Projeto de Laboratórios de Informática 3
Grupo 4

Daniel Costa (A81302) Carlos Castro (A81946) Luís Macedo (A80494)

Ângelo Sousa (A80813)

Junho 2018

1 Introdução

Este trabalho tem como objetivo o desenvolvimento de um sistema capaz de processar ficheiros XML que armazenam as informações utilizadas pelo *Stack Overflow*. Uma vez processada essa informação, pretende-se que seja possível executar um conjunto de interrogações específico (apresentado posteriormente) de forma eficiente. Esta aplicação foi desenvolvida em Java a pedido da equipa docente.

2 Estruturas de Dados

2.1 dumpSO

Esta estrutura armazena as classes *User*, *Question*, *Answer* e é organizada do seguinte modo: Os posts são guardados num *HashMap postsById* com o par chave/valor, id/*Post*. Esta estrutura foi escolhida pois permite acesso rápido aos *Posts* a partir do id, média O(1). Os *users* são guardados noutro *HashMap usersById* pela mesma razão.

Como é necessário aceder a *posts* em certos intervalos de tempo, estes são guardados em dois *TreeSets* ordenados por data, *questionsByDate* e *answersByDate*, um para as perguntas e outro para respostas. Foi escolhida esta estrutura pois implementa o interface *NavigableSet* e assim tem o método *subSet* que é útil para aceder a intervalos de tempo de maneira eficiente e também implementa *SortedSet*. logo mantém a ordem cronológica.

Os users são também guardados num *ArrayList usersByPosts*, ordenado pelo número de *Posts* publicados.

As tags são guardadas num *HashMap mapTags* cuja chave é a *Tag* e o seu valor é o *Id*.

O *dumpSO* tem métodos associados tais como:

- *getUser*, retorna o *User* dado o seu id;
- *getPost*, retorna o *Post* dado o seu id;
- *getTopUsersByPosts*, retorna os N *Users* com mais *Posts*;
- *getQuestionsByDate*, retorna as *Questions* num intervalo de tempo ordenadas pela data;
- *getAnswersByDate*, retorna as *Answers* num intervalo de tempo ordenadas pela data;
- *getPostsByDate*, retorna os *Posts* num intervalo de tempo ordenadas pela data;
- *getQuestionsWithWord*, retorna todas as *Questions* que contém a palavra;

2.2 User

Esta estrutura armazena os dados sobre um *User* e é organizado do seguinte modo:

- O Id do utilizador é guardado num *long*;
- A reputação é guardada num *int*;
- O nome e a descrição do utilizador é guardado numa *String*;
- Os *Posts* publicados pelo utilizador são guardados num *ArrayList*;

Os *Posts* publicados pelo utilizador são guardados num *ArrayList* de forma a ter rápido acesso. Este não se encontra ordenado, de forma a não gastar tempo, visto que diferentes *querys* ordenam o *ArrayList* por critérios diferentes.

Para alem das variáveis de instância, esta classe contem os métodos:

- Construtores (default, parametrizado e copia);
- getters/setters;
- toString;
- clone;

2.3 Post

A classe *Post* é uma classe abstrata que contem as informações partilhadas pelas *Question* e *Answer*, que são:

- O Id (long) do *Post*;
- Data de criação (LocalDate) do *Post*;
- Id (long) do autor do *Post*;

Para alem das variáveis de instância, esta classe contem os métodos:

- Construtores (default, parametrizado e copia);
- getters/setters;
- equals;
- toString;
- compareTo;

2.4 Question

Esta classe *extends* da classe *Post*, e contem as informações necessárias de uma pergunta.Para tal, as variaveis de instacia são:

- O titulo das pergunta, guardado como uma *String*;
- Um *Set* de *String* que guarda as tags;
- Um *List* de *Long* que guarda os ids das respostas a essa pergunta;

Para alem das variáveis de instância, esta classe contem os métodos:

- Construtores (default, parametrizado e copia);
- getters/setters;
- clone;
- *addAnswer*, que adiciona um id de uma resposta à *List* de respostas;

2.5 Answer

Tal como a *Question*, esta classe *extends* da classe *Post*, e contem as informações necessárias de uma resposta. Assim, as variáveis de instância São:

- O score da resposta, guardado como um *int*;
- O numero de comentários, guardado como um *int*;
- O Id da pergunta correspondente, guardado como um *long*;

Para alem das variáveis de instância, esta classe contem os métodos:

- Construtores (default, parametrizado e copia);
- getters/setters;
- toString;
- clone;
- compareScore, que serve para comparar score de perguntas;

3 Parser

De forma a preencher a estrutura de dado com as informações presentes nos ficheiros XML, foi necessário usar o *SAX parser*.As vantagens de usar este *parser* são:

1. Lê um documento XML de cima para baixo, reconhecendo os *tokens* que compõem um documento XML bem formado;
2. Os *tokens* são processados na mesma ordem em que aparecem no documento;
3. Reporta à aplicação a natureza dos *tokens* que o *parser* encontra, quando os encontra;
4. A aplicação fornece um *event handler* que deve ser registado com o *parser*;
5. Quando os *tokens* são identificados, os métodos *Callback* do *handler* são invocados com as informações relevantes.

4 Estratégias das Interrogações

1. Procura-se no *HashMap postsByID* pelo Id e obtém-se o *Post*. Se o *Post* for uma *Answer*, vai se buscar o id da pergunta correspondente e obtém-se a *Question* da mesma maneira. Depois, obtém-se o título e o nome do autor do *Post* e coloca-se no *Pair* que é devolvido.

2. Retorna os id's dos primeiros N elementos do *ArrayList usersByPosts*, que estão guardados por ordem decrescente de número de *Posts* no *ArrayList usersByRep*.

3. Usando o método *subSet* das *TreeSet's questionsByDate* e *answersByDate*, recebe-se um subconjunto das *Answers* e *Questions* num intervalo de tempo. Depois, calcula-se o tamanho desse subconjunto, usando o método *size* e coloca-se o resultado no *Pair* que é devolvido.

4. Itera-se o resultado do método *subSet* da *TreeSet questionsByDate* e adiciona-se a um *ArrayList* o *Id* da *Question* quando esta pergunta contiver no seu seu *HashSet tags* a *tag*.

5. Procura-se o *User* no *HashSet usersById*. Se não encontrar retorna um *Pair* vazio. Caso contrário, obtém-se o *aboutMe* e *posts* do *User* e ordena-se o *ArrayList posts* por data de criação. É devolvido um *Pair* com o *aboutMe* e os 10 primeiros *posts*.

6. Usando o método *subSet* da *TreeSet answersByDate*, obtém-se o subconjunto das *Answers* no intervalo como um *ArrayList*. Depois, ordena-se pelo número de votos e adiciona-se ao *ArrayList* que é devolvido os ids das N primeiras *Answers*.

7. Usando o método *subSet* da *TreeSet questionsByDate*, obtém-se o subconjunto das *Questions*. Depois, itera-se e conta-se as *Answers* do *ArrayList Answers* da *Question* que tem a data de criação dentro do intervalo dado, inserindo-se num *ArrayList* de *Pair* com as *Questions* e o número de respostas. Depois, ordena-se pelo número de respostas e adiciona-se a um *ArrayList* os ids das primeiras N *Questions*.

8. Itera o *ArrayList questionsByDate* ,que já está ordenado por data, e adiciona a um *ArrayList* a *Question* se o título contiver a palavra dada. Depois, obtém-se o id das *Questions*.

9. Itera o *ArrayList posts* do *User1* e adiciona a *Question* se for uma pergunta ou a *Question* da *Answer* se for uma resposta e adiciona-se a um *HashSet*. Finalmente, itera-se o *ArrayList posts* do *User2* e adiciona-se a um *ArrayList* se a *Question* estiver no *HashSet*.

10. Procura no *HashMap postsById* o *Post* pedido e obtém o *ArrayList* de ids das respostas. Percorre as respostas procurando o id no *postsById* e vai calculando a função (Scr * 0.65) + (Rep * 0.25) + (Cmt * 0.1) e guarda o id da resposta com resultado maior.

11. Adiciona-se a um *HashSet* os ids dos N primeiros elementos do *ArrayList usersByReputation*. Depois, itera-se pelas perguntas do intervalo e verifica-se se o dono da pergunta está no *HashSet* com os N utilizadores com mais reputação. Se sim, as *tags* são adicionadas a um *HashMap* que conta o número de vezes que a *tag* foi usada. Depois, converte-se para um *ArrayList* e ordena-se por ordem decrescente do número de utilizações das *tags*. Por fim, adiciona-se a um *ArrayList* o id das N primeiras *tags*.

5 Decisões de Melhoramento de Desempenho

De forma a melhorar o desempenho, evitou-se o uso de *Stream*, que não é tão bom neste aspeto comparado com alternativas como loops for e iteradores.

Grande porção do tempo de processamento encontra-se no *parser* porque insere em estruturas de dados eficientes. Deste modo, o tempo das *querys* diminui drasticamente.

Nas *querys* que tinham limite máximo de elementos, inicializamos o *ArrayList* com o tamanho máximo, minimizando assim o espaço desperdiçado e melhorando a performance visto que não é preciso redimensionar o *array*.

Na *query 7* é feito uma iteração por todas as *Questions* para contar o número de *Answers* antes de ordenar. Deste modo, evita-se ter de contar novamente em cada comparação feita no *sort*.

Na *query 9* é inserido num *HashSet* as *Questions* do *User* com menos perguntas, de forma a diminuir o número de inserções no *HashSet* que custam mais que procurar no *HashSet*.

Quando se pretendia ordenar uma determinada coleção utilizou-se um *ArrayList* em vez de *TreeSet* porque revelaram melhor performance em tempo de execução. Principalmente, o tempo de inserir e ordenar um *ArrayList* mostrou-se melhor que o tempo de inserir num *TreeSet*. Os gráficos abaixo, mostram como varia o tempo total com o aumento do número de elementos. Como era de esperar, o tempo de inserção a um *ArrayList* (no gráfico a verde) é significativamente melhor do que a um *TreeSet* (no gráfico a vermelho). O tempo que demora a iterar um *ArrayList* com N elementos também é inferior ao tempo de um *TreeSet*. Por fim, o tempo de inserir, ordenar e iterar todo o *ArrayList* mostrou-se melhor que o tempo de inserir e iterar os elementos de um *TreeSet* em todos os testes feitos. Deste modo concluímos que usar *ArrayList* nestes casos era a solução indicada.

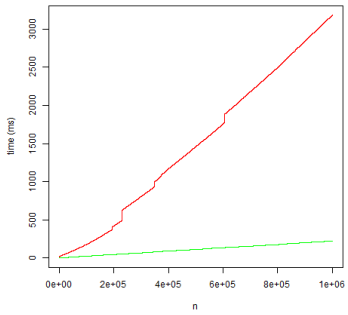

Figura 1: Inserção

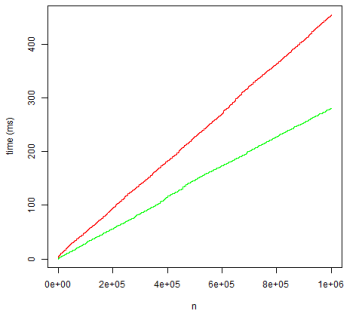

Figura 2: Iteração

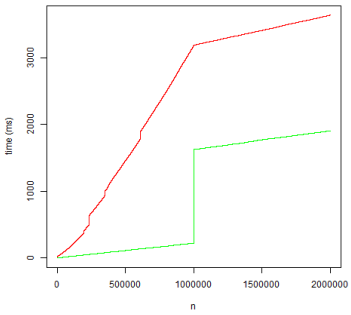

Figura 3: Inserção e iteração