Universidade do Minho

Mestrado Integrado em Engenharia Informática

2º Ano, 2º Semestre

Unidade Curricular de Laboratórios de Informática III

Ano Letivo de 2017/2018

1.º Trabalho Prático de LI 3

(Processamento de ficheiro XML Stack Overflow)

Grupo 57

André Gonçalo Castro Peixoto (a82260)

Diogo Emanuel da Silva Nogueira (a78957)

Diogo Francisco Araújo Leite da Costa (a78485)

Luís Filipe da Costa Cunha (a83099)

Índice

1. Introdução

Este projeto foi enunciado pelos docentes da Unidade Curricular Laboratórios de Informática 3 e o seu principal objetivo foca-se no desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as várias informações presentes no *Stack Overflow*. Paralelamente a este objetivo principal, existe também a ideia de se consolidar todos os conhecimentos adquiridos em Unidades Curriculares anteriores como Programação Imperativa e Algoritmos e Complexidade.

Assim, aliando uma programação mais complexa e em larga escala, através do processamento duma grande escala de dados, pretende-se criar um programa que faça jus ao pretendido e que vá ao encontro com todo o universo da Modularidade e Encapsulamento de Dados da linguagem C.

2. Estruturas e Carregamento dos Dados

Para se poder responder as interrogações a que se pretendia dar resposta segundo o enunciado, criou-se uma "estrutura-mãe" *TAD_community* que suporta todas as sub-estruturas que foram necessárias para existir uma resposta correta e otimizada desde início para cada pedido de análise dos ficheiros XML. Foram criadas 3 sub-estruturas sendo que serão faladas em pormenor nos subcapítulos seguintes.

2.1. Estrutura *hashPosts*

Esta estrutura foi criada com o intuito de armazenar todos os *posts* contidos no ficheiro Posts.xml. Com o suporte das funções *init()* e *load()* a estrutura escolhida foi uma *HashTable* da biblioteca *open-source GLIB*. Utilizando a biblioteca extensiva do *libxml2*, para fazer *parse* do documento Posts.xml para a estrutura *hashPosts*, foi usado uma *struct* que serve de intermediário entre o documento XML puro e a informação já guardada e analisada pelo programa, chamada *PostsAtr*, que irá servir de esqueleto para os atributos do XML dos *posts* já com o *parse* efetuado. Esse esqueleto terá as informações/atributos que serão necessários para as *queries* pedidas, como o *id*, *score*, *ownerUserId*, *tag*, etc. Nessa estrutura auxiliar também teremos outras sub-estruturas que nos facilitaram o trabalho de certas interrogações, tal como árvores de respostas dos *posts* que são perguntas e também listas ligadas com informações vitais para responder às interrogações pedidas.

Dessa forma, em vez de existir um *parse* sempre que seja necessário para uma *query*, agrega-se e filtra-se a informação relativa aos *posts* logo de início para otimizar depois o tempo da resposta à *query* pedida. Para algumas interrogações houve a criação de listas ligadas e um contador de forma a poder contabilizar os vários *posts* que possivelmente possam ter a mesma data, mas sendo na mesma possível estarem introduzidos na árvore que tem como método de ordenamento por datas. Assim sabemos de forma bastante otimizada e rápida, o número e o conteúdo de *posts* que possam ter datas idênticas (que aconteceu frequentemente).

2.2. Estrutura *hashUsers*

A outra estrutura utilizada em peso foi a *hashTable* que focar-se-á no armazenamento do *parse* das informações do XML dos utilizadores. Foi utilizada a mesma técnica de armazenamento com a ajuda das funções do GLIB que encapsulam, tratam e otimizam o armazenamento dos dados de forma eficiente e autónoma. Uma estrutura auxiliar foi criada, chamada *UserAtr* que receberá os atributos XML já com o *parse* efetuado, tais como o *id, displayName, aboutMe, reputation, etc.* Por sua vez também terá contadores como o nº de *posts* que um certo utilizador fez ou mesmo uma árvore desses mesmos *posts* ordenados por datas.Tal como na estrutura anteriormente falada, estes argumentos foram pensados para responder automaticamente às interrogações pedidas, dado que a informação estaria logo disponível a partir do *parse* inicial ao invés de estar a correr apenas quando a interrogação fosse pedida.

Toda esta *struct* será colocada como "valor" na *hashUsers* e como chave o *id* equivalente ao utilizador também. Desta forma, conseguimos um desempenho excelente dado que ao buscar a informação dum certo *id* de utilizador temos logo a informação otimizada de forma imediata dado que não é necessário percorrer uma *HashTable*, porque a procura da mesma baseia-se num índice de *hashes* e assim torna todas as interrogações mais eficazes e céleres.

2.3. Estrutura *hashTags*

Como última estrutura principal temos uma *HashTable* que suportará o último *parse* do ficheiro das Tags em XML. O procedimento é semelhante às *HashTables* anteriores sendo que foi criado uma *struct* que albergou todos os atributos das tags, tais como *id, tagname, count* mas informações como o nº de vezes utilizada de forma a poder dar resposta a todas as interrogações que necessitavam da informação relativa ou associadas às *tags*. Essa estrutura será posta como "valor" na *hashTags* e assim podemos obter a sua informação de forma rápida através das funções

pré-definidas da biblioteca GLIB e dessa forma responder com exatidão a todas as interrogações que utilizem total ou parcialmente das informações XML das *tags*.

3. Interrogações

3.1. Primeira Interrogação

Nesta primeira interrogação começa-se por obter toda a informação associada ao ID do *Post* que se pretende procurar, por recorrência à função *g_hash_table_lookup* (pré-definida no *GLIB*) e usando-se a *struct hashPosts* e o ID como parâmetros da mesma. **Tendo-se toda a informação (*value*) do *Post* pretendido surgem duas opções:**

- Caso se verifique que o *Post* não se trata de uma resposta, será necessário se recorrer à *struct* hashUsers de modo a obter, através do *ownerUserID* que identifica o *ID* do utilizador desse Post, o seu nome. Após isso, facilmente se consegue também o título do mesmo.
- Caso o *Post* seja uma resposta, é necessário pesquisar a pergunta a que corresponde, através do *parentID* que identifica o *ID* da pergunta a que diz respeito. Com isto, devolve-se o título e o nome de utilizador do criador dessa mesma pergunta.

3.2. Segunda Interrogação

Na segunda interrogação faz-se uso da árvore de listas *nUserPostsTree* que se encontra ordenada pelo número total de posts correspondentes aos vários utilizadores existentes. Nesta árvore a *key* corresponde ao nº de *posts* e cada nodo possui uma lista com os *Users* cujo nº de *posts* é igual a essa mesma *key*. Esta implementação da *struct nUserPostsTree* é importante de ser estabelecida para esta *query* de modo a facilitar a procura, poupando tempo e indo-se logo de encontro com o pretendido.

O que acontece é que se recorre a um ciclo *while* que existe enquanto o número N de utilizadores que se pretende obter já estiver finalizado. Ou seja, apenas é necessário recorrer à *GTree nUserPostsTree* (já ordenada e otimizada no *parse*) e ir obtendo dela os demais *Users* e os seus respetivos números de *Posts*.

3.3. Terceira Interrogação

Nesta terceira interrogação, é imperativo recorrer à *GTree treeRes* que aglomera todas as respostas dos vários *Posts* ordenadas por datas, bem como à *GTree treePerg* que trata da mesma o conjunto de perguntas. **A sequência de passos a seguir é equivalente tanto para as respostas como as perguntas:**

- Faz-se pela primeira vez o uso da *struct Info*. Esta é uma estrutura genérica criada com o intuito de se controlar/manipular as várias estruturas sempre que as percorremos através do *foreach* do *GLIB*. Por ser geral, aglomera todos os campos essenciais de modo a atender às várias interrogações/estruturas de dados.
  Para este caso, atualizamos nessa mesma *struct* a data de início e de fim do intervalo e ainda o contador de perguntas e respostas (que é o que se pretende retornar);
- Através de uma função criada para o efeito, percorre-se cada elemento da árvore *treeRes*, fazendo uso da função *g_tree_foreach* do GLIB para se poder percorrer cada um destes elementos e ainda se testar se cada resposta está no intervalo desejado (com o uso da *struct Info*, tal como foi anteriormente explicado). Caso esteja, apenas é atualizado o valor do contador da *struct Info*. O mesmo método se segue para a àrvore de perguntas.
- Por fim, retira-se o valor do contador do número de respostas e perguntas presentes nessa *struct_Info*.

3.4. Quarta Interrogação

Esta quarta interrogação faz uso da *struct Info* (já explicada e abordada em *queries* anteriores) de modo a auxiliar com as mecânicas existentes da *GLIB*. O uso desta *struct* passa pela atualização de alguns dos valores necessários para a *query* em questão, tais como o *tagcount* que servirá de base quando se precisar de verificar se a pergunta possui a *Tag* passada como parâmetro.

Através da função *TreeForEachPergTag*, percorremos a *treePerg* e verificamos se cada uma das perguntas desta árvore se encontram no intervalo de tempo em causa. Assim, se a pergunta possuir a mesma *Tag* da que foi passada como parâmetro, inserimos o *ID* numa lista, ordenadamente por cronologia inversa (tal como pretendido).

Para se ordenar a lista com os *IDs* de todas as perguntas dentro do intervalo de tempo passado como parâmetro, o grupo optou por verificar se a data de um *post* é ou não maior que a data de outro *post*, através do dia, ano e mês em que cada um destes ocorre.

3.5. Quinta Interrogação

Uma vez que se pretende obter informação proveniente do *User*, foi necessário nesta query percorrer a *HashTable hashUsers* que acumula todos os utilizadores existentes. Esta *HashTable* apenas é usada para se obter o conjunto de informações do *User* cuja chave é a passada como parâmetro na *query*. Após se ter obtido toda a sua informação, unicamente importa ficar com a árvore que contém todos os seus *Posts*. Tendo-se esta árvore, percorre-se a mesma através de uma função criada para o efeito, que trata de obter os últimos 10 *Posts* respeitando os critérios requeridos.

3.6. Sexta Interrogação

Para o desenvolvimento desta query, o grupo manteve desde início a noção de que o número de votos equivale ao *score* de uma determinada pergunta/resposta. Esta perceção foi fundamental para que se concluísse que era desnecessário fazer o *parse* de outros ficheiros *xml* que não os inicialmente idealizados perante as *queries* pedidas.

Para esta interrogação, fez-se mais uma vez uso da *struct* genérica *Info* para se poder filtrar apenas as respostas compreendidas na data desejada:

- Inicia-se uma *tree* vazia nessa *struct* (uma vez que é um dos campos da mesma);
- Com a *tree* inicializada, percorre-se a *treeRes* que contém todas as respostas e insere-se nessa árvore vazia, resposta a resposta, ordenadamente pelo *score* de cada uma;
- Com a árvore contendo as respostas todas ordenadas pelos *scores* apenas é necessário se percorrer a mesma e devolver apenas as suas N primeiras folhas.

A solução pensada para esta *query* considera o *score* total da resposta aquando da ordenação de todas as respostas na árvore anteriormente debatida.

3.7. Sétima Interrogação

Conforme foi explicado no capítulo anterior, para fazer *parse* do documento Posts.xml para a estrutura de dados *hashPosts*, foi usado uma *struct* chamada *PostsAtr*, que contém as várias informações/atributos que são necessários para as *queries* pedidas. Nesta *struct PostAtr* ficou definido que para cada pergunta existe uma árvore que amontoa todas as suas respostas. Esta foi uma das decisões pensada no futuro desenvolvimento das várias *queries* e neste caso em concreto, no facto de se precisar de percorrer todas as respostas de modo a contabilizar apenas aquelas que se encontram no suposto intervalo de tempo.

Com este pensamento definido, o desenvolvimento desta *query* tornou-se intuitivo e claro de se estabelecer:

- Numa fase inicial, percorre-se então toda a *postTreeResp* e ao fazer-se isto, vai-se contando o número de respostas que se encontram definidas naquele intervalo de tempo pedido, inserindo-se numa outra árvore de listas (desta vez a da *struct Info*) em que a *key* passa a ser o número de respostas total e o *value* passa a ser a pergunta em si. Desta forma consegue-se ter uma árvore em que se associa o número de respostas de uma determinada pergunta, otimizando e poupando trabalho/tempo para o próximo passo a ser estabelecido;
- Com esta árvore já definida é fácil de se obter os *IDs* das N perguntas com mais respostas uma vez que já se tem toda a árvore ordenada e devidamente associada.

A solução pensada para esta *query* considera o *score* total da resposta aquando da ordenação de todas as respostas na árvore anteriormente debatida.

3.8. Oitava Interrogação

O conceito desta query passa por percorrer, numa fase inicial, a *tree* que contém todas as perguntas ordenadas por datas. Á medida que se percorre cada pergunta desta árvore, verifica-se se a palavra em questão está contida no título da mesma. Caso esta verificação seja válida, insere-se essa pergunta numa lista, que naturalmente estará também ordenada por datas.

Após se ter essa *list*, basta através de um ciclo *while* ir dispondo numa outra lista inicialmente concebida, os N elementos (*IDs)* que correspondem às perguntas desejadas, começando pela pergunta com a data mais recente.

3.9. Nona Interrogação

Para esta interrogação é necessária a "extração" dos vários *posts* em que os dois utilizadores em causa intervieram. Primitivamente, obtém-se as árvores que contêm todos os posts de cada um dos utilizadores. Com isso, a ideia passou por criar um ciclo dentro de outro ciclo de modo a se criar uma comparação para se poder perceber se existe uma interação dos dois utilizadores em algum dos *posts*, ou seja, se o utilizador 1 respondeu a alguma pergunta do utilizador 2 (ou vice-versa) ou se ambos responderam a um *post* de outro alguém.

Com esta mecânica definida, o resto torna-se trivial. Devolve-se a lista com as N perguntas que foram devidamente filtradas/comparadas de acordo com o ambicionado.

3.10. Décima Interrogação

Dado o *ID* de uma pergunta, o objetivo passa por obter a melhor resposta através do cálculo de uma fórmula criada para o efeito. Para se poder chegar a esta conclusão é preciso antes de qualquer outra coisa, obter o "pedaço" da estrutura *hashPosts* que corresponde à pergunta que se pretende relacionar. Após se ter essa estrutura de dados e sabendo-se que a mesma possui na sua constituição uma outra estrutura (árvore) que contém todas as respostas associadas à pergunta em causa, a ideia passa por criar um *foreach* de modo a se percorrer cada resposta e perceber qual a melhor. Para se concluir qual a melhor criou-se uma função *encontraMelhorRespostaTree* que será a função passada para cada *each* executado na árvore e que efetua o cálculo da fórmula que define aquilo que implica ser a melhor resposta.

3.11. Décima Primeira Interrogação

Para esta última interrogação existiram várias tentativas da parte do grupo de modo a se produzir o resultado correto/esperado. Pensando no que era suposto responder, começou-se por percorrer a árvore com o conjunto de perguntas ordenadas por datas, filtrando apenas as que corresponde ao intervalo de tempo pedido e associando a reputação do utilizador a que corresponde cada uma das perguntas. Basicamente, acedendo-se aos *Users* com mais reputação no intervalo de tempo passado como parâmetro, verificaram-se todas as *tags* que esses mesmos usaram, metendo-as numa *hashtable* e contabilizando-se a quantidade de vezes que cada uma delas foi utilizada. Com este pensamento definido, restou devolver uma lista com as N *tags* mais usadas pelos N utilizadores.

É importante ainda referir que para se proceder à organização das várias *tags*, apenas se teve em conta os N utilizadores com mais reputação naquele intervalo de tempo.

4. Conclusão

Com toda uma fundamentação e explicação do projeto fica fácil fazer uma apreciação geral daquelas que foram as grandes dificuldades deste desafio. Apesar de apelar ao uso de uma linguagem por nós já conhecida e trabalhada, o projeto exigiu do grupo uma necessidade de aprofundar certos conhecimentos e de investigar novas ideias de como se poderia desenvolver um sistema eficiente, rápido e acima de tudo capaz de processar uma quantidade tão massiva de dados.

Levando todo o objetivo definido e o conceito de como tudo deveria funcionar, foi preciso estudar o funcionamento de toda a biblioteca *GLIB*, de modo a delimitar como deveriam ser desenvolvidas as várias estruturas de dados e consequentemente como manipular cada uma delas. A dificuldade maior esteve quase sempre na escolha/associação das várias estruturas e sub-estruturas, uma vez que o objetivo principal era não só responder às 11 interrogações, mas também tornar o sistema o mais célere e eficaz possível. Apesar de tudo isto, teve ainda de existir uma estruturação competente de todo o projeto para deixar a modularidade e encapsulamento de dados bem definidos.