse intervalo de datas, indentificando separadamente as perguntas das respostas. Para isso, teremos de aceder à nossa GTree e selecionar os Day's entre as datas e efetuar o somatório das variáveis n_quest e n_answer separadamente, que são o número de perguntas e respostas que ocorreram num dia.

8.4 Questions-With-Tag

Para a resolução desta interrogação é necessário receber duas datas (Date begin e Date end) e uma tag arbitrária de forma a retornar uma lista com os id's de todas as perguntas que contêm essa mesma tag. Para isso, teremos de aceder à nossa GTree e selecionar os Day's entre as datas fornecidas, posteriormente aceder à GHashTable das quest de forma a obtermos as quests que contêm essa tag e inseri-la numa lista ligada ordenada pela data. No final, a função returnará uma lista com todos os id's das N perguntas com essa tag ordenadas em cronologia inversa.

8.5 Get-User-Info

Para a resolução desta interrogação, com o id do user fornecido e de forma a obter a short bio (perfil) e os Id's dos seus últimos 10 posts, é apenas necessário aceder à estrutura User e retornar a bio e, posteriormente ordenar ambas as listas (quests e answers) pela data, e retorna os 10 mais recentes posts das duas listas.

8.6 Most-Voted-Answers

https://www.sharelatex.com/8639991335ttzgncwtmcwc Para a resolução desta interrogação é necessário receber duas datas (Date begin e Date end), e na estrutura dos Day's, aceder a hash das answers (GHashTable* hash_answer) de cada dia desse intervalo de forma a obtermos o score de cada resposta e efetuar a criação de uma lista ligada de answers ordenada por esse fator. No final a função retorna os Id's das N respostas com mais votos.

8.7 Most-Answered-Questions

Para a resolução desta interrogação é necessário receber duas datas (Date begin e Date end), aceder a hash das quest de cada dia desse intervalo de forma a obtermos o número de answer's (answer_c) criando uma lista ligada ordenada por esse fator.

8.8 Contains-Word

Para a resolução desta interrogação, dada uma palavra, será necessário aceder à estrutura das Quest e ver em cada título das quest se essa palavra está contida. No final, retorna uma lista com os id's das N perguntas cujos títulos a contenham, ordenados por cronologia inversa.

8.9 Both-Participated

Para a resolução desta interrogação, dados dois id's de dois users e um inteiro N, é necessário aceder à hash dos Users, e ao id de cada user procurar os posts em que cada um desses utilizadores participaram (GSList* quests e GSList* answers). Depois de carregar as hash com longs (id's das quest) haverá uma remoção dos id's dos post ficando apenas aqueles em que ambos participaram, a hash dos id's dos posts será transformada numa lista sendo esta posteriormente ordenada. No final, retorna a lista de id's das N perguntas em que ambos os user participaram.

8.10 Better-Answer

Para a resolução desta interrogação, dado o id de uma pergunta, será necessário aceder à estrutura das Quest e recolher a GSList* answerList (lista das answer de uma determinada quest) e verificar de qual das respostas é a melhor. Para isso basta verificar a average de todas as respostas a essa pergunta e retornar a que tiver melhor média.

8.11 Most-Used-Best-Rep

Para a resolução desta interrogação é necessário receber duas datas (Date begin e Date end) e um N número de utilizadores. Assim, é necessário organizar os users por ordem de maior reputação e guardá-los numa lista ligada de forma ordenada e ainda aceder à lista de quests que cada user fez. Acedemos à lista de quest que cada user efetuou, verificamos se essas questões estão entre as datas. Caso estejam entre essas duas datas, incrementamos a variável n_used na hashtable das tags de cada uma das tags da quets. Criamos uma lista ligada de tags ordenada pela variável n_used.

9 Conclusão

Com a conclusão deste projecto foi possível perceber os diferentes aspetos no funcionamento de programação a larga escala, uma vez que passaram pelo nosso programa uma grande quantidade de dados, aumentando assim a complexidade do trabalho. Assim, ao longo da realização deste projeto, fomos confrontados com algumas dificulades nomeadamente no caminho a seguir para responder de forma mais eficiente ao pretendido, o tipo de estruturas a usar, o parser mais adequado, tendo sido necessário um trabalho de investigação mais aprofundado de forma a dar resposta às nossas dúvidas. Contudo, este projeto permitiu que o nivel de conhecimentos sobre esta linguagem aumentasse, permitiu-nos uma maior proximidade as estruturas de dados bem como a experiência de trabalhar num projeto onde trabalhamos com uma enorme quantidade de dados.