

Relatório Trabalho Prático

8 - Processamento de *reviews* de jogos

Hugo Paiva de Almeida - 93195

Mário Silva - 93430

Este trabalho prático tem como objetivos o desenvolvimento de dois módulos, um *Bloom Filter* e uma *MinHash* para descobrir itens que sejam semelhantes, usando-os numa aplicação para resolução de um problema concreto. O problema escolhido assenta no processamento de *reviews* de jogos, que deverá garantir, no mínimo, reconhecer se um dado jogo tem *reviews* (e quantas), e determinar *reviews* semelhantes. Foi ainda desenvolvido um módulo adicional, o *Algoritmo LSH* com o objetivo de diminuir o tempo de execução na verificação de similaridades entre dados. Desta forma, e tendo em conta o tema do trabalho, foi usado um dataset que contém, além de várias informações acerca de um certo jogo nesse conjunto de dados, as *reviews* e informações deste.

Módulos Principais

Bloom Filter (adaptado)

Foi utilizada uma derivação do Bloom Filter, o *Counting Bloom Filter* uma vez que, para além de permitir saber se um certo elemento está num conjunto, permite ainda obter o número de vezes que o elemento foi adicionado ao *Counting Bloom Filter*. Desta forma, os seguintes dados foram obtidos:

- Verificar se um developer/publisher produziu um certo jogo;
- Verificar se um jogo tem *reviews* e quantas;
- Verificar se um jogo tem uma certa linguagem e se essa linguagem está disponível em texto, áudio ou ambos.
-

MinHash

Recorreu-se à implementação de uma *MinHash* para permitir a verificação da similaridade entre dados, tendo sido verificados:

- Se um dado jogo é similar a outro;
- Se um nome de jogo é similar ao nome de outro;
- A similaridade entre *reviews* de um certo jogo.

Algoritmo LSH

O *Algoritmo LSH* agrupa a assinatura de um certo elemento em bandas e calcula um valor *hash* para cada banda. Assim, reduz-se a matriz da *MinHash* consideravelmente, tendo em conta o número de bandas e, desta forma apenas são realizadas comparações na matriz da *MinHash* quando dois elementos forem considerados pares candidatos a serem similares. Para tal, basta terem um valor igual na matriz *LSH*. A maior vantagem é que o *LSH* permite diminuir bastante o número de comparações necessárias para calcular a similaridade entre elementos e, conseqüentemente, o tempo de execução.

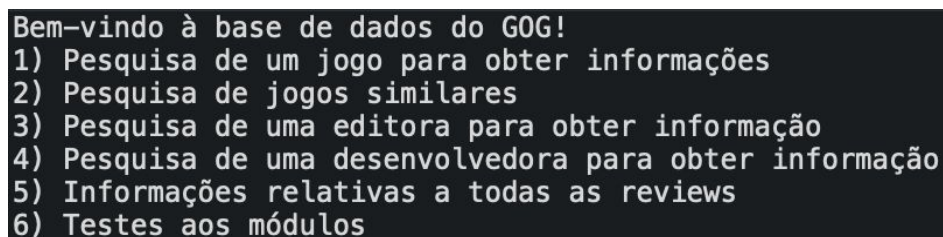
Este algoritmo foi utilizado para fazer comparações de dados de um conjunto consideravelmente grande, com o objetivo de diminuir o tempo da operação sem perder muita eficácia. Neste caso, permite otimizar o processo de:

- Verificar a similaridade entre as reviews de todo o *dataset*;
- Pesquisar por frases semelhantes ou palavras inseridas por todo o *dataset*;
- Percorrer todas as reviews de forma a encontrar e detetar spam.

Implementação

No início da implementação foram criados objetos do tipo *Game*, *Review* e *Language* para facilitar o armazenamento dos dados.

Foi implementado um conjunto de *features* que podem ser usadas diretamente na consola do programa, compilando as classes e executando a *Main*. Os ficheiros necessários para a execução encontram-se [aqui](#), sendo que os ficheiros texto utilizados na *MinHash* devem estar na *root* do projeto e o *dataset* "*games.json*" deve estar na pasta *lib* do projeto.



```
Bem-vindo à base de dados do GOG!
1) Pesquisa de um jogo para obter informações
2) Pesquisa de jogos similares
3) Pesquisa de uma editora para obter informação
4) Pesquisa de uma desenvolvedora para obter informação
5) Informações relativas a todas as reviews
6) Testes aos módulos
```

Figura 1 - Seleção de *features* da aplicação

1. Pesquisa de um jogo para obter informação

Nesta opção, é pedido para introduzir o nome de um jogo. Este pedido serve para, com o nome introduzido, ser feita uma espécie de pesquisa, apresentado o nome dos jogos com similaridade superior a 30% em relação ao nome introduzido. Esta pesquisa recorre ao módulo da *MinHash*, criando *shingles* de tamanho 2 para todos os nomes de jogos do *dataset* e a respetiva *hash* mínima, para cada função *hash*, considerando 100 funções, de todos os *shingles* do nome em questão. Após isto, são criados *shingles*, do mesmo tamanho, para o nome introduzido e realizada a respetiva comparação com os dados do *dataset*. Se não existir nenhum nome de jogo com similaridade superior a 30%, é impressa a mensagem que nenhum jogo foi encontrado, em caso contrário, é pedido ao utilizador para selecionar um dos jogos impressos.

Feita a pesquisa e seleção do jogo, é impresso o número total de reviews que esse jogo possui. Isto é alcançado, ao criar um *Counting Bloom Filter*, na leitura dos dados do programa, ao qual são adicionados os jogos que têm reviews e utilizando o respetivo método de verificação.

Posteriormente, são impressas 3 opções: informações sobre das reviews do jogo, informações sobre as línguas do jogos e informações gerais do jogo. A primeira permite visualizar informação acerca das reviews do jogo. Dentro destas informações, é possível observar reviews consideradas como *spam*, ou seja, usando o mesmo método para obter a similaridade, são impressas todas as reviews do jogo com similaridade superior a 95%, entre si e que sejam do mesmo utilizador. É possível, também, visualizar as reviews similares entre si, sem *spam*. Isto foi obtido, também, através do método anterior mas desta vez, perguntando ao utilizador qual o grau de similaridade que pretendia. Para se filtrar o conteúdo com *spam*, foi usada a mesma forma que na primeira opção, retirando essas reviews do conteúdo impresso. É permitido, ainda, introduzir, novamente, o grau de similaridade e, através do método que tem sido recorrido, é comparado o que é introduzido pelo utilizador com as reviews do jogo, imprimindo as que têm similaridade superior à introduzida. Por último, permite visualizar os utilizadores que realizaram *spam*, dentro das reviews deste jogo. Isto é, novamente, obtido comparando todas as reviews, obtendo o nome de utilizador daquelas que têm similaridade superior a 95%.

As informações sobre as línguas do jogo são obtidas selecionando a língua, mediante a impressão de todas as línguas disponíveis no *dataset* e, usando 3 tipos de *Bloom Filters* para cada língua, que indicam se a língua está presente no jogo, se tem texto e se tem áudio no jogo. Desta forma, é dada a informação se a língua está presente no jogo e se apenas está em texto, áudio ou ambos.

Por fim, são impressas informações gerais do jogo, obtidas durante a leitura e tratamento dos dados, no início do programa.

2. Pesquisa de jogos similares

A pesquisa de jogos similares utiliza uma técnica já usada na opção anterior para a pesquisa e seleção de um nome do jogo introduzido. A nova *feature* introduzida foi, na verdade, a visualização de jogos similares em relação ao introduzido. Isto foi obtido usando o módulo da *MinHash* mas, desta vez, para calcular a *hash* mínima para as 100 funções *hash*, foram tidas em consideração várias características dos jogos como: os géneros, nome, preço, classificação, etc. As características que continham várias palavras foram transformadas em *shingles*. Desta forma, é possível visualizar os jogos similares, tendo em conta que só são impressos os que têm similaridade superior a 50%.

3. Pesquisa de uma editora para obter informação

Voltando a utilizar a técnica da pesquisa mas, desta vez para todas as editoras, tendo em conta a editora introduzida, vai ser apresentado o nome das editoras com similaridade superior a 30% em relação ao nome introduzido. Desta forma, volta-se a recorrer ao módulo da *MinHash*, criando *shingles* de tamanho 2 para todos os nomes de editoras do *dataset* e a

respetiva *hash* mínima, para cada função *hash*, considerando 100 funções, de todos os *shingles* do nome em questão.

Com a pesquisa e seleção da editora feita, é apresentada a opção para verificar se um jogo foi distribuído pela editora. Voltando a realizar a pesquisa e seleção do jogo, à semelhança das outras opções, foi possível fazer esta verificação criando um *Bloom Filter*, na leitura dos dados do programa, ao qual são adicionados os jogos que são distribuídos pela editora em questão e utilizando o respetivo método de verificação.

4. Pesquisa de uma desenvolvedora para obter informação

Esta opção utiliza todas as ferramentas utilizadas na última opção descrita, com a única diferença de serem utilizadas as desenvolvedoras do jogo para obter as informações.

5. Informações relativas a todas as reviews

Quanto às informações relativas a todas as *reviews*, de todos os jogos, foi decidido a utilização do *Algoritmo LSH*, juntamente com o módulo da *MinHash*, com o objetivo de diminuir o tempo de execução na verificação de similaridades entre as *reviews*. Como a matriz com as assinaturas de cada *review* é necessária, decidiu-se escrever, previamente, para um ficheiro esta mesma, juntamente com os todos os valores usados no cálculo desta matriz. É então, inicialmente, obtida esta matriz e os valores da *HashFunction* através da leitura dos ficheiros.

São apresentadas duas opções, mostrar os utilizadores considerados como *spammers*, no *dataset* todo e, pesquisa por *review* similar a uma frase introduzida, também no *dataset* todo.

Para a primeira opção é obtido um *HashSet* de todas as *reviews* com grau de similaridade superior a 95% e que sejam do mesmo utilizador. Depois, apenas é percorrido o *HashSet* anterior e imprimindo, assim, o utilizador correspondente a cada *review*.

Para a segunda opção, é usado um método que calcula os *shingles* da frase inserida e de seguida a respectiva assinatura utilizando os mesmos valores da *HashFunction* usada para a matriz guardada no ficheiro. Posteriormente, é reduzida a assinatura em bandas, executando-se o algoritmo *LSH*, para apenas comparar estes valores *hash* da frase com as *reviews* todas.

6. Testes aos módulos

Por último, esta opção vai permitir ao utilizador realizar testes aos módulos implementados. São apresentadas duas opções, testes ao "*Counting Bloom Filter*" e "*Testes à MinHash e MinHashLSH*". A primeira irá utilizar o método *bloomTests()* do objeto da classe *Tests*, anteriormente criado. Este método cria um *Counting Bloom Filter* e insere *strings* geradas automaticamente e de forma aleatório. Posteriormente, são criadas o mesmo número de *strings*, também geradas de forma automática e aleatória e, visto que a probabilidade das *strings* serem iguais é muito baixa, se existir alguma destas últimas *strings* no *Bloom Filter*, estas são consideradas como colisões. É, também, calculada a distribuição dos elementos, de forma a verificar se a *Hash Function* usada é adequada, sendo que esta foi calculada em

percentagem, a partir da divisão da média dos elementos inseridos pelo número total de elementos do filtro, considerando-se uma distribuição uniforme se estiver entre 40% e 60%. Por fim, são feitos testes com o *dataset* e alguns elementos deste.

A segunda opção irá utilizar o método *minTests()* do objeto da classe *Test*, anteriormente criado para comparar tempos de execução entre a *MinHash* e a *MinHash* juntamente com o *Algoritmo LSH*.

Resultados Obtidos

```
-----Testes ao BLOOM-----
1) Inicializar e Inserir Strings Randoms ao Bloom...
2) Criar Outro Conjunto de Strings...
3) Verificar se as strings do segundo conjunto pertencem ao bloom...

    Dados Obtidos do Bloom:
Distribuição dos elementos: Uniforme
Número de falsos positivos: 3
Probabilidade de falsos positivos: 3.0E-4
Probabilidade teórica de falsos positivos: 1.3624906795958706E-21

-----BLOOM COM NOSSO DATASET-----

Time elapsed to create bloom (s): 1.02044E-4 seconds.

    Dados do DataSet:
Número de reviews reais de Enclave: 68
Número de reviews reais de Galactic Civilizations: 0
Número de reviews de Overlord Raising Hell: 31

    Dados do Bloom:
Enclave tem reviews? true
Galactic Civilizations tem reviews? false
Overlord Raising Hell tem reviews? true

Enclave tem quantas reviews? 68
Galactic Civilizations tem quantas reviews? 0
Overlord Raising Hell tem quantas reviews? 31

    Eliminar o jogo Enclave do Bloom:
Enclave tem reviews? false

    Teste de Falsos Positivos:
Numero de elementos inseridos no bloom (sem contar com o jogo eliminado): 2246,0
Numero de elementos que estão no bloom: 2250,0
Numero de colisoes: 2,0
Probabilidade de falsos positivos: 8.904719501335708E-4
Probabilidade teórica de falsos positivos: 0.00843688292790619
```

Figura 2 - Resultados dos testes ao *Counting Bloom Filter*

```
Print de reviews para os primeiros 80 jogos com similariedade superior a 90% com MinHash e com MinHashLSH.
Tempo de demora de criação de MinHash: 9.623788064 segundos.
Tempo de demora de criação de MinHashLSH: 5.152892697 segundos.
Tempo de procura de similares com MinHash: 0.602125534 segundos.
Tempo de procura de similares com MinHashLSH: 0.117238153 segundos.
```

Figura 3 - Resultados dos testes à *MinHash* e *MinHashLSH*, para os primeiros 80 jogos

Conclusão

Em suma, este projeto permitiu entender o funcionamento dos vários módulos tal como a sua fácil derivação. Apesar de na implementação final ter sido usada a *MinHash* com as *hashes* mínimas guardadas numa matriz, esta também foi, inicialmente, desenvolvida guardando estas *hashes* num objeto. Tal como a *MinHash*, a *Hash Function* usada não foi a primeira escolha do grupo, sendo que sofreu algumas alterações relativamente à primeira versão, de modo a otimizar o seu funcionamento.

Foi, também, concluído que recorrendo ao *Algoritmo LSH*, o tempo de procura de similaridades baixa significativamente, pelo que deverá ser uma ótima implementação para trabalhos com grandes *datasets*.