

Gestão de Biblioteca

Universidade de Aveiro

Diogo Correia, Pedro Rocha



Gestão de Biblioteca

DETI

Universidade de Aveiro

Diogo Correia, Pedro Rocha

(90327) diogo.correia99@ua.pt, (95057) pedromrocha@ua.pt

20/11/2019

Conteúdo

1. Acrónimos
2. Introdução
3. Metodologia
4. Aplicação
5. Contribuição dos autores

Capítulo 1

Acrónimos

UC'S – Unidades Curriculares

MPEI – Métodos Probabilísticos para Engenharia Informática

Capítulo 2

Introdução

Este projeto, proposto pela disciplina de MPEI, tem como principal objetivo criar uma aplicação que fará uma gestão de forma inteligente de uma biblioteca de livros. Esta aplicação, possibilitará ao utilizador de pesquisar por um determinado livro através do seu nome e/ou categoria, e obter um feedback de outros livros com o nome idêntico e da sua existência no acervo. Mais informação no seguimento deste relatório.

Este trabalho será desenvolvido com o conhecimento obtido nas aulas da disciplina de MPEI, através de alguma auto-aprendizagem através da internet, e através de algum conhecimento adquirido em UC'S anteriores, tais como Programação 1 e 2. O trabalho será escrito usando a linguagem de programação Java.

Capítulo 3

Metodologia

O início da aplicação deverá ter um pequeno menu de inicialização. Aqui, o utilizador poderá escolher entre pesquisar um livro através do seu título, onde este mesmo irá aparecer, caso exista no acervo, juntamente com todos os outros livros com um título similar a esse e de todas as categorias existentes, por outro lado, o utilizador poderá escolher primeiramente a categoria, e depois aí é que pesquisa pelo nome do livro.

Será também possível requisitar livros que estejam disponíveis no acervo, ou até mesmo verificar quais livros estão requisitados no momento. Estes, deverão apresentar a informação de quando foram requisitados, quando têm de ser devolvidos, e que pessoa os requisitou. Este trabalho terá de ter algumas características específicas pedidas pelos docentes que lecionam esta unidade curricular, como por exemplo a implementação de pelo menos dois filtros de bloom e usar recursos a tabelas de minhash. No relatório final também deverá constar testes feitos para verificar a funcionalidade de cada um dos módulos.

Capítulo 4

Aplicação

4.1 - Base de Dados	9
4.1.1 - Retirar da web	9/10
4.1.2 - Criar a base de dados	10/11
4.1.3 - Inicializar a base de dados	11
4.2 - Estrutura da aplicação	12
4.2.1 - Book.java	12
4.2.2 - Library.java	13
4.2.2.1 - Testar a Library	14
4.2.3 - Category.java	14
4.2.4 - Carregar conteúdo da base de dados	15
4.3 - BloomFilter	15
4.3.1 - Testar o BloomFilter	16
4.4 - MinHash	16/17
4.4.1 - Similaridade e Distância de Jaccard	17

4.4.1.1 - Testar a similaridade -----	18
4.4.2 - Testar o Min Hash -----	18
4.5 - Menu -----	18/19
4.5.1 - Listar Livros -----	20
4.5.1.1 - Listar todos os livros -----	20
4.5.1.2 - Listar livros por categoria -----	21
4.5.2 - Pesquisar Livros -----	22
4.5.2.1 - Verificar existência de livro -----	22
4.5.2.2 - Pesquisar livros com títulos parecidos ---	23
4.5.3 - Pesquisar Autores -----	24
4.5.3.1 - Verificar se autor existe -----	24
4.5.3.2 - Pesquisar livros de um autor -----	25
4.5.4 - Administração -----	25
4.5.4.1 - Adicionar Livro -----	26
4.5.4.2 - Remover Livro -----	26
4.5.4.3 - Requesitar Livro -----	27
4.5.4.4 - Devolver Livro -----	28
4.5.5 - Abandono da ação -----	29

4.1 Base de Dados

A base de dados desta aplicação consiste num ficheiro *JSON*, com o nome **data_base.json**. Este ficheiro *JSON* está dividido em objetos *JSON* correspondentes às várias categorias de livros que existem na biblioteca, e cada objeto de categoria, está dividido num dicionário que representa todos os livros dessa categoria. O dicionário de cada categoria tem como *Keys* os IDs dos livros, e cada *Key* tem como *Values* um array que contém as informações do livro correspondente a esse ID. O array contém o nome do livro, o autor, e uma *flag* que indica se o livro foi, ou não, requisitado.

```
{  
    categoria: {  
        "1" : ["título", "autor", "true"],  
        ...  
    }  
    ...  
}
```

4.1.1 Retirar da web

Para que houvessem já alguns livros no acervo da biblioteca, foi criado um *script* que acede ao site da Fnac, em www.fnac.pt, e, mediante o resultado do Top 100 livros de certas categorias, é baixada a página *HTML*, e são filtradas classes e ids desse ficheiro que contem os títulos e autores dos livros. A *Class* encarregue disto é a **FetchFromWeb.java**

FetchFromWeb.java

```
|    private String htmlContent  
|    private String name  
|    public FetchFromWeb()  
|    public Map<String, String> getBooksInfoMap()  
|    public List<List<String>> getBooksList()
```

Esta *Class* usa uma biblioteca do *Java*, **URLConnection**, para aceder ao conteúdo das páginas da Fnac, e guarda esse conteúdo no objeto **FetchFromWeb.htmlContent**. Depois, a função **FetchFromWeb.getBooksInfoMap()** usa outras funções desta *Class* para formatar o conteúdo todo num dicionário que a cada título corresponde o autor. Este conteúdo é, depois, transformado numa lista de listas de *Strings* (**List<List<String>>**), em que cada lista de *Strings* tem a o título, o autor e uma *flag* com o valor *false*, que indica que o livro ainda não foi requisitado.

4.1.2 Criar a base de dados

A *Class* **DataBaseCreator** usa as funções da *Class* **FetchFromWeb** para obter a tal lista de listas de *Strings*, como referido no tópico **4.1.1 Retirar da Web**. O objetivo desta *Class* é de exportar o conteúdo da lista obtida para um ficheiro *JSON* bem formatado.

DataBaseCreator.java

```
|    private FetchFromWeb[] ffwArray  
|    private List<Integer> ids  
|    private int idCounter  
|    public DataBaseCreator()  
|    private boolean exportDB()
```

São, então, criados vários objetos do tipo **FetchFromWeb**, um para cada categoria de livros. Mediante o número de conjuntos de livros, são atribuídos IDs a cada conjunto para, assim, a função **DataBaseCreator.exportDB()** criar os **JSONObject** e **JSONArray** necessários, que posteriormente vão ser escritos num ficheiro *JSON* através de um **PrintWriter**.

4.1.2 Inicializar a base de dados

A *Class* **DataBaseInitializer** é composta por, apenas, uma função **DataBaseInitializer.main()** que vai criar um objeto do tipo **DataBaseCreator** e que lhe vai passar os links para o site da Fnac. Esta *Class* chama a função **DataBaseCreator.exportDB()** para proceder à escrita da base de dados no ficheiro *JSON*.

Para correr esta *Class* e criar, assim, a base de dados, tem de passar os seguintes comandos:

1. Modificar o *\$CLASSPATH* do *Terminal* para poder aceder a uma biblioteca externa do java:

```
$ export CLASSPATH=./path/to/this/folder/library-management/lib/json-simple-1.1.jar
```

2. Compilar todos os ficheiros do java para “construir” o programa:

```
$ javac *.java
```

ou, caso não seja feito o passo 1.:

```
$ javac -cp /path/to/this/folder/library-management/lib/json-simple-1.1.jar *.java
```

3. Correr o ficheiro **DataBaseInitializer.java**:

```
$ java DataBaseInitializer.java
```

4.2 Estrutura da aplicação

Esta aplicação usa algumas *Classes* que compõem o esqueleto do programa. Sendo isto uma biblioteca, existe a *Class* **Book**, que representa cada livro individual, a *Class* **Library**, que representa a biblioteca que armazena os livros e a *Class Enum* **Category**, que indica quais as categorias existentes na biblioteca.

4.2.1 Book.java

A *Class* **Book** trata de tudo relacionado com as informações de um livro. Indica o título, o autor, a categoria, o id, se este foi requisitado, e se sim, a data do requesito.

Book.java

```
|    private int id  
|    private String title  
|    private String author  
|    public Category category  
|    private boolean borrowed  
|    private Date timeWhenBorrowed  
|    public Book()  
|    public void borrow()  
|    public void returnBook()
```

Esta *Class*, para além de outras, dispõe da função **Book.borrow()** para modificar o objeto **Book.borrowed** e o objeto **Book.timeWhenBorrowed** e a função **Book.returnBook()** para reverter a ação da função anterior. A data de requesito é obtida através da *Class* do Java **Calendar**.

4.2.2 Library.java

A *Class* **Library** é constituída por uma lista de livros (**List<Book>**) que compõe o acervo, e dá um tracking do ID dos livros, para assim fornecer um ID não repetido a novos livros.

Library.java

```
|    private int lastId  
|  
|    private List<Book> acervo  
|  
|    private List<Integer> ids  
|  
|    private String name  
|  
|    public Library()  
|  
|    private boolean addBook()  
|  
|    public boolean removeBook()  
|  
|    public boolean updateDB()  
|  
|    public int loadLibrary()
```

Esta *Class*, dispõe de funções para adicionar e remover livros do acervo, respetivamente **Library.addBook()** e **Library.removeBook()**. Estas funções modificam a diretamente lista de livros, e também modificam a ficheiro *JSON* correspondente à base de dados. Para isso, é usada a função **Book.updateDB()** sempre que existe alguma alteração, tanto na lista de livros, como nos livros em si (no caso do requisitar de um livro). Esta função simplesmente chama a função **DataBaseCreator.exportDB()** e passa-lhe a lista de livros atualizada. A partir daí é tudo trabalho para a *Class* **DataBaseCreator**.

Para que seja guardada a sessão do utilizador, e para que este não perca as mudanças que este efetuou à base de dados, existe a função **Book.loadLibrary()**, que é chamada no construtor **Book.Book()**, para que sempre que o utilizador corra o programa, a base de dados é uploaded na **Library**. A função **Book.loadLibrary()** usa funções da *Class* **LoadFromDB**, que é tratada num tópico **4.2.4 Carregar conteúdo da base de dados**, mais à frente. Esta função, à medida que recebe a informação de mais um livro, cria um novo objeto do tipo **Book** e adiciona-o o ao acervo.

4.2.2.1 Testar a Library

Em forma de *debug*, existe uma *Class* **TestLibrary**, que cria um objeto do tipo **Library** na função **TestLibrary.main()**, e utiliza funções desta *Class* para criar a biblioteca e fazer prints na consola de vários testes para entender o ponto de situação desta estrutura. Esta *Class* de teste utiliza funções como **Library.acervo()**, **Library.borrow()**, **Library.removeBook()** e **Library.addBook()**.

4.2.3 Category.java

Esta *Class* é um *Enum* com todas as categorias da biblioteca. Apresenta algumas funções para agilizar a implementação e o uso desta *Class*.

Category.java

```
| LITERATURE, THRILLER, KIDS, SYFY  
| public static String getName()  
| public static Category getCategory()  
| public static Category[] getCategories()
```

A função **Category.getName()** recebe um objeto do tipo **Category** e retorna a categoria em formato de **String**. A função **Category.getCategory()** recebe um objeto do tipo **String** e retorna um objeto do tipo **Category** dependendo de qual *String* tenha recebido. A função **Category.getCategories()** retorna uma lista de objetos do tipo **Category**.

4.2.4 Carregar conteúdo da base de dados

Para carregar a informação contida no ficheiro *data_base.json* para a *Class RunLibrary* é usada a *Class LoadFromDB*. Esta *Class* recebe o conteúdo do ficheiro *JSON* através de um *FileReader* e de um *JSONParser*. A função *LoadFromDB.getCategoryContentMap()* organiza o conteúdo por categorias e formata tudo num dicionário. O dicionário é um objeto **Map** cujas *Keys* são do tipo **Integer** e os *Values* são do tipo **List**. Estas listas são listas de *Strings*.

LoadFromDB.java

```
|    private String path  
|    private Map<Category, JSONObject> categContentRaw  
|    public LoadFromDB()  
|    public Map<Integer, List<String>> getCategoryContentMap()
```

4.4 BloomFilter

De uma maneira muito geral o Filtro de Bloom é utilizado neste trabalho com a finalidade de verificar se algum livro pertence à base de dados que o grupo criou. O utilizador escreve o nome do livro no programa executado no terminal e de seguida esse nome é analisado no Filtro de Bloom. Caso exista algum livro com esse nome, o programa irá informar o utilizador que esse mesmo livro está presente no acervo, caso contrário aparecerá uma mensagem com a finalidade de informar o utilizador que o livro não faz parte do acervo.

O Filtro de Bloom também é usado neste trabalho com uma outra função, que é verificar se determinado autor tem livros presentes no acervo ou não, usando o mesmo procedimento acima descrito. No entanto, o Filtro de Bloom tem uma particularidade: quando um livro não pertence ao acervo ele definitivamente não consta na biblioteca de livros mas o contrário já não se verifica, isto porque o Filtro de Bloom pode assumir falsos positivos. Ou seja, nunca se sabe ao certo se determinado livro, realmente, pertence ao acervo.

4.4.1 Testar o BloomFilter

Com a construção do Filtro de Bloom foi-nos pedido também uma função de teste para verificar o comportamento do mesmo. Nesta função de teste foram criados dois arrays de strings que continham vários nomes de livros, uns eram iguais em ambas as strings outros não. Com isto conseguimos verificar que o Filtro de Bloom desenvolvido estava a funcionar de maneira correta pois, na comparação de duas strings, ele dizia-nos de forma correta quais eram os livros que pertenciam aquela “mini” biblioteca.

4.4 MinHash

A *Class* **MinHash** apresenta funções que permitem contruir um array de **int** com várias hashes mínimas. O seu contrutor recebe o número de hashes, o tamanho dos *shingles* e uma list de hashes do tipo **Hash**.

MinHash.java

```
|    private int nHashes  
|    private int nmrCharsPerShingle  
|    private Hash[] hashList  
|    public int nmrShingles  
|    public MinHashes()  
|    public String[] makeShingles()  
|    public int[] getHashesForShingle()  
|    public int hash()  
|    public int[] minHashes()
```


Mediante o tamanho do *shingle*, o número de *hashes*, e uma lista de objetos do tipo **Hash**, tudo passado no construtor desta *Class*, as várias funções podem ser chamadas para obter o array de *hashes* mínimas.

A função **MinHash.makeShingles()** recebe uma *String* que é dividida em *shingles*. O exemplo de retorno desta função, para uma *String* “Harry Potter”, em que o tamanho das *shingles* é de três caracteres é o seguinte: {“har”, “arr”, “rry”, “ry”, “y p”, “po”, “pot”, “ott”, “tte”, “ter”}. Através deste array de *Strings*, transforma-se cada *shingle* numa *hash*, através da função **MathWorksFunctions.string2hash()**, ficando cada *shingle* codificado numa *hash*. A função **MinHash.getHashesForShingle()** retorna um array de *hashes* para a *shingle*, quando já transformada numa *hash*. Por fim, a função **MinHash.minHashes()** compara todos os arrays de *hashes* para cada *shingle*, e vê qual a menor *hash* em cada posição de todos os arrays de *hashes*. Todas as *hashes* menores são guardadas num array, que vai ser o array de *hashes* mínimas, o “minHash”.

Exemplo:

{“12345”, “26542”, “2045”} | {“11345”, “5024”, “10235”} | {“50020”, “25021”, “3325”}

Nestes arrays de *hashes*, a *hash* mais “pequena” na posição 0 é a 11345, sendo que $11345 < 12345 < 50020$. Seguindo esta lógica, o array de *hashes* mínimas é:

{“11345”, “5024”, “2045”}

4.4.1 Similaridade e distância de Jaccard

Com os arrays de *hashes* mínimas, calcula-se a distância de Jaccard entre um par desses arrays e recebe-se o valor da distância. Quanto maior a distância, maior é a similaridade entre duas *Strings*.

O cálculo da distância é a interseção entre os dois arrays, a dividir pela união desses mesmo dois arrays.

4.4.1.1 Testar a similaridade

A Class **TestSimilarity** calcula os *arrays* de *hashes* mínimas para duas *Strings*, através das funções da Class **MinHash** e calcula a similaridade entre essas duas *Strings* através da distância de Jaccard.

4.4.2 Testar o Min Hash

A Class **TestMinHash** calcula apenas o *array* de *hashes* mínimas de uma *String*, como forma de *debug* da Class **MinHash**.

4.5 Menu

A interface do menu da aplicação é feita através do *Terminal*. O utilizador corre o ficheiro **RunLibrary.java** para iniciar o programa, através dos seguintes comandos:

1. Modificar o `$CLASSPATH` do *Terminal* para poder aceder a uma biblioteca externa do java:

```
$ export CLASSPATH=./path/to/this/folder/library-management/lib/json-simple-1.1.jar
```

2. Compilar todos os ficheiros do java para “construir” o programa:

```
$ javac *.java
```

ou, caso não seja feito o passo 1.:

```
$ javac -cp /path/to/this/folder/library-management/lib/json-simple-1.1.jar *.java
```

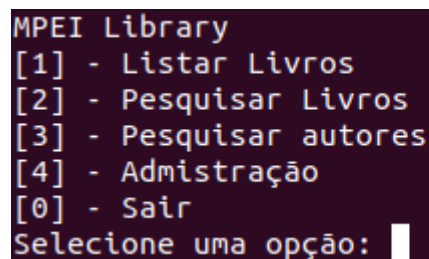
3. Correr o ficheiro **RunLibrary.java**:

```
$ java RunLibrary.java
```

RunLibrary.java

```
| static String libName  
| static BloomFilter bm  
| static Library lib  
| static Hash[] listHash  
| static MinHash minHash  
| static List<Book> listOfBooks  
| public static void displayMenu()  
| public static void main()  
| public static double similarityValue()  
| public static int getIntersections()  
| public static int[] getMinHashes()  
| public static void fillHashList()  
| public static void filterBooksByCategory()
```

Esta *Class* possui uma *main*, onde vai mostrar um menu inicial, criado pela função **RunLibrary.displayMenu()**. Neste menu o utilizador pode escolher uma de várias opções, criando um input através do teclado do número da opção, que vai ser lido através de um **Scanner**. Através de um *switch-case* do input do utilizador, é escolhido o que fazer.



```
MPEI Library  
[1] - Listar Livros  
[2] - Pesquisar Livros  
[3] - Pesquisar autores  
[4] - Administração  
[0] - Sair  
Selecione uma opção: |
```

1. Screenshot do menu inicial

4.5.1 Listar Livros

Neste menu, o utilizador o utilizador pode escolher entre duas opções: Listar todos os livros existentes no acervo, no ecrã, ou especificar uma categoria de livros, e depois listar todos os livros dessa categoria essa categoria.

```
Listar Livros
[1] - Listar todos os livros
[2] - Listar livros por categoria
[0] - Anterior
Selecione uma opção: 
```

2. Screenshot do menu Listar Livros

4.5.1.1 Listar todos os livros

Para a listagem de todos os livros, é feita apenas uma iteração pela **RunLibrary.listOfBooks**, e é feito um print de cada elemento do seu conteúdo, que é do tipo **Book**.

```
61. Frankenstein | Mary Shelley - [Ficção Científica]
62. Dune | Frank Herbert - [Ficção Científica]
63. Deuses Americanos | Neil Gaiman - [Ficção Científica]
64. Harry Potter e a Ordem da Fénix | J.K.Rowling - [Ficção Científica]
65. Harry Potter e a Pedra Filosofal Vol 1, Prémio Hans Christian Andersen 2010 | J. K. Rowling - [Ficção Científica]
66. A Guerra dos Tronos, As Crónicas de Gelo e Fogo Vol 1 | George R. R. Martin - [Ficção Científica]
67. A Tormenta de Espadas, As Crónicas de Gelo e Fogo Vol 5 | George R. R. Martin - [Ficção Científica]
68. O Festim dos Corvos, As Crónicas de Gelo e Fogo Vol 7 | George R. R. Martin - [Ficção Científica]
69. A Muralha de Gelo, As Crónicas de Gelo e Fogo Vol 2 | George R. R. Martin - [Ficção Científica]
70. A Glória dos Traidores, As Crónicas de Gelo e Fogo Vol 6 | George R. R. Martin - [Ficção Científica]
71. O Mar de Ferro, As Crónicas de Gelo e Fogo Vol 8 | George R. R. Martin - [Ficção Científica]
72. A Dança dos Dragões, As Crónicas de Gelo e Fogo Vol 9 | George R. R. Martin - [Ficção Científica]
73. O Despertar da Magia, As Crónicas de Gelo e Fogo Vol 4 | George R. R. Martin - [Ficção Científica]
74. Os Reinos do Caos, As Crónicas de Gelo e Fogo Vol 10 | George R. R. Martin - [Ficção Científica]
75. A História de Uma Serva | Margaret Atwood - [Ficção Científica]
76. A Fúria dos Reis, As Crónicas de Gelo e Fogo Vol 3 | George R. R. Martin - [Ficção Científica]
77. Fahrenheit 451 | Ray Bradbury - [Ficção Científica]

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: 
```

3. Screenshot da listagem de todos os livros

4.5.1.2 Listar livros por categoria

Para a listagem dos livros por categoria, é dado ao utilizador um novo menu, onde este pode escolher qual categoria quer ver e depois é feito o display. Mediante a escolha do utilizador, são filtrados todos os livros que apresentam a categoria indicada e é feito um print dos mesmos. A filtragem usa a função `RunLibrary.filterBooksByCategory()`, que usa a função `MinHash.getMinHashes()` e `MinHash.similarityValue()` para perceber, das categorias existentes, qual a escolhida pelo utilizador-

```
Categorias
[1] - Literatura
[2] - Thriller
[3] - Crianças
[4] - Ficção Científica
[0] - Anterior
Selecione uma opção: █
```

4. Screenshot da escolha das categorias

```
1. Por Quem os Sinos Dobram | Ernest Hemingway - [Literatura]
2. 1984 | George Orwell - [Literatura]
3. Admirável Mundo Novo | Aldous Huxley - [Literatura]
4. O Estrangeiro | Albert Camus - [Literatura]
5. A Divina Comédia | Dante Alighieri - [Literatura]
6. Os Miseráveis - Livro 1 | Victor Hugo - [Literatura]
7. Crime e Castigo | Fiódor Dostoiévski - [Literatura]
8. Dom Quixote de la Mancha | Miguel de Cervantes - [Literatura]
9. Os Miseráveis - Livro 2 | Victor Hugo - [Literatura]
10. Siddhartha, Um Poema Indiano | Hermann Hesse - [Literatura]
11. Se Isto é um Homem | Primo Levi - [Literatura]
12. A Quinta dos Animais | George Orwell - [Literatura]
13. Memorial do Convento | José Saramago - [Literatura]
14. O Velho e o Mar | Ernest Hemingway - [Literatura]
15. Lolita | Vladimir Nabokov - [Literatura]
16. Contos de São Petersburgo | Nikolai Gógol - [Literatura]

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

5. Screenshot da listagem dos livros de Literatura

4.5.2 Pesquisar Livros

Neste menu, o utilizador pode escolher entre duas opções: Verificar a existência de um livro no acervo, que é feita recorrendo à *Class* **BloomFilter**, ou então pesquisar por todos os livros que tenham um nome similar ao nome que o utilizador passe na consola, sendo que a verificação de similaridade entre a *String* do utilizador, e a *String* referente ao nome de cada livro do acervo, é feita através do cálculo da distância de jaccard entre os arrays de hashes mínimas de cada *String*, que são obtidas através da *Class* **MinHash**.

```
Pesquisar Livros
[1] - Verificar existência de livro
[2] - Listar livros com títulos parecidos
[0] - Anterior
Selecione uma opção:
```

6. Screenshot do menu Pesquisar Livros

4.5.2.1 Verificar a existência de livro

Para verificar a existência de um livro, o utilizador é, primeiramente, convidado a escrever o título de um livro. Dentro da *Class* **RunLibrary** são inseridos os títulos de todos os livros do acervo no **BloomFilter**, através da função **BloomFilter.insert()** e, por fim, é verificado se a *String* passada pelo utilizador pertence ao **BloomFilter**, através da função **BloomFilter.isMember()**. Se esta função retornar *true*, então existe a possibilidade de o título dado pelo utilizador ser o título de algum livro do acervo. Se retornar *false*, então é certo que não existe nenhum livro com esse título, no acervo.

```
Pesquisar Livros
[1] - Verificar existência de livro
[2] - Listar livros com títulos parecidos
[0] - Anterior
Selecione uma opção: 1
Título do livro: Drácula
Poderá existir um livro com o título Drácula

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

8. Screenshot da possível existência do livro

```
Pesquisar Livros
[1] - Verificar existência de livro
[2] - Listar livros com títulos parecidos
[0] - Anterior
Selecione uma opção: 1
Título do livro: Unhas Negras
Não existe nenhum livro com o título Unhas Negras

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

7. Screenshot da não existência do livro

4.5.2.2 Pesquisar livros com título parecidos

A listagem de livros com títulos parecidos é feita através de duas funções presentes na *Class* **RunLibrary**, que compõem todo o processo de obter as hashes mínimas e verificar a similaridade através da distância de jaccard. A *String* que o utilizador vai passar na consola vai passar como argumento na função **RunLibrary.getMinHashes()**. Esta função vai pegar nessa *String*, vai criar vários *shingles* através da função **MinHash.makeShingles()**, vai transformar os *shingles* em hashes, através da função **MathWorksFunctions.string2hash()**, vai criar centenas de hashes para cada *shingle* através da função **MinHash.getHashesForSingle()** e por fim vai retornar um array de hashes mínimas, de todas as hashes criadas anteriormente, através da função **MinHash.minHashes()**. Depois de este processo ser repetido para todos os livros do acervo, é calculada a distância de jaccard entre as hashes mínimas da *String* do utilizador e dos títulos dos livros. A função usada para o cálculo é a **RunLibrary.similarityValue()**, que vai calcular as interseções de ambos os arrays de hashes e vai dividir pela união dos mesmos. Por fim, se o resultado desta similaridade for favorável, então significa que em princípio os títulos são parecidos e é feito um print do livro.

```
Pesquisar Livros
[1] - Verificar existência de livro
[2] - Listar livros com títulos parecidos
[0] - Anterior
Selecione uma opção: 2
Título do livro: Gelo e Fogo
```

9. Screenshot da escolha da *String* pelo utilizador

```
61. A Muralha de Gelo, As Crónicas de Gelo e Fogo Vol 2 | George R. R. Martin - [Ficção Científica]
62. O Mar de Ferro, As Crónicas de Gelo e Fogo Vol 8 | George R. R. Martin - [Ficção Científica]
63. O Despertar da Magia, As Crónicas de Gelo e Fogo Vol 4 | George R. R. Martin - [Ficção Científica]
64. O Festim dos Corvos, As Crónicas de Gelo e Fogo Vol 7 | George R. R. Martin - [Ficção Científica]
65. A Tormenta de Espadas, As Crónicas de Gelo e Fogo Vol 5 | George R. R. Martin - [Ficção Científica]
66. A Glória dos Traidores, As Crónicas de Gelo e Fogo Vol 6 | George R. R. Martin - [Ficção Científica]
67. A Dança dos Dragões, As Crónicas de Gelo e Fogo Vol 9 | George R. R. Martin - [Ficção Científica]
68. Os Reinos do Caos, As Crónicas de Gelo e Fogo Vol 10 | George R. R. Martin - [Ficção Científica]
78. A Guerra dos Tronos, As Crónicas de Gelo e Fogo Vol 1 | George R. R. Martin - [Ficção Científica]

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

10. Screenshot do resultado da pesquisa com a string Gelo e Fogo

4.5.3 Pesquisar Autores

Na pesquisa de autores, é dada a possibilidade de o utilizador escolher entre verificar se existem livros de um autor em específico no acervo, e depois, caso exista, o utilizador tem a possibilidade de ver todos os livros desse autor.

```
Pesquisar Autores
[1] - Verificar se há livros do autor
[2] - Listar livros do autor
[0] - Anterior
Selecione uma opção: █
```

11. Screenshot do menu da pesquisa de autores

4.5.3.1 Verificar se há livros do autor

Nesta opção, é pedido ao utilizador para escrever o nome de um autor e, à semelhança do tópico **4.5.2.1 Verifica a existência de livro**, é usado o **BloomFilter** para verificar se um autor poderá ter livros seus na biblioteca. Caso isto seja verdade, o nome do autor é guardado como objeto da *Class* **RunLibrary**, para ser usado na opção referente ao tópico seguinte **4.5.3.2 Listar livros do autor**.

```
Pesquisar Autores
[1] - Verificar se há livros do autor
[2] - Listar livros do autor
[0] - Anterior
Selecione uma opção: 1
Nome do autor: J. K. Rowling
O autor J. K. Rowling poderá ter livros na biblioteca.

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

12. Screenshot da possibilidade de existência de um autor

```
Pesquisar Livros
[1] - Verificar existência de livro
[2] - Listar livros com títulos parecidos
[0] - Anterior
Selecione uma opção: 1
Título do livro: Unhas Negras
Não existe nenhum livro com o título Unhas Negras

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

13. Screenshot da não existência de um autor

4.5.3.2 Listar livros do autor

Se no opção do tópico anterior **4.5.3.1 Verificar se há livros do autor** existir a possibilidade de, realmente, o autor recebido do utilizador ter livros seus na biblioteca, então é usada a *Class* **MinHash**, e verificação de similaridade, à semelhança do que é feito nos tópicos **4.5.2.2 Listar livros com títulos parecidos** e **4.5.1.2 Listar livros por categoria**, para filtrar os livros desse autor e listá-los na consola.

```
Livros do autor J. K. Rowling
18. Harry Potter e a Ordem da Fénix | J.K.Rowling - [Literatura]
74. Harry Potter e a Pedra Filosofal Vol 1, Prémio Hans Christian Andersen 2010
| J. K. Rowling - [Ficção Científica]

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

14. Screenshot da listagem de livros do autor

4.5.4 Administração

Nesta secção, o utilizador tem acesso a quatro opções: Adicionar livro, Remover Livro, Requesitar livro e Devolver livro. Estas opções trabalham diretamente na edição do objeto da *Class* **Library** e dos objetos da *Class* **Book**.

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requesitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: █
```

15. Screenshot do menu da Administração

4.5.4.1 Adicionar livro

Esta opção recebe do utilizador as informações do livro que quer adicionar, como o título e o autor, e dá a escolher ao utilizador qual a categoria que este se enquadra. É usada a função **Book.addBook()**. O ID do livro é obtido automaticamente através da informação do ID do último livro do acervo.

```
Nome do livro:
Harry Potter e a Ordem da Fénix
Autor do livro:
J.K.Rowling

Categoria
[1] - Literatura
[2] - Thriller
[3] - Crianças
[4] - Ficção Científica
[0] - Anterior
Selecione uma opção: 1
CATEGORIA: Literatura
|||||Book 81 was registered successfully!
|||||Book 81 was successfully stored.
Livro 81. Harry Potter e a Ordem da Fénix | J.K.Rowling - [Literatura] adicionado com sucesso!

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: 
```

16. Screenshot da adição de um livro com sucesso

4.5.4.2 Remover livro

Esta opção recebe do utilizador o ID do livro que quer eliminar. É usada a função **Book.removeBook()**.

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requisitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: 2
ID do livro a remover: 5
|||||Book 5 had its registry erased successfully!
|||||Book 5 was successfully removed.
Livro 5. O Estrangeiro | Albert Camus - [Literatura] removido com sucesso!

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: 
```

17. Screenshot da remoção de um livro com sucesso

4.5.4.3 Requesitar livro

Esta opção recebe do utilizador o ID do livro que quer requisitar. É usada a função **Book.borrowed()** que verifica se o livro já foi, ou não requisitado, e caso esteja disponível para requisitar, é usada a função **Book.borrow()**. É também guardada e displayed a data de requesito do livro.

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requesitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: 3
ID do livro a requisitar: 60
Livro 60. [Requesitado] O Dia do Terramoto | Ana Maria Magalhães - [Crianças] requisitado!
Fri Dec 06 22:31:12 WET 2019

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

18. Screenshot do requisitar de um livro com sucesso

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requesitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: 3
ID do livro a requisitar: 60
Este livro já foi requisitado!

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

19. Screenshot de uma tentativa de requesito de um livro já requisitado

4.5.4.3 Devolver livro

Esta opção recebe do utilizador o ID do livro que quer devolver. É usada a função **Book.borrowed()** que verifica se o livro já foi, ou não requisitado, e caso esteja tenha sido realmente requisitado, é usada a função **Book.returnBook()**.

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requisitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: 4
ID do livro a devolver: 60
Livro 60. O Dia do Terramoto | Ana Maria Magalhães - [Crianças] devolvido!

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

20. Screenshot da devolução de um livro com sucesso

```
Admin
[1] - Adicionar livro
[2] - Remover livro
[3] - Requisitar livro
[4] - Devolver livro
[0] - Anterior
Selecione uma opção: 4
ID do livro a devolver: 2
Este livro nunca foi requisitado.

Voltar
[1] - Sair
[0] - Anterior
Selecione uma opção: █
```

21. Screenshot de uma tentativa de devolução de um livro não requisitado

4.5.5 Abandono da ação

Existem menus que aparecem com alguma frequência, que possibilitam o utilizador de sair do programa, ou simplesmente voltar para o menu principal.

```
Voltar  
[1] - Sair  
[0] - Anterior  
Selecione uma opção: 1  
A sair...
```

22. Screenshot do menu de retrocesso

Capítulo 5

Contribuição dos autores

A decisão em específico do que cada membro irá fazer ainda não foi realizada devido à fase prematura em que o trabalho se encontra, mas será sempre realizado pelos dois membros que compõem o grupo, quer seja de forma física ou por utilização de tecnologias que nos permitirá trabalhar à distância. Deste modo, podemos assumir desde já que a contribuição de cada membro do grupo neste projeto será de 50%.