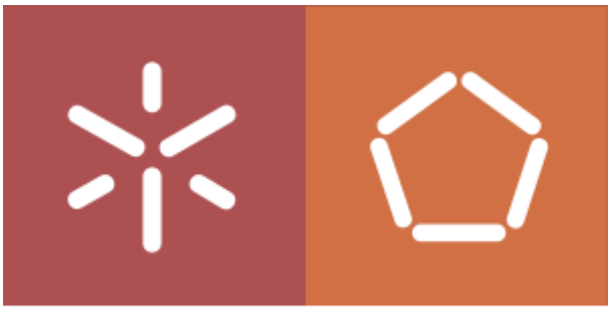


Universidade do Minho

Departamento de Informática

Projeto de Laboratórios de Informática III
Grupo 6

Gonçalo Pereira (a93168) Maria Gomes (a93314)

Gonçalo Afonso (a93178)

28 de junho de 2021

Introdução

1.1 Motivação

O presente relatório foi elaborado no âmbito da unidade curricular de *Laboratórios de Informática III* (LI3), do Mestrado Integrado em Engenharia Informática da Universidade do Minho e tem como objetivo descrever o desenvolvimento de um sistema de processamento e gestão de informações contidas em ficheiros CSV, utilizando a linguagem de programação Java.

1.2 Apresentação do problema

Os ficheiros CSV a serem processados contêm informações referentes à plataforma Yelp, plataforma de críticas e recomendação de estabelecimentos e negócios. É esperada a leitura de grandes quantidades de dados para memória. Além disso, pretendia-se a implementação de um conjunto de queries capazes de extrair informação necessária de uma forma eficiente e em tempo útil para o utilizador, assim como manipulação destes dados através de um interpretador.

1.3 Estratégia implementada

A conceção da aplicação seguiu os princípios da programação com interfaces.

Classes, interfaces e estruturas de dados

2.1 IUser e User

Respetivamente a interface e a classe que a implementa, servem para guardar dados de instâncias retiradas do ficheiros que guardem informação sobre users. A interface obriga a classe à implementação de getters da informação de um user (como o seu id ou nome), assim como um método de clonagem e de tornar em string. Além disso, a classe user ainda tem um método utilizado para validar se uma string dada pode ser transformada em um user, necessitando ter apenas 3 campos distinguidos por ';', não tendo o último campo (friends) obrigatoriamente de estar preenchido..

2.2 IBusiness e Business

Semelhante ao ponto anterior, existe uma interface e a classe que a implementa, neste caso para ficheiros que guardem informação sobre businesses, sendo por isso obrigados pela interface a implementação de diferentes getters e o método de validação verifica que todos os campos de um business estão presentes na string.

2.3 IReview e Review

O último tipo de dados guardados em ficheiro, as reviews também apresentam um conjunto de getters, impostos pela interface, diferenciados e o método de validação considera uma string válida como review caso apresente todos os campos de uma classe review, mesmo que o campo de texto esteja vazio.

2.4 IUserCat e UserCat

O conjunto dos vários users será guardado através da classe UserCat, que irá, como no projeto de C, guardá-los num hashMap em que utilizará o id de um user como a chave na hash. Decidimos seguir com o uso deste tipo de Collections para os diferentes dados (use,business e review) pois, de igual forma ao trabalho anterior, existe uma elevada necessidade de procura de elementos através do conhecimento do seu id, e podendo utilizar o mesmo para, em tempo constante encontrar, se existir, esse elemento na hash, o benefício de usar esta collection é evidente. Além disso, os vários testes de procura que serão demonstrados posteriormente neste relatório enfatizam ainda mais esta escolha.

A interface IUserCat obriga UserCat a implementar um método para adicionar um user no conjunto onde os mesmos são guardados, um método para devolver uma cópia de um hashMap com os users guardados até ao momento e um último método de verificação se um user com o id fornecido está presentemente guardado.

2.5 IBusinessCat e BusinessCat

Este conjunto funciona de forma semelhante ao ponto anterior, apenas com a distinção que trata de dados do tipo business, assim sendo, a interface IBusinessCat implica um método de adicionar um business no conjunto guardado, de devolver uma cópia de uma hash dos business guardados, verificar a existência de um business através de um certo id e um último método que devolve o tamanho do conjunto de business guardados (quantidade de business).

2.6 IReviewCat e ReviewCat

Para os dados de tipo reviews a estratégia mantém-se, sendo que a sua interface obriga à implementação de um método de adicionar uma review no conjunto de reviews guardadas, um método de retornar uma cópia da hash de reviews guardadas, um método que retorna a review com o id fornecido e, por último um método para transformar o conjunto das reviews guardadas em formato string.

Além disso, ReviewCat ainda implementa um método que verifica se existe alguma review sobre um fornecido business.

2.7 Model e Queries

O Model, assim como o nome indica na arquitetura por nós utilizada, MVC, será a classe que realizará análise e filtração dos dados. Será através desta classe que serão realizadas as diversas queries, que são implementadas segundo interfaces às mesmas dedicadas.

O Model para realizar as suas funções necessita primeiramente de carregar dados de ficheiros e portanto apresenta métodos de load que tratarão de recolher e pedir a validação dos diferentes tipos de dados, users, businesses e reviews. Este carregamento pode ser realizado diretamente pelo construtor, ou de modo a reutilizar uma instância da classe pode ser utilizado o método load de modo a carregar novos dados.

Além de carregar dados, o Model permite também guardar os dados carregados em um ficheiro objeto, de modo a, caso carrega um ficheiro do mesmo tipo, permita uma maior rapidez devido ao tamanho reduzido assim como à possibilidade de ignorar a validação dos dados, uma vez que já foram anteriormente validados.

2.7.1 Queries

Query1

Devido ao facto de esta query pedir a devolução de um par, um conjunto de negócios, assim como o seu total, resolvemos, juntar estes dois em uma classe auxiliar, NotReviewed, que irá ter como instâncias privadas um treeSet com os negócios e um inteiro que devolve o número total de negócios nesse treeSet. A utilização do treeSet deve-se a, como pedido pela query, que o conjunto de negócios que não tenham sido avaliados, seja organizado de forma alfabética assim, de forma a realizar esta operação de modo eficiente, assim que retiramos do conjunto total de

negócios, todos os que foram avaliados, vamos adicionando um a um no treeSet que, utilizando o comparator NotReviewedComp, irá permanecer organizado alfabeticamente.

Query2

Sendo que esta query pede para um determinado ano e mês o número global de reviews feitos assim como o número total de reviewers distintos, tivemos que usar um classe auxiliar diferente, neste caso, ReviewedPerMonth, que irá guardar num hashMap os diferentes reviewers, usando como chave o seu id, de modo que se numa nova review que seja encontra para o mês e ano pedidos, o utilizador já tenha sido guardado, não será criada uma nova entrada para ele.

Para chegar ao resultado pedido, percorremos o conjunto das reviews e para cada uma verificamos se a data da sua adição valida os requisitos de mês e ano, e em caso positivo, a mesma é adicionada na instância de ReviewedPerMonth, incrementando o número de reviews feitas assim como guardando, caso ainda não exista o id do reviewer.

Query3

Neste caso, o pedido é bastante semelhante ao do problema anterior sendo que em vez de ser pedido para encontrar reviews para um mês e ano específicos, é pedido para um certo user, separando as reviews pelos diferentes meses e verificar para cada, quantos negócios diferentes são avaliados. Assim, optamos por reutilizar a classe auxiliar da query2, devolvendo a query um array de 12 instâncias desta classe, cada uma representante de um mês do ano e , a hashMap que anteriormente servia para guardar os users distintos será, desta vez, utilizada para guardar os diferentes negócios avaliados em um determinado mês pelo user.

Além disso, por ser necessário calcular o score médio do user em cada mês, aumentamos a quantidade de informação que a classe guardava, de modo a guardar informação das estrelas totais dadas.

Query4

Semelhante à query3 alterando que em vez de procurar reviews feitas por um determinado user, procura reviews sobre um determinado business, guardando os users distintos que realizaram reviews no negócio em cada mês.

Query5

Percorremos o catálogo das reviews, procurando aquelas realizadas pelo userId dado, em caso afirmativo atualizamos a hash de ReviewsByBizName, registrando a quantidade de reviews para cada negócio. Por fim tornamos essa hash em uma lista e utilizando ReviewsByBizNameComp, ordenamos de forma decrescente de reviews do negócio.

Query6

Para as queries 6,8 e 9 que pedem o cálculo do conjunto dos top de um certo elemento, criamos uma classe auxiliar, TopReviews, onde, num inteiro será guardado o top pedido inicialmente no construtor da classe e uma dupla hash em que a chave da primeira hash será um determinado ano e a segunda hash servirá para guardar o conjunto dos top elementos nesse dado ano.

Esses elementos, serão representados pela classe TopReviewsAux que dependendo da ordem das string dadas no construtor de TopReviews, poderá representar um negócio ou um utilizador e, consequentemente a hashSet de TopReviewsAux guardará os distintos users que avaliaram o negócio ou os distintos negócios avaliados pelo user respetivamente.

Por último, TopReviews disponibiliza o método topBus que recebendo um determinado comparator irá criar um HashMap em que guardará para cada ano uma treeSet das topReviewsAux, organizada por esse comparator. Essa classe irá no fim devolver uma lista da concatenação dos top elementos de cada um desses treeSets.

Para a resolução deste problema percorremos o conjunto das reviews, e para cada uma adicionamos uma review na instância re TopReviews, fornecendo a data da review assim como definindo que cada TopReviewsAux representa um negócio, o user que realizou a review e o respetivo score dado, a partir daqui esta classe irá criar se necessário novas entradas na hash ou atualizar as existentes.

Percorridas as reviews devolvemos o conjunto ordenado dos negócios através da utilização do método topBus fornecendo o comparador que ordena os TopReviewsAux pelo total de reviews.

Query7

Para este problema criamos a classe auxiliar Query7aux que apenas guarda o id de um negócio, a sua cidade e o respetivo total de reviews nele feito, esta classe auxiliar tem como único propósito facilitar o print dos resultados para o utilizador.

A solução começa por criar uma dupla hash em que a key da primeira será o nome de uma cidade e a hash interior terá como chave o id de um negócio e valor o número de reviews feitas no mesmo.

Para preencher esta hash percorremos o conjunto dos businesses, criando ou atualizando entradas para cada cidade e negócio analisados. Posteriormente, criamos uma hash, results, em que para cada cidade, guardaremos uma lista dos top negócios. Para cada cidade, tornamos a sua hash em uma lista ordenada por ordem decrescente do número de reviews e pegamos na sublist de 0 a top e adicionamos a mesma em results.

Por último, tornamos results, por conveniência, em uma lista de Query7aux.

Query8

A query inquire pelo conjunto dos top users que mais negócios diferentes avaliaram.

Utilizando a estratégia da query6, as únicas distinções serão que TopReviews apenas guardará dados para um ano, arbitrariamente escolhido por nós, sendo que o mesmo é irrelevante desde que todos os elementos sejam no mesmo adicionado, também como explicado anteriormente, TopReviewsAux guardará dados sobre um user em vez de um business e, o comparator no final utilizado para devolver o top users, irá ordenar pelo tamanho do conjunto de negócios distintos de forma decrescente.

Query9

Novamente usando a classe TopReviews, utilizaremos TopReviewsAux como dados de um user, porém apenas serão adicionados a TopReviews (no mesmo ano), users que tenham feito reviews em um certo negócio ao método query9 fornecido. O comparator utilizado será o mesmo que na query6, comparando pelo maior número de reviews feitas e alfabeticamente no caso de números iguais.

Query10

A query10 requer o top negócios de cada cidade para cada estado, precisando para isso de um triplo hashMap, de modo a facilitar a adição de novas entradas e atualização de entradas pré-existentes, criamos a classe auxiliar StateBusiness que trata de fazer as verificações necessárias antes de cada operação (criar uma entrada para um estado, cidade ou business caso ainda não existam quando pedido para adicionar um score ao mesmo).

Para chegar ao resultado pretendido, percorremos a hash dos business de modo a criar as entradas necessárias em StateBusiness e posteriormente, percorremos a hash das reviews atualizando as entradas de cada negócio. Como é pedido a quantidade de avaliações de todos os negócios e não apenas dos que são avaliados, temos de percorrer as duas hashes, no caso de apenas pedir os avaliados, bastaria apenas percorrer a hash das reviews.

2.8 GestReviews

Classe que representa o controller na arquitetura MVC, permite o fluxo de dados entre o Model e a View (UI), assim como a interação do utilizador com o programa.

O controlador é encarregado de fornecer à view a informação necessária para gerar os vários menus, assim como as condições para as opções nesses disponíveis, através da definição de preconditions que depois serão lidas pela view e consequentemente demonstrará a disponibilidade das opções.

Além de definir as condições para cada opção de um menu, o controlador também é onde se define o que cada opção executa, através da definição de um handler, muitas das vezes implica a execução de um método como por exemplo, a opção de load de dados que será

redirecionada para o método loadData e posteriormente terá o seu próprio menu e conjunto de opções.

No caso da execução de uma opção correspondente a uma query, devido à nossa opção de demonstrar os resultados em forma de tabela, os resultados têm de ser tratados no método correspondente no controller de modo transferir para a view um conjunto de listas de strings, representantes das linhas da tabela e cada string representando uma coluna, verificando também a validez da página pedida pelo utilizador.

2.9 UI

Classe que representa a view na arquitetura MVC, transmite para o utilizador a visualização dos dados recebidos do controller.

A view por nós utilizada trata-se de uma adaptação da view criada nas aulas práticas de POO, mais especificamente do trabalho DriveIt.

Esta view implementa duas interfaces, PreCondition e Handler que são as condições e o que executar para cada opção de um menu respetivamente. A view recebe uma lista de string em que cada string será uma opção e, verificando se a pré-condição de cada opção foi validada demonstra-a ao utilizador.

As adaptações que realizamos na classe resumem-se aos métodos de impressão simples de mensagens coloridas, adaptação para leitura de valores dentro de um certo alcance e o método de impressão em forma de tabela.

Estrutura do projeto

Como anteriormente mencionado, o nosso projeto segue a estrutura Model View Controller, estando organizado pelas camadas:

- GestReviews (Controller) : Gere o funcionamento do programa, permitindo a interação entre a camada de dados (Model) e a de visualização dos mesmos (view).
- UI (View) : Apresenta ao utilizador os dados.
- Model : Composta por todas as outras classes, carrega os dados do programa assim como a sua gerência e modificação.

Testes de desempenho

De forma a medir o desempenho do programa, assim como as diferenças entre o uso de diferentes collections para as queries assim como diferentes métodos de carregamento de ficheiros, criamos várias classes de teste, possíveis de correr em separado da aplicação principal.

Todos os tempos que serão apresentados correspondem a uma média calculada após 10 execuções.

4.1 Leitura de ficheiros (readingBusinessTest)

Começando pelo carregamento de dados através de leitura de ficheiros, por questões de tempo apenas se testa a leitura do ficheiro dos business caso contrário a leitura de todos os ficheiros seria muito extensa, e acreditamos que um único ficheiro já dá uma boa percepção para retirar conclusões. A seguinte tabela demonstra o tempo médio em milissegundos.

Método de leitura	Tempo medido (milissegundos)
Buffer que guarda várias as linhas	244
Char a Char	618
Char a Char sem buffer	788
Linha a Linha	171
Leitura de um objeto	234
Leitura de um objeto sem buffer	8278

Pela análise dos resultados podemos concluir que o método aplicado no trabalho (primeiro), embora não sendo o mais rápido, sendo passado pela leitura linha a linha e a leitura de um objeto, sofre de alguma discrepância devido a não ser uma chamada direta do método de leitura do ficheiro, mas mesmo assim, não apresenta uma diferença muito notável com estes dois.

Além disso, é de notar a diferença estonteante entre a leitura de um objeto com e sem buffer. Ainda para mais, embora a leitura linha a linha tenha sido mais rápida do que a leitura de objeto, a utilização desta última no caso do objeto contentor dos três ficheiros de dados seria

mais benéfica, pois a falta de validação nas reviews ganharia bastante tempo que nos outros métodos tem de ser desperdiçado.

4.2 Procura de objetos

Ainda na mesma classe de teste, estudámos os desempenhos na procura de um business, através do seu id, em diferentes collections.

BusinessId procurado : 8zehGz9jnxPqXtOc7KaJxA

Collection	Tempo medido (milissegundos)
HashMap	9*10^(-4)
Lista	4.3
Lista (contains object)	0.5
Set	19.2
Set (contains object)	0.001

Analisando a tabela podemos ver que no caso de se realizar procuras através do uso de ids, o uso de HashMaps é exponencialmente superior.

4.3 Collections, diferenças numa query (query5Test)

Utilizando a query 5 como exemplo, modificámo-la de modo a tomar proveito de diferentes tipos de collections e, utilizando dois diferentes users, verificamos as diferenças no uso das collections para um user que realiza poucas reviews e consequentemente obriga a ligeiro número de inserções pela collection, e outro que obriga a um número mais considerável de inserções.

Poucas inserções (2) : ak0TdVmGKo4pwqdJSTLwWw

Collection	Tempo medido (milissegundos)
HashMap	263.7

Lista	301.56
Set	290.2
Lista Ligada	262.9
Hash Ligado	202.2

Várias inserções (670) : RtGqdDBvvBCjcu5dUqwfzA

Collection	Tempo medido (milissegundos)
HashMap	269.3
Lista	2430.3
Set	6063.7
Lista Ligada	2760.9
Hash Ligado	292.2

Concluímos portanto que, com o acumular do número de inserções, o uso de listas, ligadas ou não, e sets torna-se altamente ineficiente quando comparado com o uso de HashMaps (método utilizado na query 5) e hashes ligadas, que apresentam entre si uma diferença quase inexistente. Claro que no cenário deste projeto trabalhamos com procuras em que grande parte das vezes possuímos conhecimento do id de um elemento, que facilmente usamos como chave em uma hash, caso isto não aconteça, os resultados poderiam ser bastante diferentes.

4.4 Queries e estatísticas (performanceTest)

Por último testamos o desempenho do programa no seu estado final, isolando as diferentes funcionalidades do mesmo.

Para certas queries que envolviam o cálculo de tops escolhidos pelo utilizador, realizamos diferentes testes para verificar o impacto da performance com o aumento do top.

Da mesma forma para queries que envolviam a procurar para determinados negócios ou users, utilizamos IDs que envolveriam diferentes cargas de trabalho por parte do programa.

Testes Simples	Tempo medido (milissegundos)
Query1	491.1
Query2	261.1
Query4	276.8
Query5	291.6
Query7	701.6
Query10	942.2
Estatísticas	3935.3

Diferentes ID	Tempo medido (milissegundos)
Query3 (poucas reviews - 2)	290
Query3 (muitas reviews - 670)	242.5

Testes Simples	Top 1	Top 10	Top 100	Top 1000
Query6	870.7	863	886	859.5
Query8	2094.3	2051.4	2021.2	2019.9
Query9	283	543.7	246.5	241.6

Analisando os dados obtidos, podemos verificar que as diversas queries em que foram testados diferentes cargas de trabalhos, a variação da mesma não impacta a performance da query.

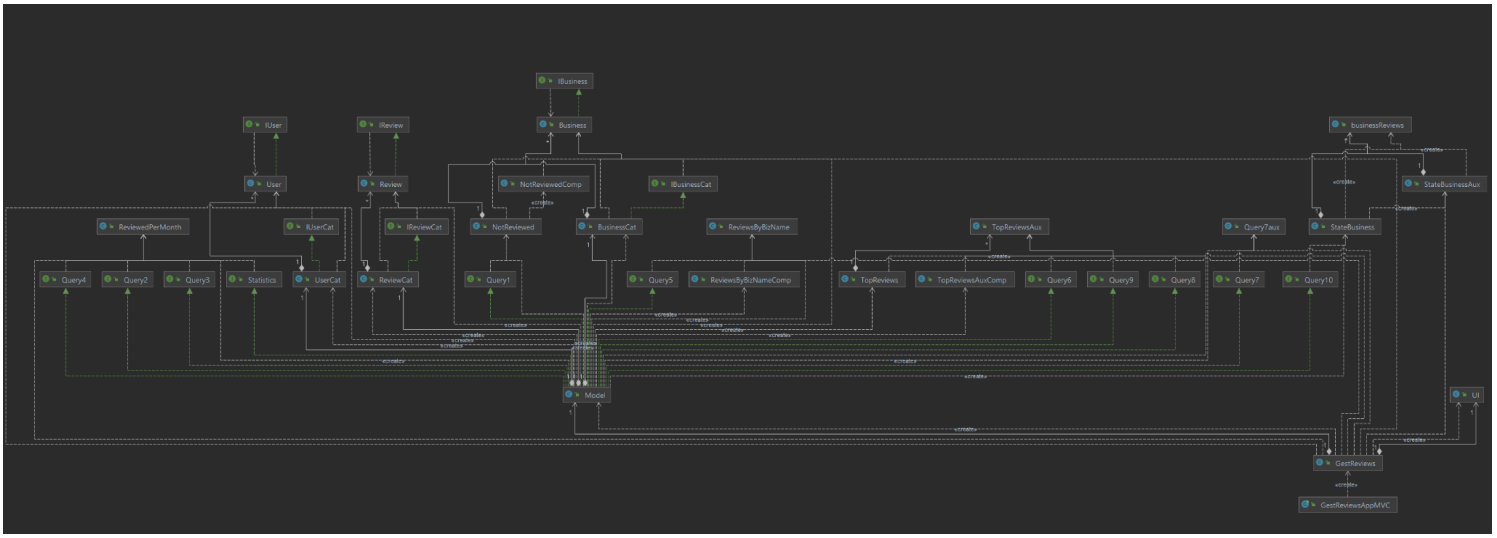
Conclusão

Acreditamos ter conseguido alcançar os objetivos que nos foram propostos para este projeto, sendo o nosso programa capaz de responder a todas as queries assim como demonstrar as estatísticas, permitindo uma interação por parte do usuário simples e com respostas relativamente rápidas.

Através do uso de interfaces, tal como pedido no enunciado, o código tornou-se mais flexível e com potencial para ser expandido mais facilmente.

Por último, acreditamos que o conjunto de testes que criamos prova a eficiência das collections por nós utilizadas, assim como a rapidez e eficiência das queries.

Diagrama de Classes



Desenho da Estrutura de Dados