Universidade do Minho
Escola de Engenharia

Laboratórios de Informática III - Relatório do Projeto de Java

Mestrado Integrado em Engenharia Informática

Ana Paula Carvalho (A61855)
Joana Arantes (A57810)
João Pires Barreira (A73831)
Miguel Cunha (A78478)

Ano letivo 2016/2017
Grupo 52

junho 2017

Índice

1 Introdução

No âmbito da unidade curricular de Laboratórios de Informática III, do 2.º ano do Mestrado Integrado em Engenharia Informática da Universidade do Minho, elaboramos este segundo projeto que, à semelhança do projeto anterior feito em C, consiste na construção de um sistema que permita analisar e extrair informação dos artigos presentes em *backups (snapshots)* da Wikipedia, mas desta vez com implementação em Java.

A fonte de dados consiste em três *snapshots* parciais da Wikipedia em inglês, que contêm a versão mais atual de um conjunto de artigos, correspondente aos meses de dezembro de 2016 e de janeiro e fevereiro de 2017. Cada *snapshot* corresponde a um ficheiro distinto no formato XML.

Ao longo deste relatório, explicaremos, com detalhe, a nossa abordagem e solução, bem como a resolução de cada interrogação *(query)* proposta para este projeto.

2 Estruturação das classes

2.1 Contributor

2.1.1 Variáveis de instância

- **id**: Variável responsável por guardar o ID do contribuidor.
- **username**: Variável responsável por guardar o nome do contribuidor.
- **revisions**: Variável inteira responsável por guardar o número de revisões feitas pelo contribuidor.

2.2 Revision

2.2.1 Variáveis de instância

- **id**: Variável responsável por guardar o ID da revisão.
- **timestamp**: Variável responsável por guardar a marca temporal da submissão da revisão.
- **title**: Variável repsonsável por guardar o título do artigo nesta revisão.
- **textSize**: Variável inteira responsável por guardar o tamanho do texto da revisão.
- **wordCount**: Variável inteira responsável por guardar o número de palavras do texto da revisão.

2.3 Article

2.3.1 Variáveis de instância

- **id**: Variável responsável por guardar o ID do artigo.
- **occurences**: Variável inteira responsável por guardar o número de ocorrencências do artigo nas *snapshots revisions*: *ArrayList* das revisões do artigo.

2.4 WikiData

2.4.1 Variáveis de instância

- **artigos**: *HashMap* de pares ID de artigo e artigo.
- **contribuidores**: *HashMap* de pares ID de contribuidor e contribuidor.
- **allArticles**: Variável inteira responsável por guardar o número total de artigos nas *snapshots*.
- **allRevisions**: Variável inteira responsável por guardar o número total de revisões nas *snapshots*.
- **q4**: *ArrayList* que armazena o top 10 de contribuidores com mais revisões.
- **q6**: *ArrayList* que armazena o top 20 de artigos de maior tamanho.

2.4.2 Variáveis globais

- **q4_size**: dimensão do *ArrayList* para o top da query 4.
- **q6_size**: dimensão do *ArrayList* para o top da query 6.

2.5 XMLParser

Para a implementação do *parsing* das *snapshots* da Wikipédia, utilizámos o *parser XML StAX* porque nos pareceu a melhor solução visto ser fácil de usar e ser eficiente em termos de gastos de memória e *CPU*.

No que toca à leitura dos ficheiros, o *parser* cria as variáveis necessárias, procura as *tags* pretendidas, retira-lhes os dados e faz o *set* dos mesmos nas estruturas. Quando encontra o fecho de uma *tag page*, insere o artigo e o contribuidor na WikiData, fazendo-o para o segundo caso apenas quando se trata de uma nova contribuição (i.e. quando *insertArticle* devolve 0).

Por decisão do grupo, são inseridos na mesma os contribuidores sem referência a um ID (ficam com ID = "N/A"), não sendo, no entanto, contabilizados para o cálculo do top (*query* 4).

2.6 Classes dos comparadores

Para além de todas estas classes, existem também as classes *ComparatorContributor*, *ComparatorArticleTextSize* e *ComparatorArticleWordCount* relativas aos comparadores.

3 Escolha das estruturas de dados

A escolha das estruturas de dados utilizadas foi feita de acordo com a análise do que era pedido nas *queries* apresentadas.

Assim, para guardar os artigos e os contribuidores foram usados dois *HashMaps*. Isto deveu-se ao facto de que existem 3 *queries* relativas à procura de um elemento concreto nas estruturas o que é mais eficiente utilizando um *HashMap* (em vez, por exemplo, de uma árvore). O uso de uma árvore ordenada pelo ID não iria trazer nenhum benefício face às *queries* apresentadas. No entanto, a *query 8* poderia ser melhorada utilizando, por exemplo, uma árvore ordenada pelo número de palavras dos artigos.

Para as *queries* 4 e 6, foram utilizadas duas variáveis auxiliares correspondentes a duas listas que são construídas aquando a inserção de um artigo/contribuidor no respetivo *Map*. Desta forma, para responder a estas queries, basta fazer uma iteração dos 10 ou 20 elementos destas listas e ordená-los decrescentemente. Assim, passou-se de uma complexidade O(N) (em que N é o número de artigos/contribuidores no *Map*) para uma complexidade O(n) (em que n é o número de elementos nos tops), o que é substancialmente melhor visto que os tops das queries atuais são de 10 e 20 elementos mas, em qualquer outro caso, são sempre iguais ou inferiores a N.

É de notar, no entanto, que o tamanho destes dois tops pode ser facilmente mudado através das variáveis globais *q4_size* e *q6_size*.

Para as *queries* 1, 2 e 3 foram utilizados três contadores que são incrementados aquando da inserção dos artigos no *Map* e que fazem com que, para responder a estas *queries*, não seja necessário fazer iterações às estruturas de dados nem chamar a função de *size* que apesar de executar em tempo constante (i.e. *O(1)*), para a aplicar é necessário primeiro fazer o get do map dos artigos cuja complexidade é *O(N)* por causa do clone de todos os elementos do *Map*.

4 Queries

4.1 Query 1 - all_articles

Calcula o número total de artigos encontrados, incluindo artigos duplicados e novas revisões de artigos já existentes.

Utiliza, para isso, a variável de instância *allArticles* da classe *WikiData* que é incrementada sempre que, durante o parsing, se faz a inserção de um artigo no map dos artigos (método *insertArticle*).

4.2 Query 2 - unique_articles

Calcula o número de artigos com um ID único encontrados, ou seja, não inclui artigos duplicados nem novas revisões de artigos já existentes.

Utiliza, para isso, a variável de instância *uniqueArticles* da classe *WikiData* que é incrementada sempre que, durante o parsing, se faz a inserção de um artigo cujo ID ainda não estava no map dos artigos (método *insertArticle*).

4.3 Query 3 - all_revisions

Calcula o número total de revisões de artigos encontradas. Inclui quer a versão base de um artigo, quer as restantes revisões feitas ao mesmo, posteriormente.

Utiliza, para isso, a variável de instância *allRevisions* da classe *WikiData* que é incrementada sempre que, durante o parsing, se faz a inserção de um artigo no map dos artigos (método *insertArticle*) cuja revisão ainda não exista no map.

4.4 Query 4 - top_10_contributors

Devolve um *array* com os IDs dos 10 colaboradores que contribuíram para um maior número de revisões de artigos, ordenado decrescentemente. Caso existam colaboradores com um número igual de contribuições, estes aparecem ordenados pelo seu ID, do menor para o maior.

Utiliza, para isso, a variável de instância *q4* da classe *WikiData*. Esta variável corresponde a uma lista com tamanho máximo definido pela constante *q4_size* e é construído sempre que, durante o parsing, se faz uma inserção de um contribuidor no map dos contribuidores (no método *insertContributor*).

4.5 Query 5 - contributor_name

Devolve o *username* de um contribuidor cujo ID é passado como parâmetro. Caso não exista nenhum contribuidor com esse ID, é devolvido o valor *null*.

É feito um *get* com o ID do contribuidor recebido e, caso este exista, aplica-se-lhe o método *getUsername*.

4.6 Query 6 - top_20_largest_articles

Devolve uma lista com os IDs dos 20 artigos que possuem textos com um maior tamanho em bytes, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o maior tamanho de todas as revisões do mesmo. Caso

existam artigos com um tamanho igual, estes aparecem ordenados pelo seu ID, do menor para o maior.

Utiliza, para isso, a variável de instância *q6* da classe *WikiData*. Esta variável corresponde a uma lista com tamanho máximo definido pela constante *q6_size* e é construído sempre que, durante o parsing, se faz uma inserção de um artigo no map dos artigos (no método *insertArticle*).

4.7 Query 7 - article_title

Devolve o título de um artigo cujo ID é passado como parâmetro, sendo apenas considerados os títulos das últimas revisões dos artigos. Caso não exista nenhum artigo com esse ID, é devolvido o valor *null*.

É feito um *get* com o ID do artigo recebido e, caso este exista, vai-se buscar a sua última revisão, aplicando-lhe o método *getTitle*.

4.8 Query 8 - top_N_articles_with_more_words

Devolve uma lista com os IDs dos N artigos que possuem textos com um maior número de palavras, ordenado decrescentemente. Para cada artigo, é apenas contabilizado o texto com o maior número de palavras de todas as revisões do mesmo. Caso existam artigos com um número de palavras igual, estes aparecem ordenados pelo seu ID, do menor para o maior.

Para esta *query* já não é possível utilizar uma variável auxiliar como no caso das *queries* 4 e 6 pois o tamanho do top é variável.

Assim sendo, foi tirado proveito das novas funcionalidades do Java 8 (*streams*), percorrendo todos os artigos da estrutura, ordenando-os utilizando o comparador da classe *ComparatorArticleWordCount*, limitando o resultado a N elementos, covertendo os IDs de strings para longs e criando o um *ArrayList* para o resultado final.

4.9 Query 9 - titles_with_prefix

Devolve uma lista com os títulos dos artigos que começam pelo prefixo passado como parâmetro, sendo apenas considerado o título da última revisão de cada artigo.

Para isso, foi tirado proveito das novas funcionalidades do Java 8 (*streams*), percorrendo todos os artigos da esrutura, retirando-lhes o título da sua última revisão, filtrando os que começam pelo prefixo recebido, ordená-los por ordem alfabética e criando o um *ArrayList* para o resultado final.

4.10 Query 10 - article_timestamp

Devolve o *timestamp* de uma dada revisão de um dado artigo, ambos passados como parâmetro. Caso a revisão e/ou o artigo não existam, é devolvido o valor *null*.

É feito um *get* com o ID do artigo recebido e, caso este exista, vai-se buscar a sua revisão com ID recebido como parâmetro (caso exista), aplicando-lhe o método *getTimestamp*.

5 Modularidade e Encapsulamento dos Dados

5.1 Modularidade do código

Em Java, a modularidade é implícita e garantida através de classes, classes abstractas, interfaces e *packages*. A nossa codificação foi dividida em quatro *packages*: common, engine, li3 e parser.

5.2 Encapsulamento dos dados

O encapsulamento é assegurado através da utilização do método clone.

Por exemplo, nos *getters* é devolvida uma cópia da variável de instância e não uma referência à mesma. Em métodos em que se devolva uma estrutura, é sempre devolvida uma cópia. O mesmo acontece nos métodos em que há uma inserção numa estrutura.

6 Conclusão

Durante a realização deste trabalho, pudemos experienciar as diferenças em codificar exatamente o mesmo programa com as mesmas funcionalidades em duas linguagens diferentes (C e Java). Foi possível notar que, em diversas situações, as estruturas existentes nas bibliotecas de Java facilitaram-nos bastante a abordagem ao problema.

Neste segundo trabalho, procurámos também ter em consideração as conclusões retiradas da parte do projeto em C, nomeadamente na escolha das estruturas, melhorarando e otimizando a nossa solução.

Em retrospetiva, achamos que apresentámos uma solução bastante eficiente, bem documentada e robusta do ponto de vista do encapsulamento dos dados.