

Java Projeto de Laboratórios de Informática III

Grupo 59

João Nunes (A82300)

Luís Braga (A82088)

Luís Martins (A82298)

7 de Maio de 2018



Universidade do Minho

Conteúdo

1	Introdução	2
2	Ficheiros	3
3	Estrutura adotada	3
4	Descrição das classes	4
4.1	Classe Data	5
4.2	Classe Post	5
4.3	Classe Tag	5
4.4	Classe Users	6
4.5	Classe Pergunta	6
4.6	Classe Resposta	6
4.7	Classe TCDEExample	7
5	Queries	7
5.1	Query 1	7
5.2	Query 2	7
5.3	Query 3	7
5.4	Query 4	8
5.5	Query 5	8
5.6	Query 6	8
5.7	Query 7	8
5.8	Query 8	8
5.9	Query 9	8
5.10	Query 10	9
5.11	Query 11	9
6	Desempenho	9
7	Conclusão	10

1 Introdução

Este projeto foi 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 dados presentes em dumps do Stack Overflow e extrair a partir daí a informação útil tal como a reputação dos utilizadores, o número de posts, etc., usando como linguagem a linguagem de programação Java.

Outros objetivos estão também associados a este, tais como a consolidação de conhecimentos adquiridos noutras unidades curriculares tais como Programação Orientada aos Objetos e Algoritmos e Complexidade.

A semântica deste trabalho segue a mesma do trabalho realizado numa primeira fase na linguagem C, que agora será implementado na linguagem Java. Este trabalho envolve uma quantidade de dados e uma complexidade estrutural e algorítmica elevada. Antes de serem respondidas as interrogações propostas foi necessário realizar um parser dos dumps, e guardar toda a informação pertinente numa estrutura desenvolvida pelo grupo que seja a mais adequada a responder as queries.

O principal desafio deste projeto seria a programação em larga escala, uma vez que o trabalho passa por milhares de dados aumentando assim a complexidade do trabalho. Para a realização deste projeto foram-nos introduzidos novos princípios de programação, com especial relevo para a Modularidade e o encapsulamento de dados.

2 Ficheiros

Os ficheiros que usamos para obter a informação e guardar o seu respetivo conteúdo foram os três dumps relativos ao Posts ao Users e ao Tags.

Os três dumps continham informações relevantes para a eventual resposta de cada uma das queries.

Como estes ficheiros estavam escritos em XML, levou-nos a criar um parser de modo a poder "traduzir" e guardar a informação relevante nas nossas estruturas.

3 Estrutura adotada

```
private HashMap<Long,Users> users;  
private HashMap<Long,Post> posts;  
private HashMap<String,Long> tags;  
private HashMap<Long,List<Data>> resPer;  
private List<Data> datas;
```

Foi decidido pelo grupo fazer uma estrutura relativamente parecida com a da fase anterior, sendo guardado em vários **HashMaps** todas as informações pertinentes da informação retirada dos vários ficheiros bem como algumas estruturas auxiliares.

Dentro da HashMap, como **Key** possui o id (post/user) e como **Values** guarda as informações relativas aos Users, aos Posts, as Tags e até uma lista com datas onde tem-se garantidamente as repostas ligadas as suas perguntas, ordenadas por cronologia inversa.

A estrutura principal foi idealizada como sendo uma HashMap pois esta permite guardar os dados pretendidos de forma a facilitar o seu acesso.

Na estrutura secundária foi criada uma lista com todas as datas processadas no sistema, para depois serem associadas.

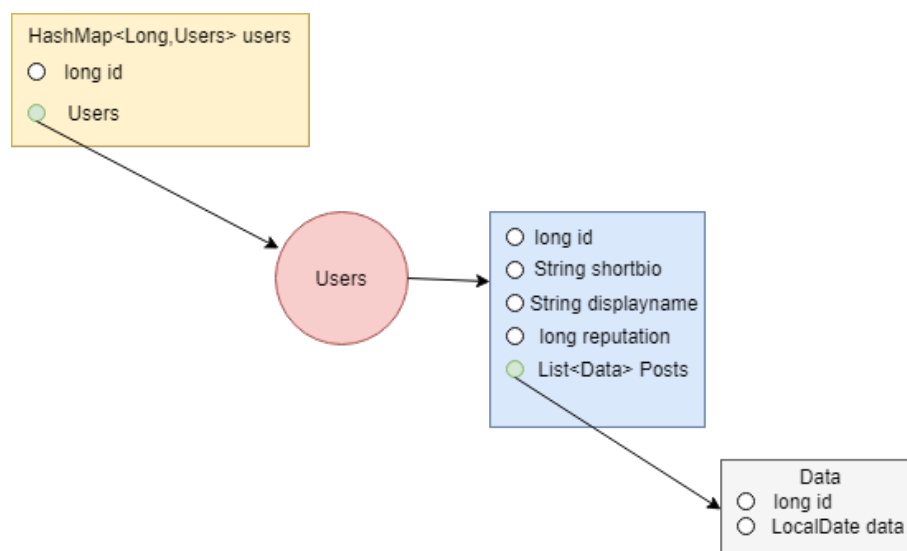


Figura 1: Diagrama do funcionamento da estrutura users

Nesta estrutura users temos um HashMap onde a **Key** é o id do user, de seguida existe um apontador para a estrutura Users como **Values**, dentro desta estrutura é guardado o id do utilizador na forma de um long, o shortbio do utilizador e o display name do utilizador na forma de String, bem como a reputação deste mesmo e uma lista com os ids dos posts do utilizador

ordenados por cronologia inversa, portanto é guardado um long id que corresponde ao id do post do utilizador e um `LocalDate` data, sendo esta a data de criação do post, encontrando-se estes por cronologia inversa porque este é o standard pedido ao longo do trabalho, sendo que as inserções nesta lista são simplificadas por consequência.

As duas estruturas seguintes apresentam um comportamento um algo semelhante a esta primeira. Na estrutura `posts`, tem-se um `HashMap` onde a **Key** é o ID do post e o campo **Values** é preenchido com um apontador para a classe `Post`, sendo que neste campo irão ser guardados um long id do post que irá servir como chave, um `String` `userid` com a identificação do user e um `LocalDate` data com a data de criação do post. Na estrutura `tags` a **Key** é a `String` da tag e o **Values** é um long com o id da tag, de modo a ser melhorada a procura nesta estrutura uma vez que depois é mais fácil serem efetuados os gets.

Na estrutura `resPer` estão todas as perguntas disponíveis nos ficheiros ligadas as respostas. Para isto ser passível de acontecer, no parse é efetuada a distinção entre perguntas e respostas através do `PostTypeId`, quando este é 1, é uma pergunta, caso contrário é uma resposta, sendo que é neste momento que a estrutura `resPer` possui significância. A **Key** é precisamente um long id que corresponde a um parent id, ou seja, o id da pergunta, sendo através de comparações que são ligadas as respostas (`PostTypeId = 2`) às perguntas, as quais respondem criando para tal uma lista, na qual vão ser armazenados o id da resposta e a data na qual essa mesma resposta foi criada. Essa lista `Data` corresponde ao campo **Values**, que é uma lista de datas com as respostas correspondentes aquela pergunta. Esta estrutura foi criada com o intuito de simplificar a resposta as queries.

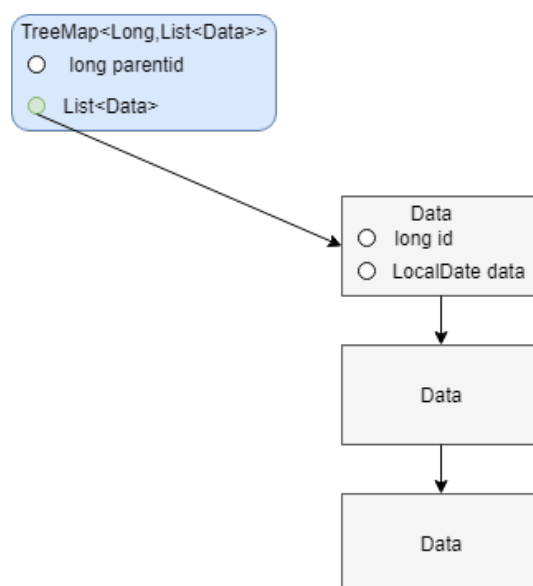


Figura 2: Diagrama do funcionamento da estrutura `resPer`

Por fim, a última estrutura principal adotada foi uma lista de `Data`, chamada `datas`, que tem importância nas queries que pedem posts por cronologia inversa visto que esta estrutura contém os ids dos posts todos e as suas datas de criação ordenadas por cronologia inversa.

4 Descrição das classes

A arquitetura da aplicação é desenvolvida conforme sete classes principais: `Data`, `Post`, `Tag`, `Users`, `Pergunta`, `Resposta` e `TCDEExample`. A primeira classe tem como função auxiliar na ligação das respostas às perguntas, bem como na ligação da lista de posts a um dado utilizado. A segunda

classe é responsável por guardar os parâmetros indispensáveis referentes ao ficheiro *Users.xml*. A terceira e a quarta classe guardam os parâmetros fundamentais referentes aos ficheiros *Posts.xml* e *Users.xml* respetivamente. A quinta e a sexta classe têm como objetivo diferenciar os dois tipos de post, segundo o `PostTypeId`, ou seja, diferenciar segundo pergunta ou resposta, sendo, obviamente, guardados campos diferentes e imprescindíveis para a eventual resposta as queries. É de notar que estas duas classes estendem para a classe `Post`, herdando portanto todos os métodos e variáveis de instância do `Post`, acrescentando os seus próprios. A última classe destina-se a efetuar o parse dos ficheiros, guardar nas estruturas e também é feita a implementação das queries, a qual o programa tem que responder.

4.1 Classe Data

```
private long id;  
private LocalDate data;
```

Esta é a classe responsável por possuir os parâmetros necessários para, por exemplo, guardar o id de um post e a sua data de criação.

- `long id`
O id que irá ser guardado na forma de um long.
- `LocalDate data`
A data de criação que é guardada na forma de um `LocalDate`.

4.2 Classe Post

```
private long id;  
private String userid;  
private LocalDate data;
```

Esta é a classe responsável por possuir os dados referentes ao posts que depois irão ser usados para responder às queries.

- `long id`
O id do post guardado num long.
- `String userid`
O user id guardado numa string.
- `LocalDate data`
A data de criação do post em `LocalDate`.

4.3 Classe Tag

```
private String tagName;  
private long tagId;
```

Esta classe é responsável por possuir os dados referentes as tags e é principalmente utilizada na resposta a query 11.

- `String tagName`
O nome da tag guardado como `String`.
- `long tagId`
O número de ocorrências da certa tag.

4.4 Classe Users

```
private long id;  
private String shortbio;  
private String displayname;  
private long reputation;  
private List<Data> posts;
```

Esta classe possui os dados referentes aos users que depois irão ser usados para responder às queries. Possui ainda uma lista com os posts associados aos users ordenados por cronologia inversa de modo a auxiliar, novamente, na resposta às queries.

- `long id`
O id do user.
- `String shortbio`
A shortbio do utilizador.
- `String displayname`
O display name do utilizador.
- `long reputation`
A reputação do utilizador.
- `List<Data> posts`
A lista dos posts do utilizador.

4.5 Classe Pergunta

```
private String titulo;  
private String tags;
```

Classe que estende para a classe Post. Esta é, portanto, uma sub-classe da classe Post que herda os métodos e variáveis dessa mesma classe e acrescenta informação referente a uma pergunta, criada com o intuito de diferenciar os posts.

- `String titulo`
O título do post (pergunta).
- `String tags`
As tags associadas à pergunta.

4.6 Classe Resposta

```
private long parentid;  
private long scr;  
private long ncom;
```

Classe que tal como a pergunta, estende para a classe Post e adiciona as suas próprias variáveis de instância e métodos, acrescentando a informação referente a um post do tipo resposta.

- `long parentid`
O id do post ao qual a resposta responde.
- `long scr`
A pontuação da resposta.
- `long ncom`
O número de comentários.

4.7 Classe TCDEExample

```
private HashMap<Long,Users> users;  
private HashMap<Long,Post> posts;  
private HashMap<String,Long> tags;  
private HashMap<Long,List<Data>> resPer;  
private List<Data> datas;
```

Classe principal do programa, onde se situa as principais estruturas de dados usadas, sendo que também é nesta classe onde é efetuado o parse dos ficheiros de modo a popular estas estruturas de dados. A resposta às queries também é efetuada nesta classe.

- **HashMap<Long,User> users**

Estrutura do tipo HashMap que possui toda a informação referente aos Users, sendo a Key o id do user e o Values a classe User.

- **HashMap<Long,Post> posts**

Estrutura HashMap que possui toda a informação referente aos Posts, sendo a Key o id do post e o Values a classe Post.

- **HashMap<String,Long> tags**

Estrutura HashMap que possui toda a informação referente as Tags, sendo a Key a String da Tag e o Values o id da tag.

- **HashMap<Long,List<Data>> resPer**

Estrutura HashMap que possui uma lista com as respostas ligadas a pergunta, sendo o Key o parent id e o Values uma lista com as respostas a uma dada pergunta.

- **List<Data> datas**

Estrutura auxiliar de modo a guardar os ids dos posts e as suas datas de criação, ordenado por cronologia inversa.

5 Queries

5.1 Query 1

Nesta query a estratégia utilizada foi, primeiramente, verificar se na estrutura posts existe algum post com o dado id, caso isto se confirme é retirado esse post com esse dado id, de seguida é verificado se se trata de uma pergunta, caso seja uma pergunta é retirado o título e o display name do utilizador que fez essa pergunta sendo colocado num Pair o resultado final.

5.2 Query 2

Na query 2 foi utilizado um stream onde o HashMap do users é ordenado de acordo com o número de posts, limitando-se depois este aos N mais ativos. De seguida é feito um map desses N mais ativos, sendo retirado o id deles e depois transformado numa lista.

5.3 Query 3

Em primeiro lugar, nesta query é feita uma lista com os posts dentro desse intervalo de tempo com o auxílio de streams, sucedendo-se para cada Data d pertencente a essa lista de posts nesse intervalo. Se o post for uma resposta é incrementada a variável associada ao número de respostas, caso contrário é incrementada a variável do número de perguntas.

5.4 Query 4

Tal como nas outras queries, irá ser utilizado um único stream, onde é utilizado a estrutura `datas`. Sucessivamente, a lista é filtrada caso esta seja uma pergunta e apenas são considerados os posts que se encontram no intervalo de tempo. De seguida, na lista de posts que contém o id dos posts resultantes do primeiro filter, é feito outro filter onde é verificado se estes, nas suas Tags, possuem a tag passada como argumento. Caso isto se verifique, é efetuado um map ao resultado onde se tira o ID e o resultado é transformado numa lista.

É de referir que ao contrário da primeira parte do projeto, a formatação das datas nesta segunda parte foi feita até ao dia, ou seja do seguinte modo: *"yyyy-MM-dd"*.

5.5 Query 5

Aqui, antes de mais, é verificado se o `HashMap` dos users contém o long id passado nesta query, verificando-se tal, é retirado e guardado o `shortbio` desse utilizador. Sucessivamente, acede à lista dos posts deste utilizador utilizando a estrutura `List<Data>`, onde, como esta já se encontra ordenada por cronologia inversa, usando um stream, apenas é necessário fazer um map para retirar os ids e limitar aos dez primeiros. De seguida este resultado é transformado numa lista sendo no fim retornado o par com o `shortbio` e os ids dos dez últimos posts.

5.6 Query 6

Nesta, é criada uma lista com os posts no dado intervalo de tempo usando uma stream, com a estrutura posts é feito um filter verificando se não é pergunta e se se encontra no dado intervalo de tempo, em que no caso de se verificar, é feito um map para fazer um clone do post na forma de resposta e é depois transformado numa lista de respostas. Com tudo isto já feito, basta utilizar um outro stream para ordenar a lista de respostas por valor do score sendo esta lista limitada às N respostas. Por fim é apenas necessário fazer uma nova stream de modo a coletar o ID transformando esta lista numa lista de long ids.

5.7 Query 7

Na query 7, desde já, foi considerado pelo grupo o score total das respostas para a resposta a esta query. Em primeiro lugar e tal como na query anterior, é criada uma lista de posts efetuando um stream na estrutura `posts` seguido de um filter verificando se se trata de uma pergunta e se se encontra no intervalo de tempo. Prontamente, é efetuado um clone do post na forma de uma resposta em que é transformado numa lista de respostas por fim. De seguida, para cada post pertencente a esta lista é retirado o id e o número de respostas, e caso este id retirado se encontre na lista de respostas ligadas às perguntas (`resPer`), é retirado o tamanho total da lista de datas (efetivamente é o número de respostas a dada pergunta), sendo o resultado adicionado a um par. No desfecho, este par é ordenado comparando os segundos valores sendo este limitado às N perguntas, no fim é transformado numa lista de ids.

5.8 Query 8

Na resposta a esta query foi utilizada uma stream na lista de datas, onde é filtrado caso seja uma pergunta e caso o título desta mesma pergunta possua a palavra passada como argumento, é prontamente limitada às N perguntas e efetuado um map para colectar os ids e no término é transformada numa lista de long.

5.9 Query 9

Aqui, primeiramente, verifica-se se o `HashMap` dos users contém o primeiro id e o segundo id, de seguida são retirados os posts do primeiro e do segundo user e são guardados numa `List<Data>`. É criado também nesta altura um set de datas auxiliares que guardará os ids e as datas deles

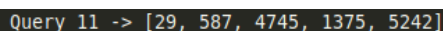
para posteriormente ordená-los. Seguidamente são efetuadas sucessivas comparações, tais como verificar se ambos se tratam de uma resposta e se ambos respondem à mesma pergunta, utilizando um método que retorna o parent id. É depois retirado o parent id da pergunta a qual estes respondem e é adicionada ao set o parent id e a Data, caso isto não se verifique são testados os outros casos. No fim o set é ordenado segundo a data e é limitado as N perguntas, sendo depois retirados os ids e transformada numa lista.

5.10 Query 10

Na primeira instância é verificado se na lista de posts e na lista de repostas ligadas à pergunta se ambas as HashMaps contêm o id passado como argumento. Logo depois é retirada a lista de posts usando como referência a lista resPer, em que para tal cada post é retirado o id e é testado novamente se para cada id retirado dessa lista se estes de facto existem no HashMap dos posts. Em caso positivo, é retirada a resposta e o user id, seguindo-se o cálculo da melhor resposta usando a função de média ponderada disponibilizada, guardando o id, para este no fim ser retornado.

5.11 Query 11

Na query 11, é criada uma lista com os N users com mais reputação de sempre, usando para isto uma stream no HashMap users onde este é ordenado segundo a reputação limitado aos N sendo transformada numa lista no final. De seguida é criada uma lista de datas que contêm o id e data de publicação de todos os posts no intervalo do N users com maior reputação, utilizando novamente uma stream, agora de modo a procedermos à partição das tags de modo a ser possibilitada a comparação. Foi usado um bloco for onde se guarda a substring e verifica-se o index »"que representa o fim da tag, sendo adicionados à lista de tags todas as tags dos posts calculados anteriormente. Por fim, é verificado se no HashMap das tags existe alguma que é igual à lista de tags calculada. Verificando-se tal, é retirada o id da tag da estrutura tags, sendo depois este adicionado a uma lista de longs, lista esta que possui o resultado final, sucedendo eventualmente ao seu retorno quando chega ao patamar superior (N). É de referir, contudo, que o resultado desta query não corresponde com os resultados referência:



```
Query 11 -> [29, 587, 4745, 1375, 5242]
```

Figura 3: Resultado da implementação do grupo da query 11 com parâmetro 1

Depois da resposta a todas estas queries, é libertado o espaço alocado na memória pela estrutura, usando, para isto, a implementação do método clear.

6 Desempenho

Como o desempenho é um dos principais aspetos a ter em consideração na realização do projeto, foi necessário ter em conta diversos aspetos para alcançar esse objetivo, nomeadamente:

- A estrutura principal usada, ou seja, HashMap, faz com que o tempo de acesso e resposta seja mais rápido, porque como não foi considerada a ordem, não é necessário percorrer os dados todos, e como o número de dados a ser processados é grande, isso faz com que a estrutura escolhida seja ainda mais eficiente.
- As listas de posts, sejam estes os posts do user ou a lista de repostas ligadas às perguntas encontram-se ordenados por cronologia inversa, o que facilita a inserção nestas mesmas estruturas.

- Foi utilizada também a biblioteca StAX do Java na realização do parse o que permitiu com que o ficheiro XML fosse lido por "*streaming*", o que faz com que este não ocupe tanta memória ao ser efetuado este mesmo parse.
- A utilização de streams ao longo das várias respostas às queries permitiu explorar o escalonamento, o que melhora o tempo de execução do programa.

1	LOAD -> 16743 ms
2	Query 1: -> 1 ms
3	Query 2 -> 205 ms
4	Query 3 -> 43 ms
5	Query 4 -> 255 ms
6	Query 5 -> 1 ms
7	Query6 -> 115 ms
8	Query 7 -> 141 ms
9	Query 8 -> 16 ms
10	Query9 -> 25 ms
11	Query 10 -> 0 ms
12	Query 11 -> 91 ms
13	CLEAN -> 21 ms

Figura 4: Tempo de execução das queries bem como o load e o clean

7 Conclusão

Ao longo da realização do projeto, o grupo foi confrontado com situações novas que puxaram pelo espírito de autonomia e também permitiu desenvolver um maior espírito auto crítico, passando pelos prós e contras de uma decisão.

Todas as estruturas usadas na realização do projeto requeriam um conhecimento prévio de conceitos abordados noutras unidades curriculares.

Foi imprescindível, também, aprender o funcionamento da biblioteca do XML, de efetuar o parse das dumps e guardar as informações úteis em estruturas.

Para esta fase do 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 na escolha das estruturas a utilizar. Para a procura nas estruturas foram utilizadas streams, pois permite, de uma forma simplificada, realizar tarefas complexas, bem como melhorar o desempenho do código.

Nesta parte do trabalho, o grupo conseguiu atingir todos os objetivos esperados, ao realizar todas as questões pretendidas, ao contrário de na primeira parte, que falhou em dar a resposta à query 11. 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 em estruturas Java é bastante simplificada, sendo que todo este processo de realização do projeto serviu mai suma vez para o grupo consolidar matéria lecionada em outras unidades curriculares, bem como ganhar uma maior experiência na realização de trabalhos de grupo.