Universidade do Minho

Mestrado Integrado em engenharia informática

2020/2021

Laboratórios de Informática III

Grupo 88

Rúben Santos (93257)

João Martins (91669)

Jorge Lima (93183)

Índice

Introdução

O projeto de Java da disciplina de LI3 tem como objetivo fundamental ajudar à consolidação experimental dos conhecimentos teóricos e práticos adquiridos na disciplina de Programação Orientada aos Objetos

Pretende-se a criação de uma aplicação Desktop em Java, baseada na utilização das interfaces e das coleções de JCF, cujo objetivo é a realização de consultas interativas de informações relativas à gestão básica de um sistema de recomendação/classificação de negócios.

Model

O *package* Model do projeto consiste num conjunto de 3 packages (*Businesses*, *Reviews*, *Users*) e de 3 classes (*Stats, Queries* e *Pair*).

Negócios

A **classe Businesses** representa negócios (e suas informações) que constituem o respetivo ficheiro (.csv).

O ficheiro businesses.csv é processado, linha a linha, pelo método **createBusinessFromLine**().

Este método cria uma instância da **classe BusinessInfo**. As variáveis de instância desta classe são apresentadas abaixo:

```
private final String id;
private final String name;
private final String city;
private final String state;
private float ratingSum;    // Used to get monthly average rating.
private int nReviews;
private boolean evaluated;

// List of Pairs: (LocalDate, BusinessMonthlyInfo)
// BusinessMonthlyInfo: #reviews, #Users and Average rating per month.
private final List<Pair<LocalDate, BusinessMonthlyInfo>> monthlyInfo;
```

Esta classe implementa a interface ***Comparable***, sendo que o método **compareTo**() realiza a comparação lexicográfica do estado e, no caso de estes serem iguais, recorre à comparação dos nomes das cidades.

Para cada mês, é mantido um par (*LocalDate*, **BusinessMonthlyInfo**) que guarda informação relativa ao desempenho mensal do respetivo negócio.

A **classe BusinessMonthlyInfo** possui a informação mensal do número de reviews, número de utilizadores diferentes e avaliação média do negócio.

Na **classe Businesses** são mantidos **2** ***HashMaps<String, BusinessInfo>***, um para negócios não avaliados (inicialmente todos) e o outro para os que já o foram.

Utilizadores

A **classe Users** representa utilizadores (e suas informações) que constituem o respetivo ficheiro (.csv).

O ficheiro users.csv é processado, linha a linha, pelo método **addUsrFromLine**().

Este método cria uma instância da **classe UsersInfo**. As variáveis de instância desta classe são apresentadas abaixo:

```
private boolean active;     // User is active? (aka made a review)
private final String name;

// List of Pairs: (LocalDate, UserMonthlyInfo)
// UserMonthlyInfo: #reviews, #Businesses and Average rating per month.
private final List<Pair<LocalDate, UserMonthlyInfo>> monthlyInfo;
```

Para cada mês, é mantido um par (*LocalDate*, **UserMonthlyInfo**) que guarda informação relativa à atividade mensal do respetivo utilizador.

A **classe UserMonthlyInfo** possui a informação mensal do número de reviews, número de negócios diferentes e avaliação média das reviews que o utilizador faz por mês.

Na **classe Users** é mantido um ***HashMap<String, UserInfo>*** para guardar toda a informação relativa aos utilizadores (String=user_id).

Avaliações

A **classe Reviews** representa as avaliações, efetuadas por utilizadores, acerca de negócios, que constituem o respetivo ficheiro (.csv).

O ficheiro reviews.csv é processado, linha a linha, pelo método **processLine**().

Este método cria uma instância da **classe ReviewsInfo**, assim como atualiza informações pertinentes relativas a utilizadores e negócios (nomeadamente o rating de um negócio, a atividade mensal de um utilizador, **etc**). As variáveis de instância desta classe são apresentadas abaixo:

```
private final String rev_id;
private final String usr_id;
private final String bus_id;
private final float rating;
private final LocalDate date;
```

Esta classe implementa a interface ***Comparable***, sendo que o método **compareTo**() recorre ao comparador da classe *LocalDate*.

As reviews são mantidas numa ***List<ReviewsInfo>*** ordenada de acordo com o comparador definido anteriormente (por data).

Consultas Estatísticas

A produção das estatísticas é maioritariamente efetuada utilizando informação presente na **classe Reviews**.

Na **classe Stats** possuímos as seguintes variáveis de instância:

```
private final int incorrectReviews;
private final int numberOfBusinesses;
private final int evaluatedBusinesses;
private final int nonEvaluatedBusinesses;
private final int numberOfUsers;
private final int activeUsers;
private final int inactiveUsers;
private final int noImpactReviews;
private final float averageReviewRating;

private final List<ReviewsMonthlyStats> monthlyStats;
```

A ***List<ReviewsMonthlyStats>*** é criada a partir do método **buildMonthlyStats**() presente na **classe Reviews**.

Esta **classe *ReviewsMonthlyStats*** é responsável por armazenar as estatísticas mensais gerais, nomeadamente a média de classificações e o número total de reviews, assim como o número de utilizadores que efetuaram reviews num dado mês.

Obter esta lista de estatísticas mensais implica percorrer a ***List<ReviewsInfo>*** e, para cada mês criar uma instância da **classe ReviewsMonthlyStats**.

Posteriormente apenas teremos de usar o método **toString**() a cada elemento da lista.

Consultas Interativas

As consultas interativas (*aka queries*) integram a **classe Queries**.

1. Lista ordenada alfabeticamente com os identificadores dos negócios nunca avaliados e o seu respetivo total;

   Os negócios já foram previamente organizados em 2 listas diferentes (avaliados e não avaliados). Basta-nos requisitar a lista pretendida e passar cada um dos identificadores de negócio para um nome de negócio, recorrendo a uma procura no *HashMap* pela informação acerca do negócio.

   T = O(N)

   N é o número de negócios não avaliados.

2. Dado um mês e um ano (válidos), determinar o número total global de reviews realizadas e o número total de users distintos que as realizaram;

Em primeiro lugar, é preciso validar o mês e o ano inseridos pelo utilizador. Logo de seguida, localizar o índice da primeira review do mês/ano inserido na lista de reviews guardada na classe Reviews (de salientar que esta lista está ordenada por data, daí podermos realizar uma procura binária).

A partir daqui apenas temos de percorrer a lista até que encontremos uma review do mês seguinte. Enquanto percorremos este pedaço de lista vamos contando o número de reviews assim como o número de utilizadores diferentes.

T = O(M * log N)

N é o número total de reviews.

M é o número de reviews no mês/ano inserido pelo utilizador.

3. Dado um código de utilizador, determinar, para cada mês, quantas reviews fez, quantos negócios distintos avaliou e que nota média atribuiu;

   Para resolver esta query basta obter a informação relativa a um utilizador, realizando uma procura no ***HashMap<String, UserInfo>*** da **classe Users**.

   Tento esta informação, a resolução da query é dada pelo método **monthlyInfoToString**().

   Esta informação é de acesso em **O(1)**, sendo que apenas temos de a mostrar ao utilizador.

   Nota: Isto é possível pois ao longo da leitura do ficheiro reviews.csv fomos mantendo atualizada a informação dos utilizadores e negócios.

4. Dado o código de um negócio, determinar, mês a mês, quantas vezes foi avaliado, por quantos users diferentes e a média de classificação;

   Esta query segue a mesma metodologia e tem a mesma complexidade temporal do que a classe anterior (iii.)

5. Dado o código de um utilizador determinar a lista de nomes de negócios que mais avaliou (e quantos), ordenada por ordem decrescente de quantidade e, para quantidades iguais, por ordem alfabética dos negócios;

   Começamos por criar um ***Set<String>*** que irá conter os identificadores de cada negócio que o utilizador avaliou. Esta informação é mantida como variável de instância da **classe UsersInfo**, sendo que para criar este Set temos de realizar uma travessia pelos valores de uma lista que contém a informação de cada utilizador organizada por data (***List<Pair<LocalDate, UserMonthlyInfo>>***), inserindo cada *business id* (guardado na instância de **UserMonthlyInfo**).

   De seguida este *Set de id's* é convertido para uma lista de nomes de negócios que é posteriormente ordenada.

   Temos que **T = O(2 * N + N * log N)**, sendo **N** o número de negócios avaliados pelo utilizador.

6. Determinar o conjunto dos X negócios mais avaliados (com mais reviews) em cada ano, indicando o número total de distintos utilizadores que o avaliaram (X é um inteiro dado pelo utilizador);

Para responder a esta questão é criado uma lista de pares, em que o primeiro elemento marca o ano das reviews e o segundo é uma lista que contém, para cada negócio, o número total de utilizadores diferentes que o avaliaram.

É criado um *Comparator* que organiza as listas anteriores por número de reviews.

Para a resolução optamos por utilizar uma PriorityQueue (aka Min Heap) para manter em cada lista apenas os X negócios mais avaliados retirando, sempre que necessário, o negócio menos avaliado (*cabeça da min heap, O (1)*).

Como tal, apenas teremos de fazer uma travessia pela lista de negócios que foram avaliados (e não por todos os negócios, como explicado anteriormente em i.) e ir mantendo a Min Heap atualizada.

Temos que **T = O(N)**, sendo **N** o número de negócios **com reviews**.

7. Determinar, para cada cidade, a lista dos três mais famosos negócios em termos de número de reviews;

   Para responder a esta questão é criado um map, em que a chave é a cidade e o valor é uma lista de pares com o número de reviews e com o nome desse negócio.

   É criado um *Comparator* que organiza as listas anteriores por número de reviews.

   Temos de percorrer todos os negócios e verificar se se qualificam para entrar no top 3 da sua respetiva cidade e para isso temos que manter as listas sempre atualizadas para poder remover sempre o menor elemento.

   Temos que **T = O (N * 3log 3) ~= T = O (N *1.5),** sendo **N** o número de **businesses**.

8. Determinar os códigos dos X utilizadores (sendo X dado pelo utilizador) que avaliaram mais negócios diferentes, indicando quantos, sendo o critério de ordenação a ordem decrescente do número de negócios;

   Para responder a esta questão é criado uma *PriorityQueue* de uma lista de pares, em que o primeiro elemento é o número de reviews diferentes e o segundo é o nome do User. No final da query transformamos a *PriorityQueue* numa lista ordenada pela ordem inversa, pois agora queremos a ordem decrescente.

   Para isto é criado um *Comparator* que organiza as listas anteriores por número de reviews diferentes.

   Para saber o número de reviews diferentes de um User inserimos todos os businesses ids num set e depois contamos o tamanho, dado que este não insere elementos repetidos.

   A partir daí inserimos na PriorityQueue se ele ainda não tiver X elementos ou se o topo da PriorityQueue mostrar um elemento com menos reviews que o atual.

   Assim temos **T = O (M * N)**

   N é o número de **users**.

M é o número de reviews que cada user fez

9. Dado o código de um negócio, determinar o conjunto dos X users que mais o avaliaram e, para cada um, qual o valor médio de classificação (ordenação cf. 5);

   Vamos começar por criar um map como uma estrutura auxiliar, inserindo neste todos os Users que avaliaram este negócio e o seu par com o número total de reviews e o seu valor médio de classificação, percorrendo assim todas as reviews.

   Para responder a esta questão é criado uma PriorityQueue de par de pares, em que o primeiro elemento é o nome do utilizador e o segundo par consiste no número de reviews e o segundo é o seu valor médio de classificação. No final da query transformamos a PriorityQueue numa lista ordenada pela ordem inversa, pois agora queremos a ordem decrescente.

   Para isto é criado um *Comparator* que organiza as listas anteriores por número de reviews a esse negócio e depois por ordem alfabética.

   A partir daqui basta inserir os top x users que avaliaram na PriorityQueue percorrendo o map acima criado.

   Assim temos **T = O (M * N)**

   N é o número de reviews.

   M é o número de users que cada fizeram reviews ao negócio dado.

10. Determinar para cada estado, cidade a cidade, a média de classificação de cada negócio.

    Para responder a esta questão é criado uma map que para cada estado vai ter um map que por vai ter uma lista dos pares nome do negócio e, a média de classificação, de cada negócio em função da sua cidade.

    Como tal, apenas teremos de fazer uma travessia pela lista de negócios que foram avaliados (e não por todos os negócios, como explicado anteriormente em i.) e ir acrescentando a esta estrutura.

    Temos que **T = O(N)**, sendo **N** o número de negócios **com reviews**

Classe Pair

Foi criada (e utilizada constantemente) uma classe Pair.

```java
public class Pair<F, S> {
    private final F first;
    private final S second;

    public Pair(F first, S second) {
        this.first = first;
        this.second = second;
    }

    public F getFirst() {
        return first;
    }

    public S getSecond() {
        return second;
    }
}
```

View

Esta parte do projeto é responsável por toda a parte visual, é dividida em dois.

i. **View** - uma parte mais geral que contem perguntas, o menu principal, o menu das queries, headers entre outros.

ii. **QueriesView** – Esta parte especializasse apenas na parte visual de todas as queries, de modo a facilitar a leitura, nos optamos por 3 modos diferentes de mostrar o resultado. Uma que simplesmente mostra os valores, normalmente usada para queries com uma frase, uma do tipo paginação que mostra resultados de x a y da lista e o utilizador tem a habilidade de executar "next" e "previous" para controlar x e y. Por último, optamos por fornecer uma visão mais detalhada para as queries mais complexas esta consiste na habilidade de mostrar toda a informação fornecida ou apenas uma "subcategoria".

Controller

i. **Controller** – Esta e a parte principal do Controller, controla todo o programa, recebe input do user e faz os respetivos pedidos ao Model a, View, ao Crono e também ao QueriesController .

ii. **QueriesController** – Este e a parte responsável por as queries, controla o menu das queries e é subdividido em controllers específicos para as queries com uma View mais complexa, que permite controlar a paginação como também a visão mais detalhada que permite encontrar uma subcategoria ou mostrar toda a informação como explicitado na parte da View.

iii. **Crono** – Esta classe é a que permite fazer a contagem do tempo. Temos os métodos para começar e para a contagem do tempo e o getter desse tempo com a conversão para segundos.

iv. **GestReviews** – Este é a parte similar ao sgr do projeto de c, contém toda a informação do projeto, contendo os três catálogos, o do Businesses,o do Users e o das Reviews. Também contém os paths de todos os ficheiros sendo eles os custom ou os defaults. Permite ir buscar todas estas estruturas e gravar e ler a informação em binário.

Testes

Sistema

OS	5.12.9-1-MANJARO x86_64
Máquina	Laptop (Micro-Star)
CPU	Intel Core i7-5700HQ (L1: 64 KiB L2: 6 MiB L3: 6 MiB)
RAM	15.52 GiB (1600 MHz)
Drives	Kingston SSD 223.57 GiB (6.0 Gb/s)

Performance

Tarefa	Tempo de Execução (s)	Utilização de memória (MB)
Carregar Ficheiros (.csv)	17.011	2383.387 (2.33 GB)
Carregar Ficheiros (.dat)	86.081	(filesize=306.23)
Guardar Ficheiros (.dat)	120.250	(filesize=306.23)
Query 1	0.248	124.096
Query 2	0.104	1.336
Query 3	0.907	174.948
Query 4	0.147	47.520
Query 5	1.016	294.788
Query 6	6.153	320.100
Query 7	0.231	30.560
Query 8	2.619	253.588
Query 9	2.063	82.292
Query 10	0.245	92.056
Estatísticas	0.026	1.528

Nota: Tempo é dado segundo a média de 10 testes. No caso da query depender do input do utilizador, é feito uma média entre os piores e melhores casos.

Conclusão

Face aos comentários da equipa docente em relação ao trabalho anterior, decidimos direcionar mais atenção para a estrutura do trabalho, o que teve consequência muito positivas na organização do mesmo.

Apesar de haver sempre espaço para melhorar, estamos satisfeitos com o trabalho que apresentamos.