Relatório

Francisco Jesus – 89084 Gonçalo Freixinho – 89251

Neste relatório será explicado o trabalho desenvolvido: como o usar e o que foi realizado, os testes que foram feitos e os resultado obtidos dos mesmos, e por fim a aplicação que usa o três módulos criados.

Os módulos e a aplicação conjunta realizados não são interativos, ou seja, não necessitam da participação do utilizador para funcionar, basta só iniciar um dos testes, para obter os resultados.

## Bloomfilter

Para o módulo do bloomfilter foi feita uma classe que tem o um construtor que aceita dois valores inteiros, o tamanho do bloomfilter e o número de funções hash que serão usadas. Dentro desse construtor é criado um objeto do tipo InitHashFunction que guarda os valores de “a”, “b” e de “p”, que são usados na universal hash function. Depois, a classe tem duas funções, uma para inserir um valor no bloomfilter e outra para procurar um elemento. Ambas são usadas só para strings que convertem a string para um valor inteiro que depois é usado na universal hash function.

Para os testes realizados, primeiro foram introduzidas poucas palavras para um bloomfilter pequeno para verificar se o bloomfilter fazia a contagem corretamente e pelos resultados obtidos o bloomfilter faz bem a contagem.

ola - 1

oi - 1

gold - 3

silver - 2

bronze- 1

llsdjfkw - 0

triple - 1

his - 1

the – 1

Depois foi usado outro counting bloomfilter para contar as palavras de um texto em inglês e foram filtradas as palavras que aparecem mais de 100 vezes.

a - 136

am - 225

and - 232

big - 232

foe - 537

he - 128

his - 140

kid - 232

law - 140

met - 140

of - 225

old - 537

the - 537

to - 155

win – 140

E estes resultados apresentam o que se esperava, que era palavras como “the”, “and”, “a”, “am”, apareçam mais num texto em inglês. Contudo estes resultados apresentam palavras com o mesmo número de ocorrências , e daqui pode-se aferir que a função hash talvez não seja a melhor para grandes quantidades de palavras diferentes.

# Itens Similares

Para este módulo foi feita outra classe, que no seu interior tem dois construtores, que aceitam o número de funções hash e um objecto do tipo InitHashFunction para obter os valores para a universal hash function. A diferença nos contrutores está que um aceita uma string e outro aceita um array de strings, este último é usado na aplicação. Depois temos três funções uma cria assinaturas onde também, no ínicio, cria shingles, esta função é chamada de SignatureMaker1 e recebe um valor inteiro que define o tamanho das shingles e que depois cria um array com essas shingles, que depois é usado para passar pela a hash function para criar a assinatura da string. Na função SignatureMaker2 é usado arrays de strings e o que faz é só a parte de da criação da assinatura. Por fim a função distance compara duas assinaturas e calcula a sua similariedade, ao comparar um elemento de um assinatura com outro que está na mesma posição na outra assinatura e incrementa a variável de elementos iguais e depois dessa contagem divide o número de elementos iguais pelo número de funções hash.

Passando agora aos teste deste módulo. Foram criadas 6 strings, duas de maior comprimento, que são muito parecidas e 4 de comprimento pequeno, em que 2 são iguais, outra é semelhante com essas duas e outra que é totalmente diferente. Os resultados obtidos foram os esperados, ou seja, as que são iguais apresentam o valor máximo de similariedade, as parecidas apresentam valores elevados e as diferentes apresentam valores baixos.

Também foram feitos testes para textos maiores, em que dois deles são muito parecidos e outro é um texto diferente. Os resultados foram, para os semelhantes, o esperado, contudo para os diferentes esperava-se menos, pode-se dever outra vez à hash function, possivelmente.

f1 - f2 -> 1.0

f1 - f3 -> 0.11

f1 - f4 -> 0.01

f1 - f5 -> 0.01

f1 - f6 -> 0.72

f2 - f3 -> 0.11

f2 - f4 -> 0.01

f2 - f5 -> 0.01

f2 - f6 -> 0.72

f3 - f4 -> 0.01

f3 - f5 -> 0.01

f3 - f6 -> 0.12

f4 - f5 -> 0.77

f4 - f6 -> 0.01

f5 - f6 -> 0.01

t1 - t2 -> 0.92

t1 - t3 -> 0.56

## Contador Estocástico

Quanto ao contador estocástico foi feita uma classe que inclui um contrutor que necessita de um valor booleano que especifica que tipo de contador é, isto é, se é um contador que conta com uma probabilidade de 0.5 ou se é um contador logaritmico, em que sua probabilidade de contagem vai diminuindo. Depois tem dois métodos que incrementam o contador, um em que se atribui logo o valor que se quer contar, que foi feito meramente para tornar os testes mais fáceis e este só faz contagem com probabilidade de 0.5. O outro método já incluí as duas maneiras de contagem. Numa quando a contagem é feita incrementa-se o contador por dois e no outro método a probabilidade vai diminuindo e o incremento aumenta.

Para os testes fizeram-se 4 contagens diferentes, uma inicial em que conta até 1000, outra que vai até 1 000 000 com o contador de probabilidade 0.5 e outra contagem até 1 000 000 com o contador logarítmico. A última contagem foi do número de palavras de um texto com 6093 palavras com um contador de probabilidade 0.5. Os resultados obtidos foram os esperados com valores próximos dos mencionados anteriormente.

contador= 964

contador 2 = 1001358

contador 3 = 950131

word count: 6093

contador 4: 6018

## Aplicação conjunta

A aplicação conjunta tem como objetivo aplicar os três módulos criados anteriormente ao mesma situação. Neste caso foram aplicados a listas de top 100 romances(livros) de websites diferentes. Com esta aplicação procura-se descobrir os livros mais comuns entre as listas e assim criar uma lista que irá conter os livros que aparecem mais entre as listas, que pode ser, posteriormente, usada para procurar listas que contenham estes livros ou então procurar listas que sejam totalmente diferentes, caso se queira uma lista que contenha livros menos conhecidos. Também com esta aplicação procurou-se saber que listas são as mais semelhantes.

Para isso foi usado o counting bloomfilter para criar a lista de livros mais comuns e as funções do módulo 3 (itens similares) para encontrar as listas mais semelhantes. O contador estocástico tem somente a função de contar o número total de livros e o número de livros sem repetições, no entando, caso o número de listas fosse enorme, poderia-se usar o contador estocástico para contar o número de vezes que cada livro aparece, esta é uma desvantagem do contador para esta situação, pois utilizam-se poucas listas e é preciso de uma maior precisão. As funções do módulo 3 também são usadas para comparar a lista dos livros mais comuns com cada das listas que a formaram para obter a sua similariedade e assim saber que listas realmente contribuíram para a sua criação e aquelas que são as que apresentam mais livros diferentes.

Os métodos usados nesta aplicação seriam mais vantajosos caso se usa-se uma quantidade maior de listas.