Universidade do Minho

Mestrado Integrado em Engenharia Informática

Projeto de Laboratórios de Informática III

Grupo 41

Ana Ribeiro(A82474) Jessica Lemos(A82061) Pedro Pinto (A82535)

12 de Junho de 2018

Conteúdo

1 Introdução

Este relatório aborda a elaboração do mesmo projeto realizado na primeira fase da disciplina de Laboratórios de Informática 3 (LI3), do Mestrado Integrado em Engenharia Informática da Universidade do Minho, na linguagem de programação Java. Este projeto consiste na implementação de um sistema capaz de processar ficheiros XML que armazenam as várias informações utilizadas pelo Stack Overflow. Uma vez processada essa informação, pretende-se que seja possivel executar um conjunto de interrogações específicas, apresentadas posteriormente, de forma eficiente.

2 Concepção

2.1 Concepção do problema

Tendo em conta o problema apresentado, decidimos organizar o nosso trabalho em dois grupos:

- Posts
- Users

Para realizar as queries relativas aos posts é necessário obter do ficheiro *"Posts.xml"* as seguintes informações: o id e o tipo do post e no caso de este ser uma resposta guardar o id da respetiva pergunta, as tags, o seu título, o id do utilizador que o elaborou, a sua data de criação e o número de comentários. Caso o post seja uma pergunta terá de se guardar também o número de respostas. A fim de tornar a resolução das interrogações deste grupo eficientes, é evidente a necessidade de implementar uma estrutura onde seja rápida a procura dos posts de uma dada data. Acrescido a esta preocupação, é indispensável ter o cuidado de organizar os posts de uma data pelo seu id.

Quanto aos users, é oportuno guardar do ficheiro *"Users.xml"* o id, o nome, a reputação e a informação do seu perfil. Para tornar eficaz a resolução destas queries, é útil estruturar as informações pelo seu id.

Para as tags, é preciso armazenar do ficheiro *"Tags.xml"* o id e o seu nome, ordenadas por este.

2.2 Concepção da solução

Com o objetivo de solucionar todas as interrogações do grupo dos posts de modo eficiente utilizamos uma *HashMap* organizada por datas, em que cada posição contém uma *TreeMap* com todos os posts dessa data ordenada pelo seu id. A escolha da *HashMap* deveu-se ao elevado número de interrogações que envolviam intervalos de tempo, para estas tornou-se imperativo evitar percorrer os posts que não tivessem incluídos nesse intervalo. Uma vez reduzida a procura, foi necessário organizar a *TreeMap* pelo o id do post, dado que a maioria das interrogações se baseiam neste fator.

Para os users optamos por uma *HashMap* organizada pelos ids dos utilizadores, dado que todas as queries deste grupo se baseiam no id. Houve também a necessidade de associar a cada user um *ArrayList* com todos os posts em que participou.

Tendo em conta que para o grupo das tags a procura se baseia no seu nome definimos uma *HashMap* disposta por este.

Para auxiliar estes dois grupos foi fundamental recorrer a dois *ArrayLists* para diminuir o tempo de resolução de interregações que envolviam tops. Um destes incluía os utilizadores estruturados pela sua reputação e o outro engloba os utilizadores ordenados pelo número de posts realizados.

3 Classes

Nesta secção iremos abordar a estrutura do nosso projeto, referindo as classes presentes, os atribuitos e o modo de funcionamento de cada uma.

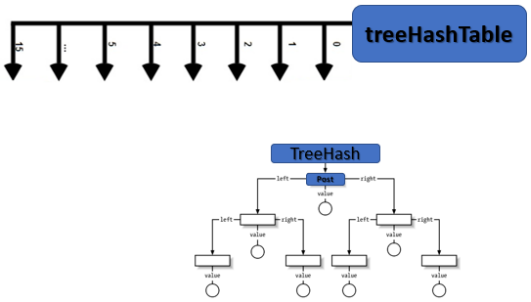
3.1 Parse

Para resolver as queries propostas é necessário realizar o *parsing* que consiste em percorrer cada um dos ficheiros e armazenar na estrutura a informação que consideramos relevante. Desta forma, para fazer o parse dos ficheiros *xml* recorremos à *API* do *Stax*. Numa fase inicial, testamos elaborar o parse utilizando a *API* do *DOM*. No entanto, este não era o método mais adequado, uma vez que carrega toda a informação para a memória.

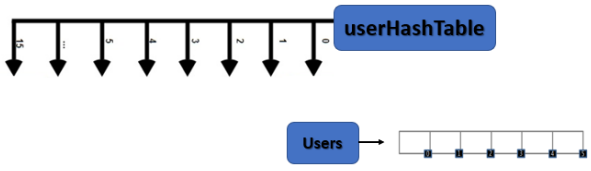
3.2 Struct

Esta classe é fundamental no nosso trabalho, dado que armazena a informação de todos os users, posts e tags. A estrutura principal utilizada inclui três *Maps* relativos aos users, posts e tags, e ainda dois *Lists* úteis nas resolução de determinadas interrogações. É importante salientar que o *Map* da estrutura dos posts inclui em cada posição um *Map* com os posts dessa data. Quanto aos users cada posição da *Map* encontra-se associada a uma *List* com os posts realizados por este.

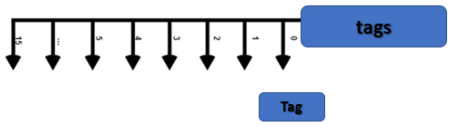
A escolha dos *Maps* deve-se à simples implementação e eficiente procura. Como não será necessário aceder a nenhuma posição diretamente decidimos optar por um *List*. Esta opção deveu-se também à fácil ordenação dos valores.



Na imagem acima encontra-se representada a estrutura dos posts constituída por uma *HashMap* em que cada posição contém uma *TreeMap* com os *Posts* realizados nesse dia.



Nesta, encontra-se esquematizado a estrutura dos users composta por uma *HashMap* com os utilizadores que têm associado um *ArrayList* com alguma informação de todos os seus posts.



Por último, podemos visualizar a estrutura das tags que corresponde a uma *HashMap* com as tags ordenadas pelo seu nome.

3.3 TreeHash

Esta classe contém a data de criação dos posts que se encontram armazenados numa *TreeMap*, bem como dois contadores, um com o número de posts tipo resposta e outro com o número de posts tipo pergunta, que serão úteis para a realização de determinadas queries.

3.3.1 Post

Esta inclui todas as informações relativa aos posts, nomeadamente:

- id - id do post
- postTypeId - tipo do post
- parentId - id da pergunta correspondente
- tag - tags do post
- title - título do post
- ownerUserId - id do criador do post
- answerCount - número de respostas ao post
- commentCount - número de comentários ao post
- score - pontuação do post

3.4 Users

Esta classe contém as informações dos users bem como um *ArrayList* com os posts de cada um, para a resolução de duas das interrogações: *getUserInfo* e *bothParticipated*, e ainda uma variável que conta o número total de posts realizados pelo utilizador.

Deste modo, a classe é constituída por:

- ownerUserId - id do utilizador
- displayName - nome do utilizador
- reputation - reputação do utilizador
- nPosts - número de posts realizados pelo utilizador
- aboutMe - short bio do utilizador
- userList - posts do utilizador (id e data de criação do post)

3.5 Tag

Esta é composta pelas seguintes informação retirada do ficheiro *Tags.xml*:

- id - id da tag
- tagName - nome da tag

3.6 maxMap

Foi imperativo a criação desta classe para a realização da query *mostAnsweredQuestions* para poder ordenar de acordo com o requerido como será explicado na secção das *Queries*. Para tal precisamos das seguintes informações:

- id - id do post
- count - fator de ordenação
- flag - indica se determinada condição se verifica

3.7 maxList

Esta classe foi útil para ordenar os posts de acordo com a data de criação destes. Sendo assim constituída por:

- id - id do post
- creationDate - data de criação do post

3.8 maxPosts

Classe utilizada para ordenar os posts ou utilizadores de acordo com um contador, que varia consoante a situação. Contendo a seguinte informação:

- id - id do post ou do utilizador
- nPosts - contador

3.9 Comparators

Tendo em conta a necessidade de ordenação dos posts e dos users tornou-se imperativo a criação de comparators que indica à estrutura a forma como ordenar a informação. Desta forma, foram criados três comparators:

- maxMapComparator - ordena a informação da classe *maxMap* de acordo com o contador
- DataComparator - ordena a informação de acordo com a data
- maxPostsComparator - ordena a informação da classe *maxPosts* de acordo com o contador

3.10 Queries

3.10.1 Load

Esta interrogação é a mais demorada visto que é responsável por processar a informação dos ficheiros e armazená-la na nossa estrutura. Nesta também procedemos à inicialização da estrutura.

3.10.2 Info_from_post

Começamos por procurar o post na *treeHashTable* e caso este seja do tipo pergunta retiramos-lhe o título e id do utilizador. Com o id acedemos à estrutura dos users e obtemos o nome do utilizador. Na eventualidade do post ser do tipo resposta retiramos o id da pergunta a que corresponde e efetuamos o proceso anterior para este.

3.10.3 Top_most_active

Recorrendo ao *ArrayList topN* que se encontra ordenado pelo número de posts de cada utilizador, extraimos os N ids pedidos e criamos a lista necessária para devolver.

3.10.4 Total_posts

Para esta query percorremos a *treeHashTable* e no caso de a data pertencer ao intervalo de tempo pertendido, somamos aos contadores do número de respostas e número de perguntas respetivamente, e de seguida criamos o par esperado com essa informação.

3.10.5 Questions_with_tag

Iniciamos por percorrer a *treeHashTable* e na hipótese de a data estar contida no intervalo de tempo desejado, vamos a todos os posts e na eventualidade de um ser do tipo pergunta, verificamos se a tag dada se encontra nas tags do post. Em caso afirmativo, guardamos num *ArrayList* o id e a data do post (maxList) para posteriormente ordená-la de acordo com a data e de seguida criamos a lista com os ids dos posts a devolver.

3.10.6 Get_user_info

Nesta interrogação acedemos à posição onde se encontra o utilizador pretendido na *userHashTable* e obtemos a informação do seu perfil e retiramos o *ArrayList* integrante, que posteriormente é ordenado de modo a obter os seus dez últimos posts.

3.10.7 Most_voted_answers

Começamos por percorrer a *treeHashTable* e na possibilidade de a data se enquadar no intervalo de tempo pedido, acedemos a todos os posts e caso seja do tipo resposta inserimos o score e o id no *ArrayList* constituído por maxPosts de modo a ordenar e retirar o top N pretendido.

3.10.8 Most_answered_questions

Nesta interrogação tornou-se imperativo a criação de uma *HashMap constituído por maxMaps*. Para esta query passamos por todas as posições da *treeHashTable* e no caso da data se encontrar no intervalo requerido percorremos os posts verificando se são do tipo pergunta ou resposta. Na eventualidade de ser do tipo pergunta, colocamos a flag a 1 na *Map* indicando que a pergunta se encontra no intervalo pretendido, caso contrário aumentamos ao count (número de respostas ao post). No final, copiamos para uma lista as maxMaps com flag a 1 para posteriormente ordená-las e obter os N pretendidos.

3.10.9 Contains_word

Passamos por todos os Posts existentes e na eventualidade de ser do tipo pergunta verificamos se a palavra dada se encontra no título. Em caso afirmativo, inserimos a *maxList* com o id e a data do post no *ArrayList* de modo a no fim ordenarmos e retirar o top N pretendido.

3.10.10 Both_participated

Começamos por aceder às posições da *userHashTable* dos utilizadores solicitados e retiramos os respetivos *ArrayLists*. Inserimos os ids do maior *ArrayList* noutro *ArrayList*, mas caso o post a inserir seja do tipo resposta, guardamos o id da pergunta correspondente. De seguida, percorremos os elementos do outro *ArrayList* e verificamos se é do tipo pergunta ou tipo resposta. Na eventualidade de ser do tipo pergunta, verificamos se o id deste se encontra no *ArrayList* criado anteriormente. Caso contrário, retiramos o parentId e utilizamos este como fator de comparação.

3.10.11 Better_answer

Nesta query, percorremos a *treeHashTable* e os posts da *TreeHash* até encontrar o post requerido e retiramos o número de respostas existente. Depois voltamos a percorrer a *TreeHashData* e sempre que encontramos um resposta ao post vamos buscar as informações necessárias para obter

a média ponderada e decrementamos ao número de respostas existentes, pois deste modo evitamos continuar a percorrer a estrutura quando já não houverem mais respostas à pergunta. Sempre que encontramos uma média maior do que a guardada até ao momento, substituimos esse valor e atualizamos o id.

3.10.12 Most_used_rep

Iniciamos por preencher um *ArrayList* com os ids dos N utilizadores com melhor reputação. De seguida percorremos a *treeHashTable* e se a data da posição pertencer ao intervalo de tempo, acedemos aos posts da *TreeHash* e para cada um verificamos se o utilizador está incluido no top N estabelecido anteriormente. Caso tal se suceda, para todas as tags desse post vamos verificar se o seu id já se encontra na *HashMap* constituída por *MaxPosts* e caso esteja incrementamos o número de vezes que esta ocorre. Caso contrário, vamos procurá-la à estrutura das tags e inserimos o seu id na nossa *HashMap*. Posteriormente os elementos da *HashMap* são inseridos num *ArrayList* de modo a obtermos as N tags mais usadas pelos N utilizadores com melhor reputação.

3.10.13 Clear

Percorremos todas as estruturas criadas libertando as informações para o qual foi alocada memória.

4 Conclusão

Assim como na primeira fase do projeto, também nesta a escolha das estruturas tiveram influência direta no desempenho das interrogações requeridas. No entanto, a utilização da linguagem Java tornou este resolução muito mais simples uma vez que não tivemos de criar tabelas de hash ou árvores binárias pois o Java contém muitas estruturas pré definidas eficientes.

Na realização desta fase tentamos corrigir as falhas apontadas na fase anterior. Assim, alteramos algumas das nossas estruturas com o objetivo de reduzir o nosso tempo de load sem comprometer significativamente o tempo de resolução das queries. Apesar de este objetivo ter sido atingido consideramos que ainda existem alguns aspetos a melhorar neste projeto.

Em última instância, consideramos que os nossos objetivos neste projeto foram alcançados.