Projeto de LI-3

Martins, José(a78821) Costa, Mariana(a78824)
Quaresma, Miguel(a77049)

10 de Junho de 2017

Conteúdo

1 Introdução

A **Wikipédia** é uma das muitas fontes de informação disponíveis na Web, possuindo uma quantidade bastante considerável de dados/informação. No entanto, para que esta informação seja útil é necessário saber processá-la e acima de tudo fazê-lo em tempo útil, recorrendo para isso a estruturas de dados que permitam uma pesquisa rápida mas sem uso excessivo de memória. O objetivo deste projeto é, por isso, implementar uma aplicação (em *Java*) que responde a um conjunto de *queries* relativas a *snapshots* da **Wikipédia** em tempo reduzido e de uma forma correta, recorrendo para isso a um conjunto de recursos já existentes e a estruturas de dados que permitam atingir este objetivo.

2 Desenvolvimento

2.1 Parsing dos snapshots

Os dados sobre os quais incidem as *queries* às quais o projeto responde encontram-se no formato XML sendo por isso necessário processá-los de forma adequada, isto é, retirar a informação útil tendo em conta a sintaxe deste tipo de ficheiro. Para processar ficheiros XML existem diversas API's , sendo que este projeto recorre à StAX para conseguir esta funcionalidade. A StAX permite a leitura dos ficheiros *on-demand*, isto é, ao contrário da DOM que guarda o resultado do *parsing* do ficheiro num objeto árvore em memória, a StAX lê o ficheiro de forma gradual permitindo reduzir a quantidade de memória utilizada. Esta API apresenta no entanto a desvantangem de não permitir o acesso a informação(eventos) que já tenha sido processada logo, caso a mesma seja necessária posteriormente à sua leitura, é preciso guardá-la.

2.2 Classes

Por forma a dividir os dados em duas categorias principais foram criadas duas classes, uma que representa artigos e outra que representa contribuidores, podendo assim instanciar tanto artigos como contribuidores de forma independente.

2.2.1 Article.java

A classe `Article`, seguindo a linha do projeto anterior, e de modo a responder às *queries* propostas, possui as seguintes variáveis de instância:

```
private long id; //id do artigo
private String title; //título do artigo
private Map<Long,String> revisions; // revisões do artigo
                                    // chave: id da revisão
                                    // valor: timestamp da revisão
private long len; //das revisões, o número de caracteres
                  //da revisão com mais caracteres
private long words; //das revisões, o número de palavras
                    //da revisão com mais palavras
```

Nesta classe, para além dos métodos usuais, é de destacar o método `public void setNewLenghtWords(String text)` visto ser este o método que calcula o número de palavras e caractéres de uma `String` passada como argumento e, para os valores sejam maiores que os presentes nas variáveis de instância `len` e `words`, atualiza-os.

2.2.2 Contributor.java

A classe `Contributor`, como a classe `Article`, segue as directrizes do trabalho anterior e, como tal, possui as seguintes variáveis de instância:

```
private long id; //id do contribuidor
private String name; //nome do contribuidor
private int nRev; // número de revisões do contribuidor
```

Esta classe implementa apenas os métodos usuais (`gets, sets, clone`, etc).

2.2.3 Comparadores

Para além das classes já referidas foram ainda implementados três comparadores necessários à implementação de algumas *queries*, sendo esses comparadores os seguintes:

`ArtCompareText.java` : compara dois artigos devolvendo -1 caso o 1º argumento tenha mais caractéres ou caso tenha o mesmo número de caracteres que o 2º argumento, mas tenha um menor id, caso contrário devolve 1; usado na query `top_20_largest_articles`;

`ArtCompareWords.java` : compara dois artigos devolvendo -1 caso o 1º argumento tenha mais palavras ou caso tenha o mesmo número de palavras que o 2º argumento, mas tenha um menor id, caso contrário devolve 1; usado na query `top_N_articles_with_more_words`;

`ComparatorContributorRevs.java` : compara dois contribuidores devolvendo -1 caso o 1º argumento tenha mais revisões ou caso tenha o mesmo número de revisões que o 2º argumento, mas tenha um menor id, caso contrário devolve 1; usado na query `top_10_contributors`;

2.2.4 QueryEngineImpl.java

Possui um `HashMap` de artigos e um `TreeMap` de contribuintes de modo a agregar artigos num conjunto e contribuidores noutro, sobre os quais podemos invocar métodos. Esta classe possui ainda três `longs` de modo a guardar o número de artigos únicos(artUn),a rtigos totais(artTot), e o número de revisões total(totRev). É nesta classe que é implementado o processamento dos *snapshots* bem como as *queries* propostas. Quanto ao processamento dos *snapshots*, este encontra-se dividido em quatro métodos constituintes:

`load(int, ArrayList<String>)` : percorre a lista de snapshots e chama o método processDoc, tendo o cuidado de tratar as exceções que ocorram;

`processDoc(FileInputStream )` : sempre que encontra uma tag `page` invoca o método `processPage`, calculando também o número de artigos totais e únicos, de acordo com o resultado do método `processPage`;

`processPage(XMLStreamReader )` : responsável por processar artigos novos, ou atualizar artigos, caso estes já tenham sido encontrados em *snapshots* anteriores, caso o artigo em causa ainda não tenha sido encontrado adiciona-o(artigo) ao `HashMap` de seguida invoca o método `processRevision` de modo a ser processada a revisão correspondente, devolve se o artigo é novo ou não;

`processRevision(XMLStreamReader , long )` : responsável por processar a revisão encontrada, caso esta seja uma nova revisão, adiciona o autor da mesma(contribuidor) ao `TreeMap` usado para o efeito, ou incrementa o número de revisões do mesmo caso este já esteja presente, atualiza o número total de revisões e guarda o seu (da revisão) `timestamp` associado ao id, ao Map do artigo em questão.

2.3 Queries

A resposta às três primeiras *queries*(`all_articles, unique_articles, all_revisions`) é imediata visto que o resultado das mesmas se encontra armazenado na três variáveis de instancia anteriormente referidas. A implementação das *queries* `contributor_name, article_title, article_timestamp` é bastante simples, sendo apenas necessário verificar se o artigo/contribuidor/revisão existe e, em caso afirmativo, devolver o valor por meio de métodos da classe implementados para o efeito. As restantes queries utilizam `streams`, como mostramos de seguida:

`top_10_contribuitors` : ordenamos de acordo com o comparador ComparatorContributorRevs, guardando apenas os 10 maiores, após isso, mapeamos contribuitors nos seus id's correspondentes, armazenando o resultado num `ArrayList`;

`top_20_largest_articles` : ordenamos de acordo com o comparador ArtCompareText, guardando os 20 maiores mapeando, de seguida, articles nos seus id's correspondentes, armazenando o resultado num `ArrayList`;

`top_N_articles_with_more_words` : ordenamos de acordo com o comparador ArtCompareWords, guardando os N maiores mapeando, de seguida, articles nos seus id's correspondentes, armazenando o resultado num `ArrayList`;

`titles_with_prefix` : mapeamos articles em titles, filtrando o resultado de modo a ficar apenas aqueles que têm como prefixo a `String` passada

como parametro, ordenando, por ordem alfabética, e armazenando num `Arraylist`.

3 Conclusão

Os tempos obtidos nesta versão do projeto são superiores aos da versão em C. Uma das possíveis causas para este comportamento poderá ser as abstrações providenciadas pela linguagem `Java` que tornam operações aparentemente simples em operações elaboradas, nomeadamente a adição de novos elementos a `HashMap`'s. Por outro lado, aumentos na performance poderiam ser obtidos através do uso de `Parallel Stream`'s, apesar de em certos casos o uso das mesmas, devido à necessidade de sincronização entre *threads*, levar a tempos superiores aos esperados.