Relatório – Trabalho Prático de MPEI

Pedro Gonçalves - 88859

Renato Valente - 89077

Módulos:

1) Counting Bloom Filter: *CountingBloomFilter.java*
2) Contador Estocástico: *ContadorEstocastico.java*
3) MinHash: *MinHash.java*
4) Projeto Final: *JogoDosDados.java*

1. Counting Bloom Filter

O nosso Counting Bloom Filter, guardado no package Teste_BloomFilter é composto por 3 ficheiros, *CountingBloomFilter.java*, *HashFunction.java* e *TesteCountingBloomFilter.java.*

1.1 CountingBloomFilter.java

Este é o ficheiro que implementa o Counting Bloom Filter.

O seu construtor recebe o número aproximado de elementos que vamos guardar e calcula o tamanho e o número de hash functions ideais.

Possui uma função para o inicializar (initialize) que nos dá 3 números gerados aleatoriamente, depois colocados na hash function, e uma função para inserir elementos no bloom filter (insert) que passa cada elemento pela quantidade de hash functions calculadas e depois é inserido num determinado index.

A função isMember, verifica se o elemento passado como argumento pertence ou não ao bloom filter e, no caso de pertencer, quantas vezes foi colocado.

Por fim, a função delete, permite-nos apagar o elemento passado como argumento, verificando antes se ele é ou não membro.

1.2 HashFunction.java

Implementa a nossa Hash Function, o seu construtor recebe o número de hash functions ideais.

É composto por 3 funções, hashFunc, que recebe a string que queremos passar pela hash function e os valores aleatórios calculados ao inicializar o Counting Bloom Filter (a e b), e retorna o índex em que a String é colocada.

As funções isPrime verifica se um número é primo e a nextPrime devolve o número primo imediatamente antes do valor do tamanho do bloom filter (o ideal para a hash function).

1.3 TesteCountingBloomFilter.java

O programa de teste.

Começamos por inserir alguns valores no Counting Bloom Filter, verificamos se dois deles são membros (um é o outro não) e removemos 3 valores em que 2 são iguais (dois que são membros e o outro não), e tudo acontece como o esperado!

2. Contador Estocástico

O contador estocástico está guardado no package Teste_ContadorEstocastico e é composto pelo *ContadorEstocastico.java*, *TesteContadorEstocastico.java* e pelo ficheiro de texto *Numbers.txt*.

2.1 ContadorEstocastico.java

É o programa que implementa o contador estocástico, o seu construtor recebe como argumento a percentagem com que queremos que conte um determinado elemento.

A função writeNStringsToFile escreve um dado número de strings, neste caso sequências de 5 números de 1 a 6 aleatórias (geradas pela função generateRandomString) num ficheiro, tanto o ficheiro como o número total de strings são passados como argumento à função.

A função readFromFile lê o ficheiro criado e guarda as sequências todas numa Array List para proceder ás contagens.

Por fim, as funções timesEqualTo comparam a sequência passada como argumento, ou uma sequência aleatória gerada automaticamente caso não passemos nada como argumento com as sequências todas guardadas no ficheiro e devolvem o número de vezes que a sequência aparece no ficheiro considerando a probabilidade que passamos ao construtor.

2.2 TestContadorEstocastico.java

No nosso programa de teste, começamos por definir o número de strings que queremos gerar, criar o ficheiro e definir a probabilidade de contagem que queremos.

Em seguida testamos quantas vezes o número gerado aleatoriamente e o número introduzido aparecem considerando a probabilidade de contagem.

2.3 Numbers.txt

É o ficheiro de texto que contém 1.000.000 strings aleatórias.

3. MinHash

Está guardada no package Teste_MinHash e contém os ficheiros *MinHash.java*, *TesteMinHash.java* e os ficheiros de texto, *RegistoDados_4iguais.txt*, RegistoDados_10iguais.txt e RegistoDados_13iguais.txt

3.1 MinHash.java

É o programa que implementa a MinHash, o seu construtor cria um array para 200 Hash Functions, inicializa os valores de “a” e “b” (função initialize) de cada Hash Function e cria um array bidimensional para os 2 Sets e com as 200 Hash Functions.

Em seguida, a função similares, verifica a similaridade dos 2 Sets, percorrendo um de cada vez e aplicando as 200 Hash Functions aos 13 elementos de cada Set, calculando o mínimo que é a o valor mínimo que se obtém dos 13 valores passados por cada Hash Function.

No Final, a função CompararMinSimilares, vai comparar o valor minimo das Hash Functions criadas por cada Set e se forem iguais conta o número de MinHashs Idênticos, depois aplicando a fórmula, obtemos a distância de Jacard.

3.1 TesteMinHash.java

No nosso programa de teste, vamos comparar 3 Sets diferentes um de cada vez (LerDados_4iguais, LerDados_10iguais, LerDados_13iguais), com outro Set fixo inicializado pela função criarSetAtual.

A função readDataBase vai ler o Set de cada ficheiro acima referido. Depois vamos criar as 3 MinHashs com cada um destes Sets e comparar com o Set fixo, podendo depois observar no terminal, com a função paraImprimir, uma tabela com os mínimos de cada HashFuntion em cada Set, mas só mostramos os 20 primeiros em vez dos 200, porque chega para analisar, também aparece para cada ficheiro o valor dos MinHash Identicos (max. 200) e a distância de Jaccard, que quando mais idênticos forem as MinHahs menor será a distJaccard.

4. JogoFinal

4.1 JogoDosDados.java

-> É o programa principal. É o programa para correr o jogo!!

4.2 Dices.java

Serve para rodar os dados aleatoriamente.

4.3 FazerJogadas.java

Não é preciso correr, apenas serviu para simular a criarmos 10.000 jogos aleatórios em vez de estarmos a jogar 10.000 vezes.

Para a junção dos módulos decidimos fazer um jogo em que consiste rodar dados 3 vezes, fixar dados durante essas 3 vezes e formar sequências conforme as opções disponibilizadas em baixo no jogo. Cada uma destas opções têm pontos definidos e o objetivo do Jogo é acumular o maior número de pontos para ficar com a melhor pontuação.

A pasta JogoFianlJogadas, contém todos os Jogos feitos e para isso nós simulamos 10.000 jogos.

Para implementar o BloomFilter no jogo, ao inicializar o Jogo, este vai guardar no BloomFilter os valores dos últimos 500 jogos e como cada jogo contém 13 valores, vai inserir no BloomFilter 6500 valores. Ao decorrer do jogo, quando se roda os dados 3 vezes podemos escolher uma opção disponibilizada abaixo. Depois dessas 3 vezes ficamos um valor que vai ser comparado com os 6500 valores através da função isMember. Do lado direito do jogo irá aparecer se o valor Pertence ou Não Pertence e quantas vezes aparece. Para confirmar que o Pertence/Não Pertence está a funcionar

bem, implementamos um **Teste_BloomFilter_ComOJogo** em que vai comparar o valor saído com um ficheiro com valores de 11111 a 26666.

Para implementar o Contador Estocástico, contamos as vezes que são inseridos valores no BloomFilter e como visto anteriormente, deviam ser inseridos 6500 valores, e com o contador estocástico podemos obter uma variação de 200 valores (6500 +/- 200), esta informação aparece ao inicializar o jogo no seu topo.

Para implementar a MinHash no jogo, ao chegarmos ao final do jogo, vamos comparar o Set dos 13 valores que nos deu no Jogo atual com cada Set de todos os jogos feitos, que são cerca de 10.000, dizendo-nos qual desses jogos todos foi o jogo mais parecido com o jogo atual, também nos dirá qual o numero de MinHashs Idênticas e a distância de Jaccard.

Por fim o jogo atual é guardado na pasta JogoFianlJogadas e está pronto para ser comparado com jogos futuros.

Teste_BloomFilter_ComOJogo

Como já foi explicado anteriormente, serve para podermos comprovar que o BloomFilter está a funcionar com o Jogo.

O fazerRegisto.java apenas serve para fazer o registo dos valores entre 11111 e 26666 para um ficheiro texto, RegistoDados.txt.