Universidade do Minho
Escola de Engenharia

Trabalho Prático em C

LI3

Mestrado Integrado em Engenharia Informática

2º Semestre
2017-2018

A77045 - Ricardo Barros Pereira
A74260 - Luís Miguel da Cunha Lima
A79034 - Rafaela Maria Soares da Silva
A81898 - Hugo Manuel Gomes Nogueira

5 de maio de 2018
Braga

Resumo

Perante a proposta, a desenvolver ao longo da primeira fase do projeto, para criar um sistema capaz de processar ficheiros *XML* que armazenam as informações utilizadas pelo *Stack Overflow*, e extrair a informação útil, foi exigida alguma reflexão devido à grande quantidade de dados e consequente eficiência de resposta às interrogações. Tudo isto requereu a escolha acertada, ou apropriada, das estruturas de dados a utilizar, e também um estudo da biblioteca *"libxml2"*, como forma necessária de ler e carregar toda a informação útil dos *backups* já mencionados, para uma *struct*.

Depois de realizado todo este trabalho, o resultado encontrado foi eficiente e os objetivos e respostas às interrogações cumpridos.

Conteúdo

1 Introdução

O projeto *"Stack Overflow"* surge no âmbito da primeira fase do projeto de Laboratórios de Informática (LI3), do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Sendo um projeto a ser resolvido na Linguagem "C", este baseia-se na construção de um sistema que permita analisar os artigos presentes em *backups* do *Stack Overflow*, e na extração de informação útil, como por exemplo, saber quais são os utilizadores mais ativos ou quais são os temas mais comuns, entre outros. Deste modo, houve a necessidade de pensar em estruturas de dados e algoritmos para a resolução de interrogações no menor tempo possível e de forma eficiente. Numa primeira fase, foi essencial o estudo da biblioteca *"libxml2"* como forma de fazer *"parse"* aos *backups* e posteriormente guardar a informação útil nas estruturas e com os tipos que melhor se adequavam. Depois de carregados todos os dados necessários, numa segunda fase foram implementadas as respostas às *"queries"* propostas pelos docentes. Já no fim do projeto, os testes dos docentes às implementações propostas foram essenciais à comprovação das mesmas.

Em suma, a Secção 2 descreve o problema a resolver, enquanto a Secção 3 apresenta e discute a solução proposta pelo grupo, desenvolvida ao longo do semestre. O relatório termina com a conclusão na Secção 4, onde é apresentada uma análise crítica dos resultados obtidos. Além disso, é feito um balanço tendo em conta as dificuldades ao longo do desenvolvimento do projeto.

2 Descrição do Problema

No contexto da primeira fase do projeto de LI3, pode-se afirmar que foram propostas três tarefas. Como primeira instância, seria necessário estudar a fundo a biblioteca *"libxml2"*, de seguida, era requerido que fosse implementada uma estrutura para armazenamento dos dados, *"TCD istruct"*, bem como um conjunto de funções que tirasse partido da mesma e, por fim, dar resposta a onze *"queries"*.

A primeira tarefa permitiu uma maior familiarização da biblioteca *"libxml2"*, pois foi com o uso da mesma que foi possível percorrer os backups fornecidos pelos docentes, e extrair para a estrutura a informação necessária à resposta das interrogações. Foi também essencial o tutorial presente no site da própria biblioteca que serviu como guia de execução na elaboração do método responsável pela extração da informação do ficheiro *XML* (*parse*).

Numa segunda tarefa, realizamos todas as funções de criação, inserção, procura, e remoção necessárias para conseguirmos respostas eficientes às interrogações propostas. Encontram-se também a resolução já de algumas *"queries"*, como as de carregamento de dados, inicialização da estrutura e remoção da mesma.

Por último, seguia-se a tarefa de mais rápida resolução, pois apenas se baseava em conceitos básicos da linguagem C, e todas as estruturas e dados úteis já estariam acessíveis. Esta parte do projeto implicou dar resposta às várias interrogações, e verificar, se as mesmas estariam a ser executadas corretamente e em tempo razoável.

3 Conceção da Solução

Em grupo, foi proposto que, em conjunto, seriam resolvidas as grandes questões acerca da escolha e implementação da estrutura de dados, da análise e inserção de dados na estrutura. Estando esta primeira fase resolvida, passamos à distribuição das *queries* pelos membros do grupo. Procurou-se, assim, obter uma linha de trabalho clara, eficaz e que nos permitisse obter os resultados esperados.

Dividimos a modulação das tarefas em dois grupos, entre os quais *"Memory Maintenance"* e *"queries"*. Estas modulações correspondem, respetivamente, a *queries* responsáveis por alocar a memória necessária para as estruturas de dados a utilizar e carregamento de dados a partir do documento *XML* e a *queries* mais complexas juntamente com outras que estariam mais relacionadas com um tratamento menos complexo dos dados recolhidos.

3.1 Estruturas de Dados

A estrutura de dados a utilizar pode ter um grande impacto no desenvolvimento das tarefas e na eficiência, portanto foi necessário desde logo definir como tratar a informação passível de ser manipulada pelo programa.

Numa fase inicial, analisamos e avaliamos dois tipos de estruturas de dados, as *"AVLTree"* e as *"HashTable"*, pois são as estruturas mais eficientes para grandes quantidades de dados. Aparentemente as *HashTable* teriam uma *performance* melhor e mais rápida naquilo que seriam os tempos de inserção e procura, *0(1)*. Quando comparado com os tempos das *AVLTree*, *0(log n)*, parece fazer todo o sentido escolher a *HashTable*, mas surge o problema de que em muitas das *queries* a realizar nos são pedidas respostas por uma certa ordem e este tipo de dados não oferece armazenamento de dados por ordem. Por outro lado, as *AVLTree* inserem os dados por ordem e a nossa procura seria muito mais facilitada por esse aspeto, e ainda, contando com o facto de que seria muito mais complicado implementar uma *HashTable* e para funcionar corretamente e de forma eficiente o seu algoritmo teria de ser perfeito.

A menos que o tamanho máximo da *HashTable* seja conhecido e estabelecido no momento em que a tabela é criada, esta se redimensionará periodicamente conforme os dados são adicionados, o que pode levar a uma "degradação"do desempenho. Este não é um problema quando são usadas as *AVLTree*. Analisando todas estas variantes, optamos por implementar *AVLTree's*.

Relativamente ao tipo concreto de dados, como esperado, requereu bastante esforço de programação pois o objetivo é que características como encapsulamento, robustez, segurança sejam garantidas de uma forma simples e eficaz. Para isto foi necessário implementar algumas técnicas para em C podermos criar

um correto módulo de dados, reutilizável e com implementação escondida apenas acessível via acesso a uma API (interface.h). Sendo assim, a nossa estrutura geral é demonstrada a seguir:

```
typedef struct TCD_Community{
    Post post;
    User user;
}TCD_Community;
```

Dentro desta estrutura existem duas estruturas, uma capaz de guardar todos os *Posts* publicados na plataforma, com todas as informações mais importantes. Tais como o título, o utilizador que o publicou e outras não menos relevantes. E ainda outra estrutura relativa a todos os utilizadores presentes na plataforma do *Stack Overflow* e suas informações.

O nosso tipo abstrato de dados, TAD_Community, foi declarado no ficheiro interface.h como sendo um apontador para o TCD. É esta a regra que garante em C um "data hiding", ou seja, esconder dos utilizadores a implementação dos dados, garantindo assim que estes não tenham acesso direto.

```
typedef struct TCD_Community * TAD_Community;
```

Visto que o projeto apresenta uma dimensão considerável, decidiu-se optar por uma organização cautelosa, no intuito de zelar por um código legível e "controlável". O projeto foi dividido em três partes – load, init e parse. No init, inicializa-se a estrutura. No load, os dados são carregados para a estrutura. E, por último, no parse, é feito o *parsing* dos *posts* e dos *users* pela biblioteca *libXml2*. Para os módulos terem acesso a informações(como as referentes ao utilizador por exemplo) é necessário criar uma *header file*(.h) que seja utilizada somente para se tornar acessível aos outros módulos, sem conter código fonte C.

4 Estratégias

4.1 Pergunta 1

Para a interrogação 1, *info_from_post*, dado o identificador de um *post*, a função deve retornar o título do *post* e o nome do autor da pergunta. Inicialmente decidimos criar a função *getPost* para conseguirmos extrair facilmente a informação necessária do respetivo *post*. Esta informação resume-se ao tipo de *post*, à obtenção do título do *post* e ao *id* do utilizador. Criou-se ainda outra função, *getName*, a partir do *post* em questão e do *id* de utilizador, para ter acesso ao nome do utilizador. Assim, obtemos o pretendido, o título do *post* e o nome do utilizador.

4.2 Pergunta 2

Esta pergunta tinha como objetivo obter os utilizadores mais ativos na plataforma do *Stack Overflow*, ou seja, os utilizadores com mais *posts* de sempre. Inicialmente foram criados dois *arrays* com o mesmo tamanho capazes de armazenar os *ids* do top utilizadores e no outro *array* o respetivo número de *posts* de cada utilizador. Era necessário percorrer a estrutura onde estão armazenados todos os utilizadores e conforme vamos descendo na estrutura vamos comparando o atributo *numPosts* que contém o número de *posts* que cada utilizador publicou na plataforma. Se este utilizador, contiver um número de *posts* maior do que o primeiro *id* do *array* (*arrays* encontram-se ordenados crescentemente) este é substituído pelo novo *id* e pelo novo número de *posts* aos respetivos *arrays*. De referir ainda que para utilizadores com o mesmo número de *posts*, prevalece o utilizador a mais tempo encontrado.

4.3 Pergunta 3

O objetivo desta pergunta é bastante simples - queremos obter o número total tanto de perguntas como de respostas efetuadas num determinado intervalo de tempo arbitrário. Para alcançar este objetivo, percorremos a árvore anteriormente criada e vamos comparando as datas respetivas de cada *post* (utilizando a função *smaller*) com o intervalo de tempo dado. Se este estiver do intervalo apenas é verificado o tipo de *post* (pergunta ou resposta) e incrementada a variável respetiva.

4.4 Pergunta 4

Cada pergunta publicada na plataforma tem as suas respetivas *tags* que dizem respeito ao tema do *post*. Nesta pergunta, queremos obter todas as perguntas que contenham uma determinada *tag* fornecida. Para isto foi criado um *Array* auxiliar e sempre que era encontrada uma pergunta com determinada *tag* e dentro do intervalo de tempo definido era adicionado o *Id* dessa pergunta ao *Array* Auxiliar.

4.5 Pergunta 5

Para esta pergunta 5, queremos identificar a atividade do utilizador, ou seja, um pequena biografia sobre o utilizador e os seus últimos dez *posts* publicado por este mesmo. Maior parte do trabalho efetuado para responder é feito quando são inseridos todos os *posts* nas estruturas construidas. Cada utilizador contém informação permanente dos seus últimos dez *posts*, ou seja, foram criados dois *arrays* para cada utilizador, um com os *Ids* dos últimos dez *posts* deste e outro com a respetiva data da publicação do respetivo *post*. À medida que é adicionado um novo *post* de um determinado utilizador, estes *arrays* são atualizados de forma a que estes *arrays* tenham sempre guardados os dados relativos dos seus últimos dez *posts*. Tendo esta informação, basta procurar o utilizador e buscar estas informações.

4.6 Pergunta 6

O objetivo desta pergunta é mostrar as respostas com melhor classificação dentro de um intervalo de tempo arbitrário, para isto é necessário fazer a diferença de *Up Votes* e *Down Votes*. A estratégia para a resolução desta pergunta tem muitas semelhanças a algumas anteriormente explicadas, em que é

percorrida toda a árvore dos *posts* em que cada *post* contém um atributo com a diferença dos *Votes*, e guardamos os *ids* das melhores respostas e que estejam dentro do intervalo de tempo, num *Array* que depois é retornado como resposta a esta pergunta.

4.7 Pergunta 7

Para esta pergunta, queremos identificar as perguntas com maior número de respostas dentro de um intervalo de tempo pré definido. Para isto baseamo-nos num atributo presente nos *posts*, o *nAnswers* que é calculado aquando do *load* dos dados dos ficheiros de *XML*. Sempre que um *post* é adicionado e se o seu tipo for do tipo resposta, este valor (*nAnswers*) é atualizado na pergunta ao qual a resposta foi efetuada. Assim, apenas temos que percorrer a árvore dos *posts* e verificar as perguntas que tem o atributo *nAnswers* maior e que tenha sido publicado, como dito em cima, no intervalo de tempo fornecido como argumento.

4.8 Pergunta 8

Esta interrogação tem como objetivo devolver um determinado número de *Ids* de perguntas que contenham uma dada palavra no seu título. Para responder a esta interrogação foram criados dois *Arrays* - um para os *Ids* pretendidos e outro para os *Timestamps* correspondentes, que vão sendo preenchidos conforme é percorrida a árvore dos *posts*. Finalmente é feita uma ordenação de forma a apresentar a pergunta mais recente no início da lista. Como não foi referido no enunciado da pergunta se o objetivo era guardar os N *posts* mais recentes ou os mais antigos, o que o grupo optou por fazer foi devolver os primeiros N *Ids* das perguntas que contém a determinada palavra.

4.9 Pergunta 9

O objetivo desta pergunta é devolver as últimas N perguntas em que participaram dois utilizadores passados como parâmetro. Para isto foi usada uma classe auxiliar, de nome *Array*, capaz de guardar e gerir dados usando funções auxiliares como *initArray*, *existe*, *insertArray* e *freeArray*. Inicialmente começamos por guardar em dois *Arrays* falados anteriormente, um para cada utilizador, todos os *posts*, tanto perguntas como respostas, em que os utilizadores participam. Sempre que um utilizador publica uma resposta é guardada no seu respetivo *Array* o *Id* da pergunta ao qual a resposta foi efetuada. De seguida, são combinados os *Arrays* dos dois utilizadores guardando apenas os *Ids* dos *posts* em que aparecem nos dois *Arrays*. Finalmente este ultimo *Array* é ordenado e devolvido conforme o pretendido.

4.10 Pergunta 10

Esta interrogação tem como objetivo obter a melhor resposta de uma respetiva pergunta conforme uma fórmula de calculo específica. Para isto é percorrida toda a estrutura dos *posts* e verificado se o tipo de *post* é do tipo resposta e que contenha o *Id* da pergunta igual ao *Id* fornecido. Se sim, é aplicado a fórmula anteriormente falada. No fim, a melhor resposta é aquela que obtiver um maior valor conforme o *score* da resposta, a reputação do utilizador, o seu número de votos e de comentários.

5 Conclusão

Tendo em conta o objetivo do projeto, a criação de um sistema que permitisse analisar a imensa quantidade de dados do *Stack Overflow* criou-nos de facto dificuldades que tivemos de ultrapassar. A implementação deste sistema permitiu não só o desenvolvimento das capacidades de raciocínio e programação, mas também serviu para podermos perceber um pouco daquilo que será o nosso futuro, a nível do uso de bibliotecas, análise de grandes quantidades de informação.

O maior problema deste projeto foi, conjugando o facto de que usamos uma biblioteca já existente, encontrar o raciocínio que nos permitisse ter a melhor eficiência possível. Toda a manutenção de memória e carregamento e extração de dados permitiu uma resolução simples e eficaz das *queries* propostas.

Embora tenhamos utilizado uma estrutura que armazenou por *id's*, várias *queries* exigiam-nos que procurássemos informação por datas, ou seja, poderíamos ter utilizado uma estrutura que ordenasse os *posts* por data, como um calendário.

Por último, o resultado foi positivo. A implementação deste sistema foi conseguida, e todo o trabalho foi recompensado com o resultado, tanto um sistema capaz, como tudo o que aprendemos e melhoramos com a realização do mesmo.