Projeto de Laboratórios de Informática III
Grupo 25

Alexandre Mendonça Pinho (a82441) Joel Filipe Esteves Gama (a82202)
Tiago Martins Pinheiro (a82491)

5 de Maio de 2018

Conteúdo

Capítulo 1

Introdução

Este relatório apresenta uma explicação da primeira fase do projeto da disciplina de Laboratórios de Informática III, do 2º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho, que toma a forma de um projeto a ser desenvolvido na linguagem de programação imperativa *C*.

Como projeto de avaliação é nos proposto responder a onze interrogações sobre processamento de dados de comunidades do Stack Exchange. Como forma de resposta desenvolvemos as estruturas de dados, tivemos em consideração a abstração e modulação dos dados e também usamos os algoritmos que nos pareceram mais adequados para conseguir um bom resultado nos tempos de resposta de cada interrogação.

Neste relatório apresentamos a descrição do problema a resolver, assim como explicamos os tipos de dados, as estruturas usadas e as estratégias de resposta e melhoria dos algoritmos de resposta às interrogações.

Capítulo 2

Descrição do Problema

O projeto está dividido em duas fases, sendo o objetivo desta primeira fase responsde às onze interrogações utilizando a linguagem de progamação *C*. Para responder às onze interrogações foi necessário a escrita de várias funções, sendo também necessário fazer o debugging, e ter conhecimentos de algoritmos de procura e da biblioteca de parsing de ficheiros XML.

Em cada interrogação são pretendidos resultados distintos. Utilizando os tipos já disponibilizados pelos professores implementamos as funções de resposta, assim como as funções auxiliares para cada interrogação.

Capítulo 3

Conceção da Solução

3.1 Estruturas de dados usadas

Nesta fase do projeto utilizamos várias estruturas de dados para conseguir um melhor desempenho:

- Struct TCD_community, estrutura com a informação das tabelas de hash e das listas onde estão armazenados os diferentes tipos de dados.
- Struct tag_count, estrutura que tem o id da tag, o nome da tag e o número de vezes que foi usada.
- Struct date, estrutura que representa uma data, com a informação do dia, mês e ano.
- Struct linked_list, implementação de listas ligadas.
- Struct llist, implementação de uma lista de longs.
- Struct str_pair, estrutura que representa um par de strings.
- Struct long_pair, estrutura que representa um par de longs.
- Struct post, estrutura que armazena a informação relevante de um post.
- Struct tag, estrutura que armazena a informação relevante de uma tag.
- Struct user, estrutura que armazena a infromação relevante de um user.

3.2 Modulação funcional e Abstração de dados

A modulação do código torna o desenvolvimento da aplicação estruturado, tornando mais fácil a manutenção e reutilização do código. Para isto, são utilizados os ficheiros *header* e o código é dividido em vários ficheiros. Estes diferentes ficheiros são usados para criar 'classes' de 'objetos', assegurando a abstração de dados e o encapsulamento.

- O ficheiro *common.h* define funções de utilidade geral.
- O ficheiro *community.h* define as funções de acesso à estrutura de dados principal e as funções de resposta às queries.
- O ficheiro *date.h* define as funções de acesso à estrutura de dados Date.
- O ficheiro *interface.h*, tal como o nome indica, implementa a interface.
- O ficheiro *linked_list.h* define as funções de acesso à estrutura de dados LINKED_LIST.
- O ficheiro *list.h* define as funções de acesso à estrutura de dados LONG_list.
- O ficheiro *pair.h* define as funções de accesso a estrururas de dados dos tipos STR_pair e LONG_pair.
- O ficheiro *post.h* define as funções acesso à estrutura de dados POST.
- O ficheiro *tag.h* define as funções acesso à estrutura de dados TAG.
- O ficheiro *user.h* define as funções acesso à estrutura de dados USER.

3.3 Estratégias seguidas em cada uma das interrogações

3.3.1 Interrogação 1

O utlizador pretendido é acedido através da tabela de hash, e é retornado um par de strings que contém o título do post e o nome do autor do mesmo (acedido através do id do utilizador, guardado na estrutura do post).

3.3.2 Interrogação 2

Cada utilizador é inserido numa LONG_list de tamanho N, ordenadamente (algoritmo de inserção do 'insertion sort') em função do número de posts (descendente).

3.3.3 Interrogação 3

É iterada a lista de posts, mantendo a contagem do número de perguntas e respostas feitas num dado intervalo de tempo. No final, é retornado o par de longs destas contagens.

3.3.4 Interrogação 4

É iterada a lista de posts, e para cada post, verifica que foi criado dentro do intervalo de tempo dado, que é questão, e que contêm a tag especificada. Caso se verifique, adiciona o post a um lista ligada (e incrementado um contador do seu tamanho). Cada post dentro desta lista é inserido ordenadamente numa lista de longs do tamanho da lista ligada por ordem cronológica inversa.

3.3.5 Interrogação 5

É criado um utilizador 'dummy' para conter a informação de perfil e os IDs dos últimos 10 posts (guardados na estrutura do utilizador) do utlizador em questão.

3.3.6 Interrogação 6

É iterada a lista de posts, e para cada post, é verificado se é resposta e se foi criado dentro do intervalo de tempo dado. Se sim, é adicionado o respetivo id por ordem decrescente de pontuação (campo *Score*) num lista de longs com N elementos.

3.3.7 Interrogação 7

É iterada a lista de posts, e para cada post, é verificado se é uma pergunta e se foi criado no intervalo de tempo dado. Nesse caso, é inserido o seu id por ordem decrescente do número de respostas num lista de longs com N elementos.

3.3.8 Interrogação 8

Para cada questão dentro da lista de posts, é testado se o seu título contém a sequência de carateres dada como argumentos. Caso contenha, o respetivo id é inserido por cronologia inversa, numa lista de longs e N elementos no máximo.

3.3.9 Interrogação 9

São inseridos os ids dos posts em que participaram ambos os utilizadores dados numa lista de longs com N elementos ordenado por cronologia inversa. Para determinar se ambos os utilizadores participaram num post, é iterada a lista dos posts, e para cada pergunta, é verificado se, um deles é autor da pergunta e outro de uma resposta ou se ambos são autores de pelo menos uma resposta.

3.3.10 Interrogação 10

É percorrida a lista dos ids das respostas da pergunta e, para cada resposta, é calculado a média ponderada, especificada no enunciado, e depois retornado o id da resposta com a média mais alta.

3.3.11 Interrogação 11

Primeiro é calculada a lista dos ids dos N utilizadores com melhor reputação, e a partir dessa lista, é calculada a lista dos ids das N tags mais utlizadas por estes utilizadores dentro de um determinado intervalo de tempo. Para calcular a primeira lista, é iterada a lista de utilizadores e inserido numa lista de N longs o id de cada utilizador em função da sua reputação (ordem decrescente). Na segunda fase, é iterada a lista de posts, e contada a frequência absoluta da ocorrência de cada tag (através da estrutura *TAG_COUNT*), excluindo posts que não foram criados dentro do intervalo de tempo dado ou que não foram postados pelos N utilizadores com melhor reputação. Por fim, para cada tag contabilizada, é inserido o seu id por ordem decrescente da contagem da sua ocorrência.

3.4 Estratégias para melhoramento de desempenho

Para inserir elementos de forma ordenada nas listas pedidas como resultado, é utilizada uma procura linear. Devido à aparente ineficiência deste processo pois é de complexidade linear, foi tentada uma implementação usando árvores de procura binária com um certo tamanho máximo (retirando o valor máximo/mínimo sempre que necessário para manter o tamanho). Teóricamente, esta alteração resultaria numa redução do tempo de execução, especialmente para inputs de tamanho maior.

O que se verificou, no entanto, foi exatamente o oposto. O tempo de execução era ordens de magnitude superior ao original. Isto deve-se á forma como estão dispostos os dados: como os posts no ficheiro Posts.xml estão quase ordenados cronológicamente, a inserção numa lista por procura linear torna-se em média muito mais eficiente (tempo constante) que a inserção numa árvore binária (que se tornava muito desiquilibrada). Para além disso, existia o *overhead* de manter a árvore binária com um certo número de elementos, que não existia no caso de uma simples lista.

Capítulo 4

Conclusão

Foi-nos proposto, como projeto de avaliação, fazer um programa com capacidade de responder a onze interrogações sobre dados provenientes de comunidades do Stack Exchange. A implementação foi feita na linguagem de programação C, mas utilizando conceitos de programação por objetos e garantido a abstração e dados e o encapsulamento.

Este método de desenvolvimento, tal como a utilização de ferramentas Unix, foi-nos útil no raciocínio sobre as operações do programa e no debugging.

Para torna o código mais sucinto e extensível era possível generalizar as funções de inserção ordenada.

Uma possível otimização seria implementar uma árvore de procura binária equilibrada, garantido a inserção em tempo *quasi-linear*. No entanto, é duvidoso que isto seria mais eficiente que a implementação atual devido à estrutura dos ficheiros XML do dump, como explicado antes.