Universidade do Minho
Escola de Engenharia

Wikipédia(Java)

Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III
2º Semestre
2016-2017

A70565 Bruno Manuel Borlido Arieira
A70938 Diogo Meira Neves
A70824 Luís Miguel Bravo Ferraz

11 de Junho de 2017
Braga

Conteúdo

Capítulo 1

Resumo

O trabalho relatado neste documento foi desenvolvido no âmbito da unidade curricular Laboratórios de Informática III de forma a consolidar conhecimentos de JAVA face a um problema de características e dimensões reais. Foi por isso proposto realizar-se um trabalho prático nesta linguagem.

Neste relatório aborda-se a arquitetura de classes escolhida pelo nosso grupo, assim como a enumeração de cada classe e as suas variáveis de instância, a análise mais detalhada das estruturas de dadas utilizadas e os resultados dos tempo de execução das queries pedidas.

Capítulo 2

Introdução

Este projeto consiste na construção de um sistema que permita analisar os artigos presentes em backups da Wikipédia, realizados em diferentes meses, e na extração de informação útil destes.

Cada um destes backups está num ficheiro.xml, disponibilizado pela equipa docente, e para cada um deles o programa deve fazer o parse das informação útil. Deste modo, para guardar os dados de modo a garantir o encapsulamento dos mesmos, usaram-se quatro classes: Contribuidor, GrupoArtigo,Revisao e QueryEngineImpl. Depois dos ficheiros serem percorridos e os dados serem carregados para as respetivas estruturas, desenvolveram se as Queries propostas no enunciado do projeto prático. Para responder às diferentes Queries foram utilizados os métodos presentes nas classes já referidas e alguns métodos presentes na API do java, de modo a garantir que os tempos de execução destas fossem otimizados.

Capítulo 3

Arquitetura das classes

Mostramos como concebemos o nosso programa e as respetivas relações existentes entre as diferentes classes.

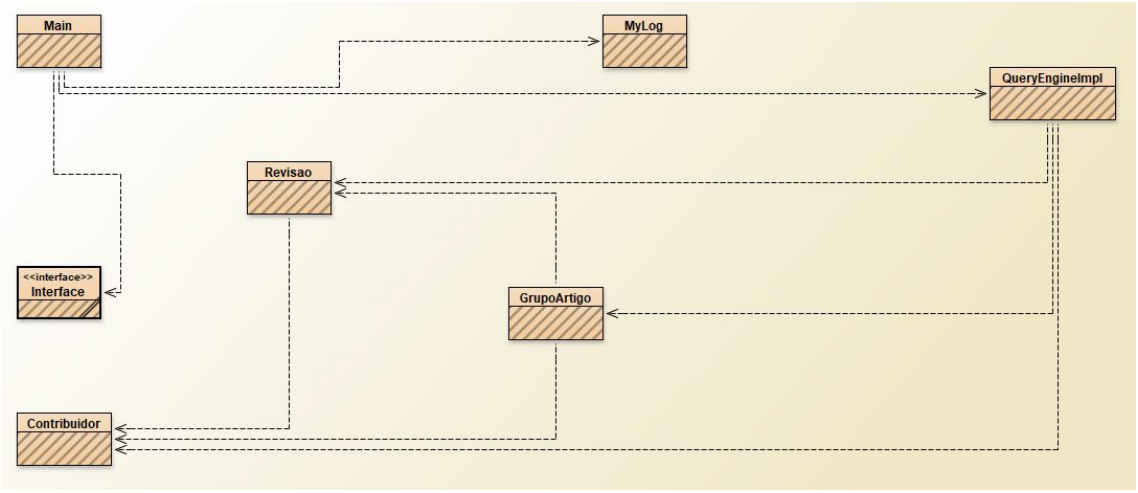

Figura 3.1: Representação da arquitetura.

Capítulo 4

Classes

4.1 Contribuidor

`public class Contribuidor{}`

4.1.1 Variáveis de Instância

- `private long idcontribuidor;`
- `private String nomecontribuidor;`

Esta classe contém a informação relativa a um contribuidor, tal como o nome sugere. As variáveis de instância apresentadas, representam o id do contribuidor e ao seu nome, respetivamente.

4.2 Revisao

`public class Revisao{}`

4.2.1 Variáveis de Instância

- `private String titulo;`
- `private String timestamp;`
- `private long idartigo;`
- `private long numpalavras;`
- `private long nbytes;`

Esta classe contém a informação relativa a uma revisao, tal como o nome sugere. As variáveis de instância apresentadas, representam o titulo do artigo a que a revisão se refere,ao contúdo da tag "timestamp"do xml, ao id do artigo a que a revisao pertence, ao número de palavras que esta contém, e ao número de bytes que o conteúdo da tag "text"ocupa, respetivamente.Decidimos criar as variáveis "nbytes"e "numpalavras"de forma a evitar carregar o texto da revisão para a memória, uma vez que para a realização das queries não precisamos de utilizar diretamente o texto.

4.3 GrupoArtigo

`public class GrupoArtigo{}`

4.3.1 Variáveis de Instância

- `private Set<Long>g_artigo;`
- `private Map<Long,Revisao>g_revisao;`
- `private Map<Long,Contribuidor>g_contribuidor;`
- `private long nartigos;`

Esta classe contém informação relativa ao conjunto de artigos analisados. As variáveis de instância apresentadas, representam o conjunto dos id's dos artigos, o conjunto de toda a informaçao,que achamos necessária para a realização das queries, presente nas revisões dos artigos(chave de procura "idrevisao"), e o conjunto de todos as revisoes dos artigos, associadas a quem a fez, ou seja, cada "idrevisao"(chave de procura) vai ter a informação relativa a um "Contribuidor".

4.4 QueryEngineImpl

`public class QueryEngineImpl implements Interface {}`

4.4.1 Variáveis de Instância

- `private GrupoArtigo grupo;`

4.4.2 Assinatura dos Métodos

- `public GrupoArtigo ParseDoc(String args) {}`

  Este é o método que retira o conteúdo dos campos do xml, e insere na estrutura. Optamos por seguir o modelo Stax, uma vez que este, faz o parse sem precisarmos de aceder à informação toda em simultâneo, enquanto que o DOM carrega logo a informação toda para a memória, o que com o tamanho so snapshots que estamos a analisar nos traria problemas de memória. Na realização deste método tivemos de ter alguns cuidados, como por exemplo, a criação de variáveis auxiliares e a utilização do método "clone()"(de modo a garantir o encapsulamento dos dados), também verificamos que certos artigos não continham os campos "username"e respetivo "id", e por isso também tivemos de fazer essa verificação.De forma a otimizar os tempos de execução, também decidimos não extrair o texto, porque as queries nao pediam nada que obrigasse a ter o texto na memória. Então decidimos contar o número de palavras do texto(recorrendo ao método auxiliar "contaPal()") e o número de bytes que o texto ocupa(recorrendo ao método "getBytes()"da API do java). Depois tivemos de fazer as inserções na estrutura.

- `void init();`

  Método que inicia a estrutura, ou seja, esta está vazia, apenas foi alocado o espaço para receber os dados.

- `void load(int nsnaps, ArrayList<String> snaps_paths);`

  Método que carrega os dados de vários snapshots para a estrutura. Este método utiliza o "parseDoc", ou seja, o método "parseDoc"é chamado "nsnaps"vezes. Os snapshots são passados através do ArrayList "snaps_paths[]".

- `long all_articles();`

  Método que conta todos os artigos que foram analisados,contando com as revisoes, ou seja,fazemos o "getNartigos()", presente na classe GrupoArtigo, que nos devolve um long com o resultado que pretendemos.

- `long unique_articles();`

  Método que conta todos os artigos analisados, mas apenas pelo idartido, ou seja não inclui as revisões. Para isso fazemos o "getg_artigo()", presente na classe GrupoArtigo, que nos devolve um set,ou seja elimina os repetidos, com os "idartigo"de todos os artigos analisados, depois usamos o método "size()"presente na API do java obtemos o resultado que pretendemos.

- `long all_revisions();`

  Método que conta todas as revisões, para isso fazemos o "getg_revisao"e obtemos um map que contém todas as revisoes e respetivos conteúdos, depois utilizamos métodos "stream()"e "count()"da API do java 8.

- `ArrayList<Long> top_10_contributors();`

  Método que irá retornar os 10 contribuidores com mais contribuições. Para a resolução desta querie começamos por passar todos os ids de contribuidores encontrados nas revisões para um ArrayList. De seguida, passamos esses mesmos ids para um HashSet de modo a que so ficassemos com uma ocorrencia de cada id, visto que o set não permite elementos repetidos. O proximo passo passou por contar todas as ocorrencias de cada id, para isso utilizamos o hash que so continha uma ocorrencia de cada um e contamos essa quantas vezes aparecia no array com todos, resultados esses que foram colocados num TreeMap, com o id e o numero de ocorrencias. Por fim foi necessario analisar os 10 com mais contribuições.

- `String contributor_name(long contributor_id);`

  Método que devolve o nome do contribuidor correspondente ao id passado como parametro. Para a resolucao da querie, iteramos o map "g_contribuidor"e vamos verificando se o id passado como parametro é igual ao id de alguma das entradas do map, quando for igual fazemos um break de modo a não ter de correr o map todo.

- `public ArrayList<Long> top_20_largest_articles() {}`

  Método que retorna um ArrayList com os ids dos 20 artigos com o maior número de bytes. Para a realização desta função criamos um comparator em que o critério de ordenação é do maior número de bytes para o menor.Depois criamos um List "temp", de revisoes, auxiliar para armazenar as revisoes de forma decrescente, utilizamos o método "sorted()"da API do java 8. Depois criamos outra variavel auxiliar para guardar os "idartigo"das revisoes, para isso percorremos a variavel "tmp"e obtivemos um ArrayList com os "idartigo"correspondentes as revisoes, sem nunca perder a ordem decrescente do numero de bytes, por fim foi só retirar os repetidos, e adicionar os 20 primeiros a outra variavel auxiliar.

- `String article_title(long article_id);`

  Método que devolve o titulo do artigo correspondente ao id do artigo passado como parametro. Para a resolucao da querie, iteramos o map "g_revisao"e vamos verificando se o id passado como parametro é igual ao id de alguma das entradas do map, quando for igual fazemos um break de modo a não ter de correr o map todo.

- `ArrayList<Long> top_N_articles_with_more_words(int n);`

  Método que retorna um ArrayList com os ids dos n artigos com o maior número de palavra. Para a realização desta função criamos um comparator em que o critério d eordenação é do maior número de palavras para o menor. Depois criamos um List "temp", de revisoes ,auxiliar para armazenar as revisoes de forma decrescente, utilizamos o metodo "sorted()"da API do java 8. Depois criamos outra variavel auxiliar para guardar os "idartigo"das revisoes, para isso percorremos a variavel "tmp"e obtivemos um ArrayList com os "idartigo"correspondentes as revisoes, sem nunca perder a ordem decrescente do numero de palavras, por fim foi só retirar os repetidos, e adicionar os "n"primeiros a outra variavel auxiliar.

- `ArrayList<String> titles_with_prefix(String prefix);`

  Metodo que devolve todos os titulos que têm um dado prefixo. A resolução desta querie tornou-se bastante simples. Assim, foi apenas necessário analisar o TreeMap das revisões e usando a função "startsWith", pre-definida, retirar todos aqueles que o prefixo era valido. Tendo em atençao que nao podia haver repetidos na lista pelo o que era necessario antes de inserir algo na lista, verificar que não estava la.

- `String article_timestamp(long article_id, long revision_id);`

  Método que devolve o "timestamp"correspondente ao id do artigo e ao id da revisao passados como parametros. Para a resolucao da querie, iteramos o map "g_revisao"e vamos verificando se os ids passados como parametro são iguais aos ids de alguma das entradas do map, quando os dois forem iguais fazemos um break de modo a não ter de correr o map todo.

- `void clean();`

  Método que limpa a informacao contida nas estruturas criadas, para isso foi apenas necessario utilizar o metodo "clear()"presente na API do java.

Capítulo 5

Estruturas de dados

Nesta fase do relatório apresentamos uma análise mais detalhada das estruturas de dados mais importantes que constituem o nosso trabalho prático, que se encontram na classe "GrupoArtigo".

5.1 Classe GrupoArtigo

5.1.1 Set$\langle Long \rangle g_artigo$

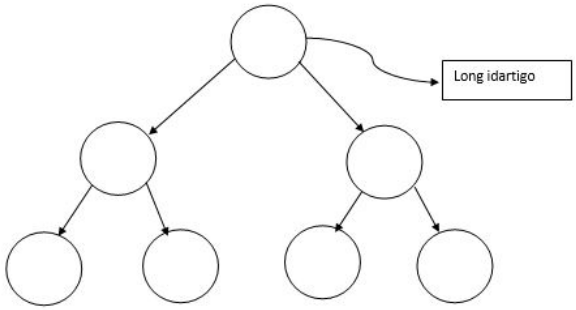

Figura 5.1: Representação da estrutura "g_artigo".

5.1.2 Map$\langle Long, Revisao \rangle g_revisao$

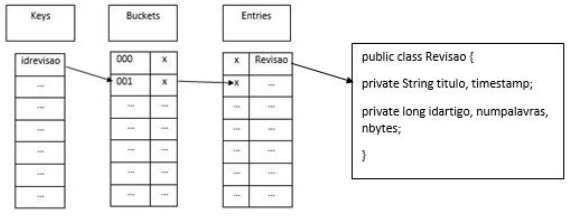

Figura 5.2: Representação da estrutura g_revisao.

5.1.3 **Map**$\langle Long, Contribuidor \rangle g\_contribuidor$

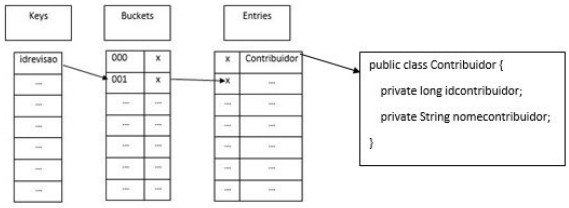

Figura 5.3: Representação da estrutura g_contribuidor.

Capítulo 6

Tempos de execução

Analisando os tempos de execução vemos que as queries que demoram mais tempo, são as queries de ordenação. Também foram as queries que nos causaram mais dificuldades, porque com a nossa estrutura de dados, tendo o "idrevisao"como key dos hashmap's, e uma vez que estas queries pedem "idartigo"tivemos de utilizar algumas variaveis auxiliares, e alguns ciclos o que aumenta os tempos de execução.

```
INIT -> 2 ms
LOAD -> 20373 ms
ALL_ARTICLES -> 28 ms
UNIQUE_ARTICLES -> 22 ms
ALL_REVISIONS -> 62 ms
TOP_10_CONTRIBUTORS -> 443 ms
CONTRIBUTOR_NAME -> 19 ms
TOP_20_LARGEST_ARTICLES -> 457 ms
ARTICLE_TITLE -> 19 ms
TOP_N_ARTICLES_WITH_MORE_WORDS -> 462 ms
TITLES_WITH_PREFIX -> 25 ms
ARTICLE_TIMESTAMP -> 20 ms
CLEAN -> 13 ms
```

Figura 6.1: Tempos de execução das queries.

Capítulo 7

Conclusão

Com este trabalho conseguimos atingir todos os objetivos a que nos propusemos. Uma vez que foi uma transposição direta dum trabalho feito anteriormente (em C), a tarefa ficou facilitada, no entanto foram preciso fazer algumas adaptações devido ás linguagens terem características diferentes. Assim, tivemos de ter em atenção também o encapsulamento, usar as coleções mais adequadas e responder às Queries de forma eficiente. Apesar de termos tido algumas dificuldades, principalmente na definição das estruturas de dados, achamos que as queries tem tempos de execução bastante positivos.Por fim achamos que podiamos ter explorado mais o Java 8, de forma a simplificar algumas queries.