***ADEETC*-LEIC**

2017/18i

CCD

Exercício 1:

Utilizando um conjunto de 3 métodos estáticos:

-> **calculateFreq** que recebendo um Map<Byte, Intiger> irá retornar um Map<Byte, Double> em que a key são os bytes ocorridos no ficheiro lido e o value a probabilidade do símbolo.

-> **calculateEntrop** que recebendo um Map<Byte, Double>, key é o byte do ficheiro e value a probabilidade, aplica a formula da entropia sobre os valeus do mapa recebido como argumento.

-> **readFile** que recebendo uma String como argumento irá ler o ficheiro com o nome do argumento e irá retornar um mapa com Map<Byte, Intiger> em que cada símbolo ocorrido no ficheiro corresponde à key e o value ao numero de ocorrências desse símbolo.

Aplicando a aplicação a um ficheiro de texto com o seguinte texto “aaabbbbbbc” cujo a probabilidade será para os símbolos:

$$\mathrm{P(a)} = \frac{3}{10}$$

$$\mathrm{P(b)} = \frac{6}{10}$$

$$\mathrm{P(c)} = \frac{1}{10}$$

$$E = \sum_{n=S}^{Sn} \left(\mathrm{P(n)} * \log 2\left(\frac{1}{\mathrm{P(n)}}\right)\right) = 1.3_{\text{(aproximadamente)}}$$

Com execução da aplicação os valores obtidos foram iguais aos acima referidos.

Assim sendo podemos concluir que a aplicação está a funcionar corretamente.

Exercicio 2:

Tendo em conta o proposto para este exercício e o exercício 1, realizado anteriormente, foi criado apenas mais um método **makeout** que recebendo como parâmetro um mapa em que as key correspondem aos símbolos que irão ser inseridos no ficheiro e o values a sua probabilidade de ocorrência, este método irá criar um novo ficheiro e irá correr um algoritmo que tento em conta as probabilidades de cada símbolo e um número gerado de maneira random irá selecionar o símbolo a ser inserido.

Como a utilização da aplicação do exercício 1 foi possível verificar que a entropia do ficheiro original e do novo ficheiro criado era muito semelhante assim sendo podemos concluir que a aplicação criada para este exercício está a funcionar corretamente.

Entropia original = 4.4734206199646

Entropia copia = 4.475713729858398

Exercicio 3:

Este exercício recebe como argumento um ficheiro, e faz um mapeamento de caracteres, contando o numero de ocorrências que um caracter aparece frente a outro. Quando acaba de fazer o mapeamento, conta os números de ocorrência de cada primeiro caracter, e converte o valor para uma percentagem. Depois de fazer o mapeamento, eu cria um ficheiro, onde o programa recebe o primeiro caracter, e depois através de um gerador de números aleatórios em 0 e 1, e adicionado o próximo caracter conforme as percentagens do mapa. O programa recebe um numero de caracteres que escreve no ficheiro. A solução depois e testada com os programas do exercício anterior.

Exercicio 4:

Neste exercício pretende- se calcular a entropia de fonte, implementar um gerador com as mesmas características e obter o comprimento médio da árvore de Huffman.

Os símbolos gerados por esta fonte vão estar compreendidos entre 3 e 18 pois se calhar 3 vezes 1 no dado lançado ou 3 vezes 6. Deste modo a fonte só vai ter 15 símbolos possíveis.

A análise do gerador é semelhante ao dos exercícios anteriores mas com uma particularidade para além dos 15 símbolos possíveis.

Anteriormente os símbolos tinham no máximo 1 byte mas neste caso podem ter até 2 bytes pois se o valor tiver 2 caracter, por exemplo, o 18.

Para resolver este problema é necessário que no dicionário seja possível representar este par de caracter na chave.

A árvore de Huffman está a ser gerada em função dos resultados estatísticos da fonte. Ordenando as probabilidades de forma crescente e gerar os nós parent partir das folhas.

Em termos de símbolos gerados confirma-se que os símbolos que aparecem com mais frequência na geração da fonte também têm mais frequência no simulador por nós implementado.

Exercicio 5:

Para esta aplicação foram criadas 2 classes, **Codificador** que analisa os símbolos e atribui-lhe um sequência de bits e a class **Descodificador** que tento em conta uma sequência de bits lida de um ficheiro e um e as sequencias de bits para cada símbolo descodifica a sequencia e conseguindo assim os símbolos dessa sequencia codificada.

A class **Codificador**:

Esta class contem 4 métodos, **codif**, **mapCode, cod** e **creatFile**.

-> codif, utilizando os métodos do **Exercicio 1, readFile, calculateFreq** e **calculateEntrop,** obtém um mapa que contem os símbolos do ficheiro mais a sua frequência bem como o valor da entropia. Com o mapa das frequências faz uma chamada para o método **mapCode** e **creatFile.**

-> mapCode, com o auxilio de um object **Domain**, o método irá analisar cada valor do mapa obtido, contendo os símbolos e sua respetiva probabilidade de ocorrência, e irá obter o um intervalo para cada símbolo, e a posição media desse intervalo, com estes valores e com o auxilio do método **code** irão ser preenchidos 2 mapas, um em que a key é o símbolo e outro em que a key é a sequencia de bits desse símbolo.

->cod, este é o método responsável por criar a sequencia binaria para cada simbolo.

Usando o cálculo para deter minar o numero de bits para a codificação de cada símbolo.

$$nBits = \log2(\frac{1}{\mathrm{P(n)}}) + 1$$

Arredondando sempre para cima, Obtém-se sempre o número mínimo de bits necessário para cada símbolo.

Após a obtenção do número mínimo de símbolos, uma transformação do valor medio do intervalo desse símbolo em binário.

$$2^{-1}\ 2^{-2}\ 2^{-3}\ \dots\ 2^{-nBits}$$

Retornando no fim a sequencia obtida.

->creatFile, este método irá ler o ficheiro e a cada simbolo lido irá substituilo por a sua respetiva sequencia binaria obtida no método **mapCode** e **cod.**

Observações:

Aplicando esta class a uma esquecia de símbolos “aaabbbbbbc” obteve-se:

Símbolo	Probabilidade
a	$\frac{3}{10}$
b	$\frac{6}{10}$
c	$\frac{1}{10}$

Tabela 1 Probabilidade Símbolo

Símbolo	Intervalo	Média	nBits	Sequencia
a	[0,0.3[	0.15	3	001
b	[0.3,0.9[	0.6	2	10
c	[0.9,1[	0.05	5	11110

Tabela 2Codificção por Símbolo

Tendo em consideração os valores obtidos podemos concluir concluímos que a codificação foi realizada com sucesso para a sequência de símbolos apresentada.

A class **Descodificador**:

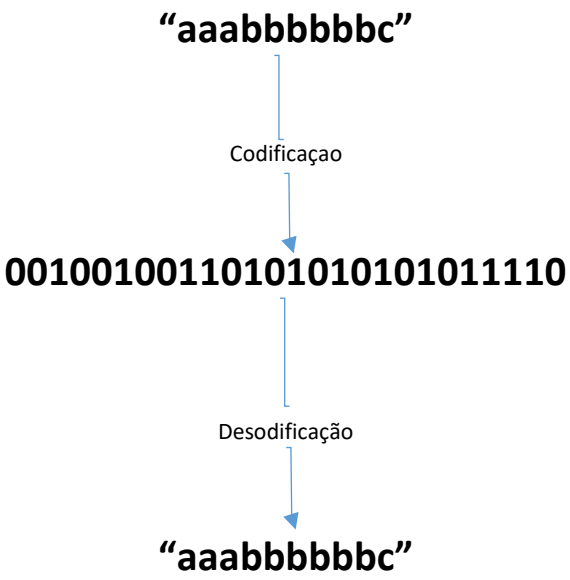
Esta class contém apenas 2 métodos, **descod** e **seeWhatBytes.**

-> descod, é o método que irá ler o ficheiro cujo o nome foi passado como parâmetro e por cada linha de símbolos codificados irá chamar o método **seeWhatBytes,** com o seu retorno irá substituir a linha lida pelos símbolos descodificados.

-> seeWhatBytes, como apresentado no enunciado este descodificador tem acesso ao dados que o codificador criou na codificação, assim sendo com o uso de um mapa criado e preenchido a quando a codificação, este método irá montar um sequencia bit a bit e sempre que adicionar um bit verificara se esse mapa contem essa sequencia como chave, caso não contenha essa chave então irá adicionar o próximo bit a sequencia, caso contrario, irá buscar ao value o símbolo descodificado e ira adiciona-lo a sequencia de símbolos de retorno.

Observaçoes:

Para a sequência:



Tendo em conta os resultados obtidos pode-se concluir que a descodificação correu corretamente.

Exercício 6:

Os métodos mais usados são os de huffman e o código aritmético, mas ambos têm as suas vantagens e desvantagens. O sistema numérico assimétrico vem tentar resolver os contras dos sistemas mais usados. O código de huffman consegue processar com grande alfabeto de símbolos, mas tem as suas relativas falhas. O código aritmético, para contrariar o de huffman, e bastante preciso, mas as suas operações aritméticas são bastante lentas. O sistema numeral assimétrico trabalha com bits de dimensão N, onde N é logaritmo de 2. Conseguindo assim trabalhar com grande alfabeto, mas usando uma tabela pequena. Mas o facto de ser assíncrono, consegue encriptar a mensagem simultaneamente com a chave. Assim consegue trabalhar com grandes dicionários com relativamente pouco tempo.

Exercício 7:

Neste exercício pretende-se fazer compressão de ficheiros utilizando o algoritmo Deflate contido na biblioteca jzlib. Ao analisar a compressão obtida pelos diferentes niveis de compressão podemos concluir que:

- A taxa de compressão é maior quanto maior for o tamanho do ficheiro que pretendemos comprimir.
- A compressão sem compressão aumenta o tamanho final do ficheiro após compressão.
- A compressão best compression tem taxas de compressão muito semelhantes à default compression.
- A compressão default e a best compression são as compressões que comprimem em maior quantidade.
- A best speed compression apesar de ser mais rápida não comprime tanto como as restantes.
- Para casos de equiprobabilidade dos simbolos da fonte o desempenho da compressão é muito mau tendo mesmo o tamanho do output ser superior ao de input.

8. Neste exercício pretendemos usar o algoritmo de deflate para comparar o desempenho da compressão para grupos de ficheiros, sendo a compressão efetuada com a concatenação de todos os símbolos ou comprimir todos os ficheiros em separado.

Podemos concluir que a compressão em coletivo tem melhor desempenho que a compressão em separado.