Universidade do Minho

# Relatório - Laboratórios de Informática III

MIEI - 2º ano - 2º semestre
Universidade do Minho

## Wikipedia - Trabalho prático nº2 - Java

## Grupo 63


| Ricardo Certo | José Bastos | Guilherme Guerreiro |
|---|---|---|
| A75315 | A74696 | A73860 |


11 de Junho de 2017

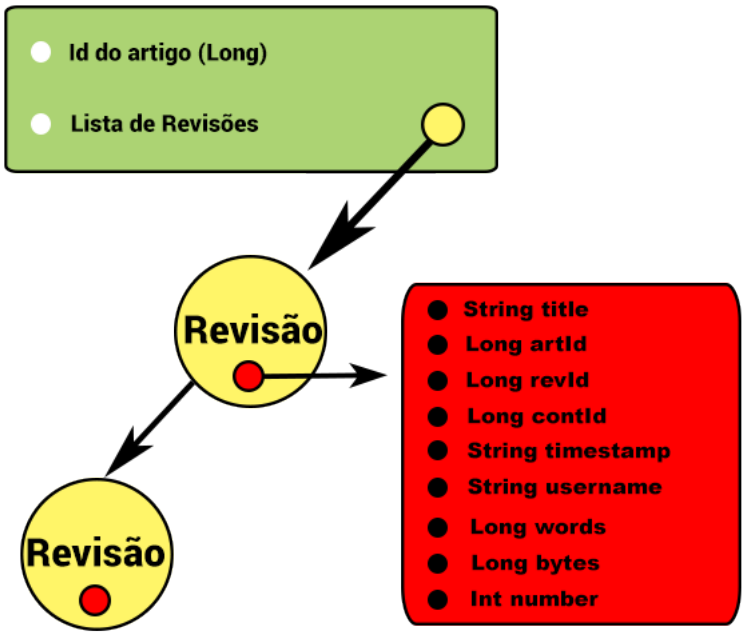
# Conteúdo

# 1 Introdução

Este projeto foi nos solicitado pelos docentes da unidade curricular de Laboratórios de Informática III e tem como principal objetivo a realização de um programa que permita analisar os artigos presentes em backups da Wikipedia, realizados em diferentes meses, e extrair informação útil para esse período de tempo como, por exemplo, o número de revisões, o número de novos artigos, etc. Outros objetivos estão também associados a este, como a consolidação de conhecimentos adquiridos em unidades curriculares anteriores.

A semântica deste trabalho segue a mesma do trabalho realizado numa primeira fase na Linguagem C, agora vai ser implementado na Linguagem Java. Este projeto envolve uma enorme quantidade de dados e uma complexidade estrutural e algoritmica elevada. Os dados que temos de analisar são os snapshots do Wikipedia dos meses de Dezembro , Janeiro , Fevereiro e Março. Antes de respondermos ás interrogações propostas tivemos de realizar um parser aos snapshots e guardar toda a informação relevante numa estrutura desenvolvida por nós que seja a mais adequada para responder ás interrogações. O principal desafio do projecto seria a programação em larga escala, uma vez que pelo nosso programa passam milhares de dados, aumentando assim a complexidade do trabalho. Para que a realização deste projeto fosse possível, foram-nos introduzidos novos princípios de programação, com especial relevo para a Modularidade e encapsulamento de dados.

## 2 Estrutura Adotada




Decidimos fazer uma estrutura relativamente parecido com a da fase anterior, neste caso guardando num **TreeMap** o id do artigo como **Key** e um **ArrayList** como **Value** com todas as revisões relativas ao artigo em questão.
O grupo decidiu usar como estrutura principal um **TreeMap** pois permite guardar ordenadamente os dados pretendidos de forma a facilitar o seu acesso.
No **ArrayList** decidimos guardar todas as revisões relativas ao artigo em questão tendo o cuidado de não guardar repetidos. Para tal, utilizamos uma variável **number** para que, sempre que houver uma revisão repetida, incrementar nessa mesma revisão essa variável. Com isto, temos acesso ao número total de revisões (incluíndo repetidas) e a lista fica mais simplificada, uma vez que não se guarda repetidos.

# 3 Descrição das classes

A arquitetura da aplicação é desenvolvida conforme três classes classes principais: QueryengineImpl , Revision , Parser. A primeira classe referida anteriormente tem como função a implementação das interrogações a que o nosso programa tem de responder. A segunda é responsável por guardar os paramatros que achamos necessários para uma dada revisão. A última classe é responsável por fazer o parser dos snapshots e guarda-los na respetiva estrutura.

## 3.1 Classe QueryEngineImpl

Esta classe é responsável por possuir a nossa estrutura principal que é a seguinte: *TreeMap<Long, List<Revision»* , ou seja , cada id de um artigo vai ter a si associado uma lista de revisões. É também a classe responsável por responder a todas as interrogações colocadas no ínicio do projeto.

```
private TreeMap<Long, List<Revision>> wiki;
private long allArt;
private long allRev;
private String name;
private String title;
private String time;
private Map<Long,Integer> t10;
private Map<Long,Long> t20;
private Map<Long,Long> tN;
```

- TreeMap<Long, List<Revision» wiki - uma estrutura do tipo TreeMap que contém toda a informação dos snapshots, sendo a Key o ID do artigo e os Values as revisions associadas a esse artigo.
- long allArt - variável para representar o número total de artigos disponíveis.
- long allRev - variável para representar o número total de reviões presentes nos snapshots.
- String name - Username do contribuidor. Variável usada na query 5(contributor_name).
- String title - Título do artigo em questão, usava na query 7(article_title).
- String time - Timestamp de um dado artigo de uma dada revisão, usada na query10(article_timestamp).
- Map<Long,Integer> t10 - estrutura que permite contabilizar o número de contribuições, sendo este o seu value, dado um certo contribuidor que é a key. Usada na query4, permitindo saber os 10 users com mais contribuições.
- Map<Long,Long> t20 - estrutura que contém como key o ID do artigo e como value o tamanho do texto do respetivo artigo, em bytes dos 20 artigos que possuem textos com maior texto em bytes. Usada na query6.
- Map<Long,Long> tN - estrutura que contém como key o ID do artigo e como value o número de palavras do texto do respetivo artigo, permitindo saber quais os N artigos que possuem textos com maior número de palavras. Usada na query8.

## 3.2 Classe Revision

A classe Revision tem como objetivo guardar todos os parametros importantes que servem para responder ás interrogações inicialmente colocadas. De seguida podemos observar quais foram os parametros que achamos importantes guardar na nossa estrutura que diz respeito a uma dada revisão.

```
private String title;
private long artId;
private long revId;
private long contId;
private String timestamp;
private String username;
private long words;
private long bytes;
private int number;
```

- String title - título do artigo.
- long artId - ID do artigo
- long revId - ID da revisão
- long contId - ID do contribuidor
- String timestamp - timestamp da revisão
- String username - username do contribuidor
- long words - número de palavras contidas no texto
- long bytes - número de bytes que ocupam o texto
- int number - número de revisões repetidas.

## 3.3 Classe Parser

Esta classe é responsável por interpretar o XML presente nos snapshots e converte-lo na nossa estrutura principal.

```
private TreeMap<Long, List<Revision>> wiki;
```

TreeMap<Long, List<Revision» wiki - a mesma estrutura com a mesma funcionalidade da classe QueryEngineImpl.

# 4 Interrogações

Após escolhermos e implementarmos a nossa estrutra e o parser ter sido feito, passamos agora para o processo de resposta das seguintes interrogações:

```
long all_articles();

long unique_articles();

long all_revisions();

ArrayList<Long> top_10_contributors();

String contributor_name(long contributor_id);

ArrayList<Long> top_20_largest_articles();

String article_title(long article_id);

ArrayList<Long> top_N_articles_with_more_words(int n);

ArrayList<String> titles_with_prefix(String prefix);

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

Antes de realizarmos qualquer tipo de interrogação, temos que garantir que a nossa estrutura foi inicializada, e pelo facto de se tratar de uma linguagem orientada a objetos não é necessário alocar previamente memória para as estruturas, mas apenas inicializa-las da seguinte forma:

```
public void init() {
    wiki = new TreeMap<Long, List<Revision>>();
    t10 = new HashMap<>();
    t20 = new HashMap<>();
    allArt = 0;
    allRev = 0;
    name = "";
    title = "";
    time = "";
}
```

Depois de a estrutura estar inicializada temos necessidade de fazer o carregamento da informação retirada dos snapshots através da função load. De seguida apresentamos o seu respetivo prótótipo.

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

Depois de as estruturas estarem inicializadas e carregadas corretamente com a informação relevante dos snapshots, o programa irá encontrar-se apto para realizar as interrogações colocadas já referidas em cima.

Por último, depois de realizarmos todas as interrogações temos de libertar o espaço utilizado na memoória pela nossa estrutura, sendo que para isso implementamos função clean:

```
public void clean() {
    wiki.clear();
    t10.clear();
    t20.clear();
    tN.clear();
}
```

Não necessitamos de implementar as classes Comparator, por causa do java 8 permitir fazer comparações diretas entre os values de HashMap, que foram as únicas comparações realizadas na elaboração das interrogações.

# 5 Conclusão

Este foi o projeto que até ao momento gerou um maior número de dificuldades a todos elementos do grupo. Ao longo da realização do mesmo fomos confrontados com situações novas que puxaram pelo nosso espírito de autonomia e também desenvolvemos uma maior capacidade avaliativa de quando tomamos uma decisão passando pelos prós e pelos contras da mesma.

Tivemos necessidade de aprender como funciona a libraria do XML de modo a conseguirmos efetuar o parse dos snapshots e guardar as informações úteis nas estruturas referidas acima.

O facto de haver necessidade de utilizar tipos opacos e aceder a variáveis de uma dada classe foi também uma das nossas grandes dificuldades, algo novo para nós e que nos obrigou a estruturar o trabalho de maneira completamente distinta do que estavamos habituados até então, como por exemplo a clonagem de uma string/estrutura antes de a retornar.

Para este projeto, foi necessário obter uma melhor compreensão do funcionamento da linguagem java, tanto da sua complexidade em termos de procura como também da melhor forma de obter os resultados eficientemente, nomeadamente a escolha das estruturas a utilizar. Para a procura nas estruturas foram utilizados *Streams* pois permite realizar de uma forma simplificada tarefas complexas, bem como exibir um código mais agradável de se ler. Com este trabalho, o grupo atingiu os objetivos esperados ao realizar de uma forma rígorosa todas as questões pretendidas e estamos satisfeitos com a sua versão final. Em comparação com o projeto em C, este foi de facto mais simples pois a estratégia relativa à construção do código já se encontrava idealizada e a procura nas estruturas em java é bastante mais simples. Todo este processo de realização do projeto serviu para o grupo consolidar matéria lecionada em outras unidades curriculares e ganhar uma maior experiência na realização de trabalhos em equipa.