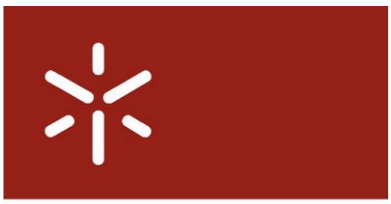


Universidade do Minho

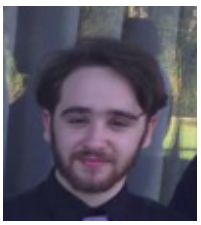
Laboratórios de Informática 3

MIEI - 2º ano - 2º semestre
Universidade do Minho

Trabalho Prático

João Vieira
A76516

Rui Vieira
A74658



António Gomes
AE3654

5 de Maio de 2018

Conteúdo

1. *Introdução*

No âmbito da unidade curricular de Laboratórios de Informática III, foi nos proposto o desenvolvimento de um sistema capaz de lidar com enorme volume de dados relacionados com uma das maiores plataformas de troca de conhecimento, stackoverflow. Este projeto considera-se um grande desafio para nós pelo facto de passarmos a programar em grande escala, uma vez, que o projeto exige lidar com grandes volumes de dados e com uma complexidade algoritmica e estrutural mais elevada. Nesse sentido, o desenvolvimento deste programa será realizado a partir dos princípios da modularidade (divisão do código fonte em unidades separadas coerentes), do encapsulamento (garantia de proteção e acessos controlados aos dados), da conceção de código reutilizável e escolha otimizada das estruturas de dados.

2. *Abordagem do problema e Descrição das estruturas criadas*

2.1 Problema

O projeto tem como principal objetivo responder a onze interrogações sobre a plataforma de troca de conhecimento, Stack Overflow. De forma a responder às interrogações deve ser desenvolvido um programa em C bem estruturado e o mais otimizado possível utilizando conhecimentos adquiridos diversas unidades curriculares frequentadas até à presente data. A informação é nos dada em vários ficheiros, XML que guardam diferentes dados, depois de analisar com bastante cuidado enunciado e ficheiro notamos que só necessitávamos de fazer parse do Posts.xml, Users.xml e Tags.xml. Parser esse que insere informação em estruturas de dados e respetivas sub estruturas.

2.2 Tipos Concretos de dados

2.2.1 TCD community

O tipo concreto de dados, TCD,é constituído pelas principais estruturas dados definidas para realização do projeto. Para estruturas principais optamos pela utilização de três HashTable referentes a Users, Respostas e Perguntas.

Para Users, optamos pela utilização de uma HashTable usando o ID como 'key', pois permite mapeamento direto, e uma sub-estrutura User como 'valeu', implementando a mesma utilizando a glib.

Para Respostas, optamos pela utilização de uma HashTable usando o ID como 'key', pois permite mapeamento direto, e uma sub-estrutura Resposta como 'valeu', implementando a mesma utilizando a glib.

Para Perguntas, optamos pela utilização de uma HashTable usando ID como 'key', pois permite mapeamento direto, e uma sub- estrutura Pergunta como 'valeu' que contem um AVL de respostas ordenada por data, utilizando a glib em ambas.

Optamos pela utilização da glib por conter estruturas já definidas e otimizadas, não necessitando de reinventar a roda, mas necessitando ainda assim de muita pesquisa para a sua utilização pois só podemos usar funções já definidas.

3. *Modularidade e encapsulamento*

3.1 Modularidade funcional no código

A modularidade de um código consiste na divisão em partes distintas do projeto de forma a que a sua leitura seja mais legível, a sua manutenção mais fácil e ainda que este tenha um melhor desempenho. Desta forma, o nosso projeto é dividido em ficheiros com extensão .h e ficheiros .c. Os ficheiros com extensão .h são utilizados para especificar funções, constantes e tipos de dados necessários na utilização dos ficheiros .c. Estes ficheiros têm ainda a função de definir a interface de um módulo visto que são responsáveis por realizar tarefas específicas e consistem em retornar um valor associado ao seu nome.

3.2 Encapsulamento

De forma a manter o encapsulamento, definimos funções para leitura, escrita, manipulação, comparação e respetivos auxiliares de cada estrutura de suporte nas respetivas livrarias garantindo assim o encapsulamento e preservação do conteúdo das mesmas.

3.3 Abstração de dados

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. Esta abstração permitiu uma maior limpeza e legibilidade do código desenvolvido.

4. *Estruturas e sub-estruturas*

4.1 Conjugação de estruturas e sub-estruturas

Tal como referido anteriormente, no TCD, implementamos três HasTable que utilizam sub-estruturas definidas por nós que explicaremos de seguida com mais detalhe.

4.2 Sub-estruturas

4.2.1 DateTime

Muitas das interrogações necessitam da comparação de datas mais extensiva de forma a não perder informação quando a key é uma data. De forma a que tais perdas não se verifiquem criamos uma estrutura, DateTime que é em tudo semelhante à data fornecida pela equipa docente mas mais completa pois vai até aos segundo a passo que dada só chega aos dias, como podemos ver de seguida no seu conteúdo:

- **year**- int representando o ano da data.
- **month**- int representando o mês da data.
- **day**- int representando o dia da data.
- **hour**- int representando a hora da data.
- **minute**- int representando o minuto da data.
- **second**- int representando o segundo da data.

4.2.2 Users

Para representação dos dados referentes a um utilizador criamos uma estrutura users que se insere dentro da HashTable de users (usando o seu id como key) tendo como conteúdo :

- **Id**- long usado como identificador do utilizador.
- **Bio**- char* (string) de descrição do utilizador.
- **Rep**- int representado a reputação do utilizador.
- **Name**- char* (string) nome do utilizador.
- **Views**- int representando o numero de views do utilizador.
- **VoteDif**- int representando a diferença de votos (positivos-negativos) do utilizador.
- **NrPosts**- int representando o numero de posts efetuado pelo utilizador. É de notar que este numero é calculado no parser para todos os utilizadores.

4.2.3 Resposta

Para representação de dados referentes a uma resposta criamos uma estrutura Resposta, que se insere dentro da HashTable de respostas (usando o seu id como key) e da AVL de respostas referente a uma Pergunta usando a sua data como key na AVL. A Respostas tem como conteúdo:

- **Id**- long usado como identificador da resposta.
- **ParentId**- long usado como identificador da pergunta à qual pertence esta resposta.
- **CreationDate**- DateTime representando a data de criação da resposta.
- **score**- int representando o score da resposta.
- **OwnerUserId**- long usado como identificador do utilizador a que a resposta pertence.
- **CommentCount**- int representando o numero de comentários à resposta.
- **Rate**- float representando o rate da resposta. É de notar que este rate é calculado no parser para todas as respostas.

4.2.4 Pergunta

Para representação de dados referentes a uma pergunta criamos uma estrutura Pergunta, que se insere dentro da HashTable de perguntas (usando o seu id como key) tendo como conteudo:

- **Id**- long usado como identificador da pergunta.
- **CreationDate**- DateTime representando a data de criação da pergunta.
- **Score**- int representando o score da resposta.
- **Title**- char* (string) representando Titulo da pergunta.
- **Tags**- char** (array de strings) representando as tags usadas na pergunta.
- **nTags**- int numero de tags usadas na resposta.
- **CommentCount**- int representando o numero de comentários à pergunta.
- **Resp**- GTree* de respostas à pergunta. É de notar que árvore usa DateTime como key facilitando assim as interrogações entre intrevalos de tempo.

5. *Interrogações*

Após a devida analise/reflexão do problema e armazenamento dos dados, estão reunidas as condições necessárias para responder às interrogações referidas no enunciado.

5.1 Interrogação 1 - info from post

Esta interrogação procura a partir do id de um post, retornando o titulo do post e nome do seu autor.

5.1.1 Estratégia

Devido ao uso de HashTables a resposta a esta interrogação é quase trivial, bastando procurar direta na HashTable de perguntas pelo id dado, caso não exista fazemos o mesma na HashTable de respostas encontrando assim a sub estrutura do post da qual usamos o OwnerUserId para mais uma vez procura na HashTable de users e saber o seu nome. Caso o post seja uma respostas obriga ainda a uma procura da pergunta a partir do ParentID indo buscar o titulo da respetiva pergunta.

5.2 Interrogação 2 - top most active

Esta interrogação encontra os top N com o maior numero de posts de sempre.

5.2.1 Estratégia

De forma a responder a esta interrogação será sempre necessário percorrer toda a HashTable de users ficando com os que tem o valor mais alto de numero de posts calculado por nós no parser.

5.3 Interrogação 3- total posts

O número total de posts, consistindo em total de perguntas e total de respostas, pode ser respondido com uma travessia sobre a hash table de perguntas e respostas respectivamente incrementando um contador para perguntas e um para respostas.// As travessias sobre as g hash table podem ser efectuadas utilizando duas diferentes ferramentas disponibilizadas pela biblioteca glib.// 1 -¿ Utilizando a função foreach pode-se aplicar uma função de retorno boleano gbool a todos os elementos retornando true caso seja para parar e false caso seja para continuar para o próximo elemento. 2 -¿ Utilizando interators podemos iterar sobre as estruturas directamente com um ciclo while à semelhança do que é possível em linguagens orientadas a objectos como java. Optamos por utilizar iterators na maior parte das querys à exceção da query 10 que exige travessias sobre arvores as quais requerem foreach.

5.4 Query 5 - get user info

É pedido a informação relativa a um certo user e a lista dos seus ultimos posts sendo perguntas ou respostas.

5.4.1 Estratégias

A informação relativa a qualquer user de um certo id pode ser obtida atravéz de mapeamento direto utilizando o seu id na hash de users através de lookup, uma das vantagens da utilização de hash tables. A lista de ID's dos posts efectuados por esse user pode ser obtida por uma travessia a cada hash de perguntas e respostas armazenando ordenadamente por data cada post que tem como owner user id o id do user em questão. Este mecanismo é efectuado em ambas as hash tables guardando o id das 10 melhores perguntas e 10 utlimas respostas separadamente e ordenadas. Por final escolhe-se os ids dos 10 ultimos posts entre as 20 perguntas e respostas.

5.5 Query 6 - most voted answers

Questões com mais votos obtidas dentro de um determinado intrevalo de tempo.

5.5.1 Estratégias

Travessia com iterator sobre a hash table de respostas em que para cada resposta é testada a sua data em relação ao intrevalo de tempo fornecido. Caso este teste seja verídico e caso o score da resposta atual seja maior que o pior score armazenado até agora, é efectuada uma inserção ordenada aonde a regra de ordenação é o score. Este paço é efectuado por uma função auxiliar.

5.6 Query 7 - most answered questions

Perguntas com maior numero de respostas efectuadas dentro de um dado intrevalo de tempo.

5.6.1 Estratégias

Graças à utilização da livraria glib para as estruturas de dados principais esta query pode ser resolvida com uma simples travessia sobre cada pergunta. No decorrer desta travessia é verificada a data de publicação da pergunta e caso esta esteja dentro do intrevalo fornceido é pedido o número de nodos da árvore de respostas dessa pergunta atravéz da função g tree n nodes que apareçe encapsulada dentro da função getSize(Pergunta p) que retorna o numero de nodos da arvore de respostas da pergunta p. Caso este valor seja maior que o menor anteriormente armazenado este é armazenado no seu lugar e uma ordenação é feita para manter o menor valor armazenado no índice N-1;

5.7 Query 8 - contains word

Dado uma palavra são procuradas N perguntas que contêm a palavra procurada no seu título;

5.7.1 Estratégias

Travessia genérica à semeçhança de outras querys em que é utilizada uma função auxiliar searchTitle(word, getTitle(p)) que testa se a palavra existe dentro de uma pergunta retornando 1 no caso de verdade.
Ordenação da inserção por data.

5.8 Interrogação 9- both participated

Esta interrogação devolve uma lista de tamanho N com ids de perguntas onde dois dados utilizadores participam.

5.8.1 Estratégia

Esta interrogação requereu um estrutura auxiliar, com os ids dados, flags para o mesmos e uma lista de Perguntas de tamanho N, percorrendo a HashTable de perguntas e a sua árvore em busca dos dois ids que caso estejam presentes são inseridos na lista perguntas que está ordenada por data. A travessia sobre a árvore exige a utilização de foreach daí a necessidade de criar uma estrutura auziliar query de modo a que a informação que se prentende ser passada à função utilizada pelo for each seja passada toda encapsulada num tipo de dados só visto que foreach apenas aceita um tipo de dados adicional para além do par chave valor que passa à função por omissão.

5.9 Interrogação 10 - most used best rep

Esta interrogação devolve a melhor respostas a partir do id de uma dada perguntas.

5.9.1 Estratégia

De forma a responder a esta questão optamos por uma procura simples na HashTable de perguntas percorrendo depois a respetiva árvore de respostas ficando com a resposta com melhor Rate calculado para cada resposta no parser. A travessia da tabela é efectuada atravéz da utilização de foreach com uma função adicional iterateRate que guarda a resposta com melhor rate comparando cada elemento com a última resposta guardada.

6. *Conclusão*

Durante a realização deste projeto deparamos-nos com bastantes desafios sendo que o maior foi sem duvida a dimensão dos dados, forçando a bastante reflexão sobre a melhor maneira possível de responder às interrogações. Contudo apesar da não conseguirmos responder à interrogação 11 sentimos-nos satisfeitos com trabalho realizado, pois permitiu consolidar conhecimentos que até então não tivéramos oportunidade de por em prática num trabalho tão extenso e complexo.