Relatório Final - Sistema de Gestão de Stock

Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro

Marco Silva 84770, Raquel Rainho 84891
masilva@ua.pt, raquel.a.rainho@ua.pt

Capítulo 1

Sistema De Gestão de Stocks

1.1 Introdução

No âmbito da cadeira de Métodos Probabilístico para Engenharia Informática (**MPEI**), foi sugerida a realização de um projeto, sem tema predefinido, de modo a ser possível a demonstração da utilização conjunta de algoritmos probabilísticos para determinar a pertença a um conjunto (*set membership*) e determinação de itens similares (*finding similar itens*).

1.2 Estrutura

Este projeto foi devidamente estruturado de modo a que a sua leitura seja mais simples e o seu desempenho mais eficaz.

Para tal, foram criadas e desenvolvidas múltiplas classes, das quais: 3 dedicadas a testes, 2 remetentes a leitura e escrita de ficheiros e as restantes ao funcionamento do programa.

Testes

- BloomTest.java
- MinHashTest.java
- Test.java (testa o programa em geral)

Ficheiros

- FileRdWr.java
- ConteudoFicheiros.java

Funcionamento do Programa

- Cliente.java
- ClienteSet.java

- Compras.java
- ComprasSet.java
- CountFilter.java
- DistanceFilter.java
- Jaccard.java
- Loja.java
- LojaInterface.java
- LojaSet.java
- MinHash.java
- Supermercado.java (*Main*)

1.3 Implementação

Primeiramente, optou-se pela utilização de um BloomFilter, no entanto, após discussão do tema do projeto com o professor da aula prática, foi-nos aconselhado utilizar um CountFilter, uma vez que, para armazenar o stock de um supermercado, seria mais simples e eficaz, dado que, para além de armazenar o produto é também possível armazenar uma certa quantidade do mesmo. Além do método de inserção de um CountFilter normal, foi desenvolvido um método nesta classe (*CountFilter.java*) onde são passados, como argumentos, um produto e quantidade respetiva, ao invés de ser passado unicamente o produto, tornando, assim, o CountFilter mais eficiente. Para o correto funcionamento desta classe, utilizou-se uma função hash, semelhante à função já conhecida *string2hash*, que, recorrendo ao que foi lecionado nas aulas, funciona como k funções hash (neste caso k = 5, determinado pela formula apresentada numa aula teórica e assumindo que n (quantidade total de produtos numa loja) seria cerca de 750 e m (número de diferentes produtos) seria 100);

Para a determinação da semelhança de itens foram desenvolvidas 3 diferentes classes:

1. (**MinHash.java**) -> Aqui, é criado um array de Sets por cliente e, de seguida, calculada a menor de 5 hashes para cada produto que esse mesmo cliente comprou.
2. (**Jaccard.java**) -> Nesta classe, é calculada a distância entre cada um dos pares de clientes possíveis através da minHash calculada na classe anterior e que retorna, também, as distâncias por cada cliente.
3. (**DistanceFilter.java**) -> Depois do cálculo da distância, é nesta classe que são escolhidos os pares, consoante o *threshold* que foi passado na inicialização da classe. Esta retorna os pares de clientes e a distância entre os mesmos.

Para que as classes que foram referidas anteriormente possam ser usadas, foram, então, desenvolvidas as classes correspondentes ao Sistema de Gestão de Stocks em si. Para tal, foram necessárias classes para criar e guardar os produtos em cada supermercado e para criar e guardar novos clientes. Além disto, foram desenvolvidas classes para que o utilizador do programa possa ser um "gestor/cliente"de um supermercado à sua escolha, tendo a disponibilidade de:

0. Terminar o programa -> Figura 1.1
1. Efetuar uma compra -> Figura 1.2
2. Repor o stock dos supermercados -> Figura 1.3
3. Registar um novo supermercado -> Figura 1.4
4. Registar um novo produto -> **??**

1.4 Instruções

Aqui serão expostas as diferentes opções, já referidas, que o utilizador do programa terá, tal como o resultado da escolha de cada uma.

De notar que os resultados podem diferir dos apresentados mais à frente.

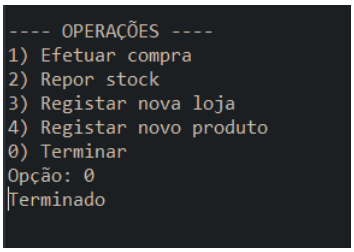

Figura 1.1: Terminar Programa

```
---- OPERAÇÕES ----
1) Efetuar compra
2) Repor stock
3) Registar nova loja
4) Registar novo produto
0) Terminar
Opção: 1
Insira nome da loja que pretende frequentar (Continente, Pingo-doce, Intermarch?):Continente
Insira o seu NIF:
296793220
Fim de MinHash
Fim de Jaccard
Fim de Distance
Cliente semelhante: 478856656
Poderá querer comprar os seguintes produtos:
R?cula
Condicionador
Queijo ralado
Alface
Gordura vegetal
Leite
Palmito
Espinafre
Polpa de tomate
Hamb?rguer
Polvilho
R?cula
Hamb?rguer
Insira o numero de produtos que pretende comprar:
2
Produto: vassoura
Quantidade: 1
Produto: baunilha
Quantidade: 2
Compra registada com sucesso!
```

Figura 1.2: Efetuar Compra

```
---- OPERAÇÕES ----
1) Efetuar compra
2) Repor stock
3) Registar nova loja
4) Registar novo produto
0) Terminar
Opção: 2
Stock Reposto!
```

Figura 1.3: Repor Stock

```
---- OPERAÇÕES ----
1) Efetuar compra
2) Repor stock
3) Registar nova loja
4) Registar novo produto
0) Terminar
Opção: 3
Insira nome da loja que pretende registar: MiniPreço
Insira o numero de produtos que pretende ter na loja: 5
Produto: chocolate
Quantidade: 3
Produto: bolacha
Quantidade: 4
Produto: baunilha
Quantidade: 1
Produto: pessego
Quantidade: 5
Produto: camarao
Quantidade: 10
Loja registada com sucesso!
```

Figura 1.4: Registar Supermercado

1.5 Funções Hash

De seguida, serão apresentadas imagens que mostram ambas as funções hash utilizadas neste projeto e, ainda, um método que, apesar de implementado, não é utilizado no âmbito do funcionamento do projeto. Este calcula a probabilidade de falsos positivos para o CountFilter em questão.

```
int hash(String s){
    int hash = 16573;

    for(int i = 0; i < s.length(); i++){
        hash = (int) ((hash*i + s.charAt(i))%(Math.pow(2, 32)-1));
        if(hash < 0)
            hash = -hash;
    }

    return hash;
}
```

Figura 1.5: Função Hash do CountFilter.java

```
private int[] hash(String s, int k){
    int[] h = new int[k];

    for(int j = 0; j < k; j++){
        int hash = 2017+k;
        s = s.concat(""+j);

        for(int i = 0; i < s.length(); i++){
            hash = (int) ((hash*31 + s.charAt(i))%(Math.pow(2, 32)-1));
            if(hash < 0)
                hash = -hash;
        }
        h[j] = hash;
    }
    return h;
}
```

Figura 1.6: Função Hash da MinHash.java

```
double falsePositives(){
    double count = 0;

    for(int i = 0; i < bloom.length; i++){
        if(bloom[i] >= 1)
            count++;
    }
    double d = (double) (1.0/n);
    count = count*k;
    double x = Math.pow(1-d,count);
    x = 1 - x;
    x = Math.pow(x, k);
    return x;
}
```

Figura 1.7: Calculo de falsos positivos CountFilter.java

1.6 Validação

Ao longo do projeto foram-se realizando testes às respetivas classes, de forma a garantir o correto funcionamento destes blocos em individual e conjunto, para que não houvesse comprometimento do projeto final, no entanto devido às diferenças que possam existir entre as máquinas que correm este projeto o mesmo funcionar de forma mais ou menos fluida.

1.7 Conclusão

O projeto realizado cumpre tanto com os requisitos impostos no começo como também os objetivos impostos por nós mesmos (apesar de não funcionar totalmente como seria de esperar), ainda que tenham sido necessárias várias alterações às nossas ideias iniciais, sempre de modo a aumentar a eficiência e a simplicidade da resolução dos problemas que fomos enfrentado ao longo do desenvolvimento do mesmo.

1.8 Divisão do trabalho

O projeto foi desenvolvido em conjunto pelos elementos do grupo, tendo estes discutido ideias e feito a devida implementação. Sendo assim pode-se afirmar que tanto Marco Silva como Raquel Rainho têm uma participação de 50% cada para o aspeto final do projeto.