

**Instituto Superior de Engenharia de
Lisboa**

**ENGENHARIA INFORMÁTICA E DE
COMPUTADORES**



***SISTEMA DE MULTIMÉDIA
PRIMEIRO TRABALHO PRÁTICO***

LI41N-LI61N

Autores – Grupo 4:

40602, Sara Sobral

40682, Renato Júnior

40686, Eduardo António

Índice

Desenvolvimento de conclusões	3
Exercício 1.....	3
Recorrendo à função file_entropy.m	3
Recorrendo à função coder_evaluation.m	6
Exercício 2.....	7
Recorrendo à função coder_decoder_evaluation.m	7
Exercício 3.....	8
Exercício 4.....	9
REGRAS DO SISTEMA DE INFORMAÇÃO	10
Exercício 5.....	10

Desenvolvimento de conclusões

Exercício 1

Recorrendo à função file_entropy.m

- Obtenha o histograma e analise o seu formato (concentrado ou disperso). Relacione o formato do histograma com o valor de $H(X)$.

Para correr a função file_entropy.m foi realizado um script (ex1ab.m) que chama a função para todos os ficheiros presentes em testFilesSM.zip.

O valor máximo da entropia será sempre 8bit/simb pois por omissão tem 256 bits.

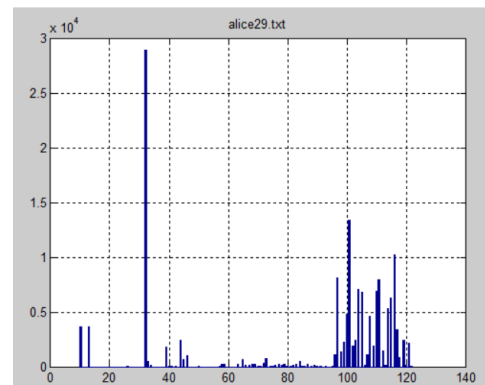
- alice29.txt

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Analisando o histograma nota-se uma grande dispersão na ocorrência dos símbolos, sendo o ficheiro analisado de texto, o símbolo ocorrido mais é o “e”.

Existindo uma grande dispersão o valor da entropia estará afastado do seu limite superior.

$$H(x) = 4.5677 \text{ bit/simb}$$



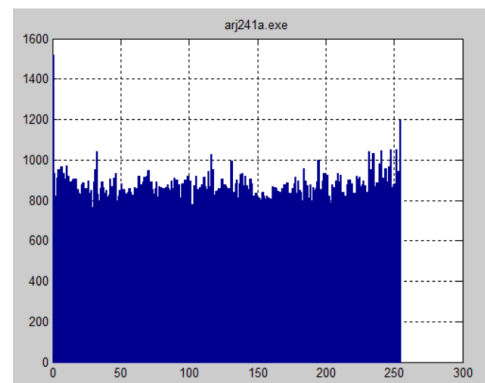
- arj241a.exe

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Analisando o histograma nota-se uma grande concentração na ocorrência dos símbolos.

Existindo uma grande concentração no histograma o valor da entropia estará próximo do seu limite superior.

$$H(x) = 7.9958 \text{ bit/simb}$$



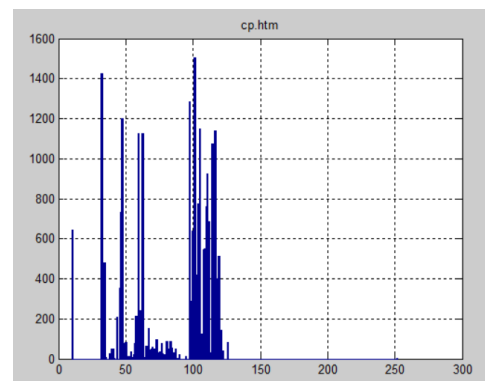
- cp.htm

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Só existem ocorrências de símbolos no intervalo [0,125[. Analisando esse intervalo nota-se uma grande dispersão na ocorrência dos símbolos.

Portanto o valor da entropia estará perto da mediana do seu intervalo.

$$H(x) = 5.2291 \text{ bit/simb}$$



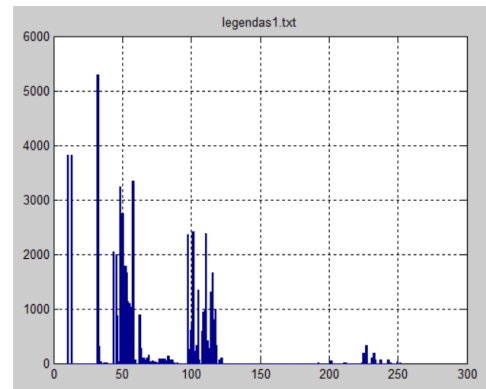
- legendas1.txt

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Analisando o histograma nota-se uma grande dispersão, havendo uma maior ocorrência do símbolo “e”, pois este é um ficheiro de texto.

Portanto o valor da entropia estará perto da mediana do seu intervalo.

$$H(x) = 5.1565 \text{ bit/simb}$$



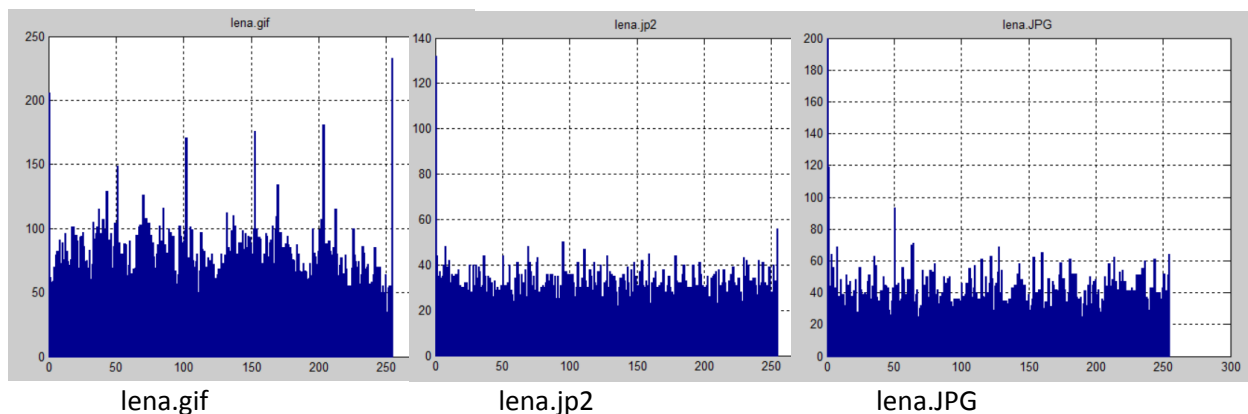
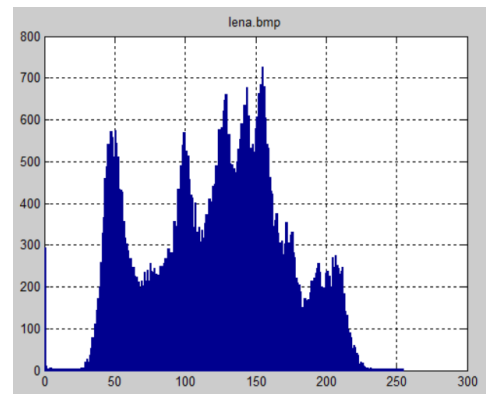
- lena.bmp

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Sendo o ficheiro uma imagem haverá maior concentração na ocorrência dos símbolos, pois cada pixel está relacionado com o seu vizinho.

Existindo uma grande concentração no histograma o valor da entropia estará próximo do seu limite superior.

$$H(x) = 7.4588 \text{ bit/simb}$$



$$H(x) = 7.9500 \text{ bit/simb}$$

$$H(x) = 7.9642 \text{ bit/simb}$$

$$H(x) = 7.9324 \text{ bit/simb}$$

Sendo o ficheiro uma imagem haverá maior concentração na ocorrência dos símbolos, pois cada pixel está relacionado com o seu vizinho. O valor da entropia está muito próximo do seu limite superior.

Neste formato o resultado do histograma é semelhante, havendo pouca diferença na ocorrência de cada símbolo, sendo que no formato .gif a ocorrência é mais “desregulada”, notando-se mais a diferença entre os símbolos mais e menos ocorridos.

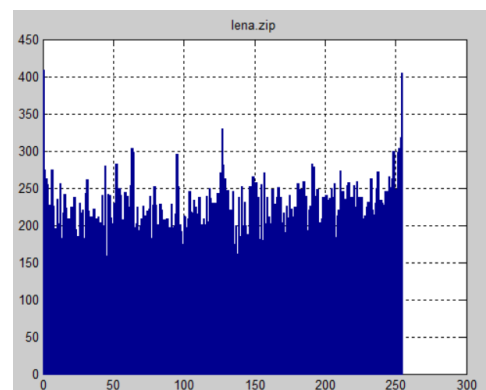
- lena.zip

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Sendo o ficheiro um zip de uma imagem haverá maior concentração na ocorrência dos símbolos, mas a ocorrência de cada símbolo que foi codificado será maior que se fosse a imagem original.

Existindo uma grande concentração no histograma o valor da entropia estará próximo do seu limite superior.

$$H(x) = 7.9832 \text{ bit/simb}$$

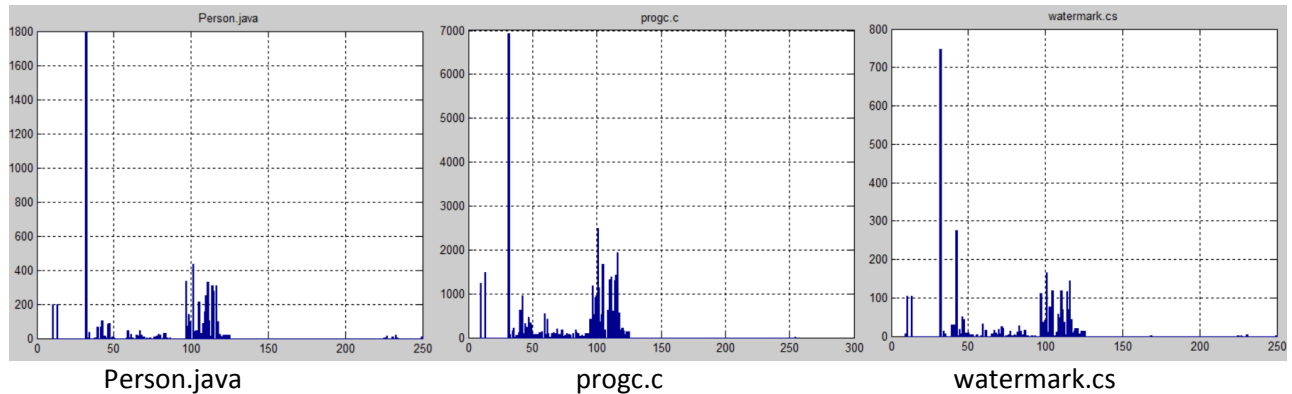
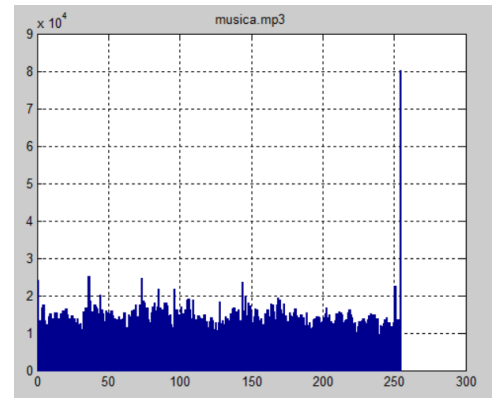


- musica.mp3

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

$$H(x) = 7.9551 \text{ bit/simb}$$

A frequência de cada símbolo é sempre semelhante portanto, existindo uma grande concentração no histograma o valor da entropia estará próximo do seu limite superior.



$$H(x) = 4.4595 \text{ bit/simb}$$

$$H(x) = 5.2011 \text{ bit/simb}$$

$$H(x) = 4.7225 \text{ bit/simb}$$

Analisando o histograma nota-se uma grande dispersão na ocorrência dos símbolos, sendo o ficheiro analisado de texto, o símbolo ocorrido mais é o “e”.

Existindo uma grande dispersão o valor da entropia estará afastado do seu limite superior, portanto em ambos os formatos o valor da entropia será semelhante.

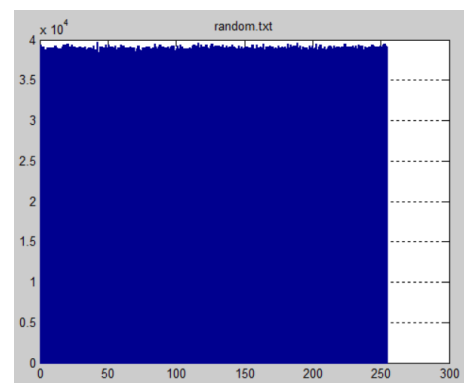
- radom.txt

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

Por ser um ficheiro que gera aleatoriamente os seus símbolos, a probabilidade de ocorrência de cada símbolo será igual. Portanto o histograma apresentado terá para todos os símbolos a mesma ocorrência.

O valor da entropia para casos como este está no seu limite máximo.

$$H(x) = 8.0000 \text{ bit/simb}$$



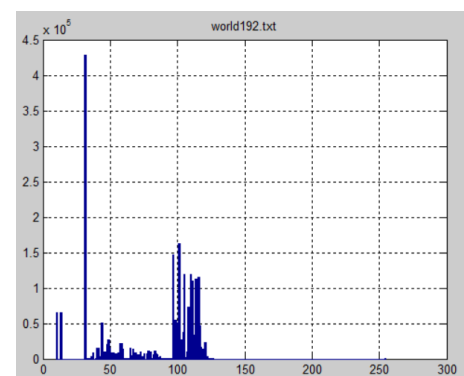
- world192.txt

$$0 \leq H(x) \leq \log_2(256) (=) 0 \leq H(x) \leq 8$$

$$H(x) = 4.9983 \text{ bit/simb}$$

Analisando o histograma nota-se uma grande dispersão na ocorrência dos símbolos.

Existindo uma grande dispersão o valor da entropia estará afastado do seu limite superior.



Recorrendo à função coder_evaluation.m

- Avaliar as taxas de compressão de cada codificador em cada ficheiro. Relacionar a taxa de compressão, com os valores da entropia do ficheiro.

Para correr a função coder_evaluation.m foi realizado um script (ex1cd.m) que antes de chamar a função pergunta ao utilizador se quer chamar a função para apenas um ficheiro ou todos os ficheiros de testFilesSM.zip, a interação com o utilizador é feita através da janela de comandos.

- TC: Taxa de compressão
- PR: Percentagem de remoção
- BPB: Bit por byte

Ficheiros	Adaptive Huffman			Semi Adaptive Huffman			Arithemtic Semi Adaptive Huffman		
	TC	PR	BPB	TC	PR	BPB	TC	PR	BPB
alice29.txt	0.5767	42.3338	4.6133	0.5794	42.0629	4.6350	0.5733	42.6711	4.5863
arj241a.exe	1.0014	-0.1398	8.0112	1.0016	-0.1590	8.0127	1.0007	-0.0683	8.0055
cp.htm	0.6630	33.7032	5.3037	0.6628	33.7235	5.3021	0.6581	34.1910	5.2647
legendas1.txt	0.6493	35.0701	5.1944	0.6508	34.9211	5.2063	0.6474	35.2595	5.1792
lena.bmp	0.9369	6.3110	7.495	0.9391	6.0933	7.5125	0.9365	6.3530	7.4918
lena.gif	1.0114	-1.1360	8.0909	1.0089	-0.8909	8.0713	1.0062	-0.6175	8.0494
lena.jp2	1.0321	-3.2087	8.2567	1.0288	-2.8784	8.2303	1.0267	-2.6660	8.2133
lena.JPG	1.0226	-2.2552	8.1804	1.0189	-1.8885	8.1511	1.0157	-1.5677	8.1254
lena.zip	1.0055	-0.5510	8.0441	1.0043	-0.4311	8.0345	1.0025	-0.2503	8.0200
musica.mp3	0.9926	0.7426	7.9406	0.9969	0.3093	7.9753	0.9945	0.5516	7.9559
Person.java	0.5740	42.6036	4.5917	0.5784	42.1600	4.6272	0.5761	42.3895	4.6088
progc.c	0.6574	34.2621	5.2590	0.6579	34.2117	5.2631	0.6537	34.6256	5.2300
watermark.cs	0.6238	37.6161	4.9907	0.6318	36.8151	5.0548	0.6293	37.0715	5.0343
random.txt	1.0005	-0.0519	8.0042	1.0005	-0.0507	8.0041	1.0000	-0.0041	8.0003
world192.txt	0.6278	37.2225	5.0222	0.6335	36.6505	5.0680	0.6273	37.2672	5.0186

A taxa de compressão é sempre semelhante nos vários codificadores Huffman, por isso irem comparar a taxa de compressão do codificador Huffman adaptativo com a entropia.

Ficheiros	H(x)	Valor máximo H(x)	TC
alice29.txt	$H(x) = 4.5677 \text{ bit/simb}$	8 bit/simb	0.5767
arj241a.exe	$H(x) = 7.9958 \text{ bit/simb}$		1.0014
cp.htm	$H(x) = 5.2291 \text{ bit/simb}$		0.6630
legendas1.txt	$H(x) = 5.1565 \text{ bit/simb}$		0.6493
lena.bmp	$H(x) = 7.4588 \text{ bit/simb}$		0.9369
lena.gif	$H(x) = 7.9500 \text{ bit/simb}$		1.0114
lena.jp2	$H(x) = 7.9642 \text{ bit/simb}$		1.0321
lena.JPG	$H(x) = 7.9324 \text{ bit/simb}$		1.0226
lena.zip	$H(x) = 7.9832 \text{ bit/simb}$		1.0055
musica.mp3	$H(x) = 7.9551 \text{ bit/simb}$		0.9926
Person.java	$H(x) = 4.4595 \text{ bit/simb}$		0.5740
progc.c	$H(x) = 5.2011 \text{ bit/simb}$		0.6574
watermark.cs	$H(x) = 4.7225 \text{ bit/simb}$		0.6238
random.txt	$H(x) = 8.0000 \text{ bit/simb}$		1.0005
world192.txt	$H(x) = 4.9983 \text{ bit/simb}$		0.6278



Pelos valores apresentados podemos assumir que a taxa de compressão está relacionada com a entropia de um ficheiro, pois quanto mais perto estiver a entropia do seu limite máximo maior será a taxa de compressão.

Exercício 2

Recorrendo à função `coder_decoder_evaluation.m`

- Escolher 12 ficheiros de teste (com 4 ficheiros de cada conjunto Calgary Corpus, Canterbury Corpus e Silesia Corpus). Compare e comente os resultados obtidos, para os três pares codificador/descodificador

FICHEIRO	Adaptive Huffman			Semi Adaptive Huffman			Arithmetic Semi Adaptive Huffman		
	TC	PR	BPB	TC	PR	BPB	TC	PR	BPB
alice29.txt	1.7341	-73.4117	13.8729	1.7260	-72.6009	13.8081	1.7443	-74.4320	13.9546
arj241a.exe	0.9986	0.1396	7.9888	0.9984	0.1588	7.9873	0.9993	0.0683	7.9945
cp.htm	1.5084	-50.8369	12.0669	1.5088	-50.8831	12.0706	1.5195	-51.9548	12.1564
lena.bmp	1.0674	-6.7361	8.5389	1.0649	-6.4887	8.5191	1.0678	-6.7840	8.5427
lena.gif	0.9888	1.1232	7.9101	0.9912	0.8830	7.9294	0.9939	0.6137	7.9509
lena.jp2	0.9689	3.1089	7.7513	0.9720	2.7978	7.7762	0.9740	2.5968	7.7923
lena.JPG	0.9779	2.2055	7.8236	0.9815	1.8535	7.8517	0.9846	1.5435	7.8765
musica.mp3	1.0075	-0.7482	8.0599	1.0031	-0.3103	8.0248	1.0055	-0.5546	8.0444
Person.java	1.7423	-74.2271	13.9382	1.7289	-72.8908	13.8313	1.7358	-73.5794	13.8864
progc.c	1.5212	-52.1193	12.1695	1.5200	-52.0026	12.1602	1.5297	-52.9650	12.2372
watermark.cs	1.6030	-60.2979	12.8238	1.5827	-58.2657	12.6613	1.5891	-58.9104	12.7128
world192.txt	1.5929	-59.2928	12.7434	1.5785	-57.8544	12.6284	1.5941	-59.4062	12.7525

Tabela de tempos

FICHEIRO	Adaptive Huffman		Semi Adaptive Huffman		Arithmetic Semi Adaptive Huffman	
	Cod.	Desc.	Cod.	Desc.	Cod.	Desc.
alice29.txt	0.2960	0.2720	0.2950	0.2970	0.3180	0.4390
arj241a.exe	1.1680	1.4130	1.2120	1.2930	1.3710	1.6270
cp.htm	0.2060	0.2200	0.2320	0.2390	0.2570	0.2570
lena.bmp	0.2290	0.2290	0.2470	0.2500	0.3150	0.3610
lena.gif	0.8150	0.8340	0.9510	0.8940	0.9100	1.1300
lena.jp2	0.1990	0.2150	0.2270	0.2480	0.2580	0.2820
lena.JPG	0.9880	0.9130	0.8480	0.9500	0.9320	0.9610
musica.mp3	1.7340	1.5010	1.6180	3.8320	4.7930	1.9060
Person.java	0.1970	0.2010	0.2130	0.2520	0.2490	0.2740
progc.c	0.8010	0.8910	0.8370	0.9110	0.9710	1.0260
watermark.cs	0.1970	0.2050	0.2100	0.2370	0.2430	0.2770
world192.txt	0.8120	0.8140	0.7670	0.7850	0.8730	1.0110

A TC, PR, BPB são sempre semelhantes nos vários codificadores Huffman, o mesmo acontece com os descodificadores.













Onde podemos realmente observar diferenças é no tempo de execução de cada um. Por norma o tempo de codificação do Huffman aritmético é sempre pior, exceto no caso do ficheiro lena.JPG em que consegue ser melhor que o Huffman adaptativo.

Na descodificação o Huffman adaptativo é mais rápido, logo a seguir o semi adaptativo e de seguida o aritmético. No entanto existem dois casos em que o Huffman aritmético consegue superar o semi adaptativo (arj241a.exe e musica.mp3).

- Sobre os 12 ficheiros escolhidos na alínea anterior aplicar um codificador. Apresentar a taxa de compressão obtida para cada ficheiro. Compare as taxas de compressão com as obtidas anteriormente

O codificador escolhido foi o WinRar.

- do: dimensão original
-dc: dimensão codificada
-dd: dimensão decodificada
- TC: Taxa de compressão

						Comparação da Taxa de Compressão dos vários Huffman		
		do	dc	dd	TC	Adaptativo	Semi Adaptativo	Aritmético
 alice29.rar	51 KB							
 arj241a.rar	219 KB	149	51	149	2,921569	<	<	<
 cp.rar	8 KB	219	219	219	1	=	=	=
 lenabmp.rar	45 KB	25	8	25	3,125	<	<	<
 lenagif.rar	21 KB	66	45	66	1,466667	<	<	<
 lenajp2.rar	9 KB	21	21	21	1	=	=	=
 lenaJPG.rar	11 KB	9	9	9	1	=	=	=
 musica.rar	3 677 KB	11	11	11	1	=	=	=
 Person.rar	3 KB	3738	3677	3738	1,01659	~	~	~
 prog.c.rar	13 KB	7	3	7	2,333333	<	<	<
 watermark.rar	2 KB	39	13	39	3	<	<	<
 world192.rar	520 KB	4	2	4	2	~	~	~
		2416	520	2416	4,646154	<	<	<

Exercício 3

- **Descrição da metodologia**

Este exercício tem como objetivo implementar uma fonte de símbolos com alfabeto e função massa de probabilidade (FMP) genéricos.

Inicialmente perguntamos ao utilizador quantas sequências ele pretende obter, de cada fonte. Ou seja, se o utilizador quiser cinco sequências, o nosso programa irá gerar cinco sequências decimais e cinco sequências alfanuméricas

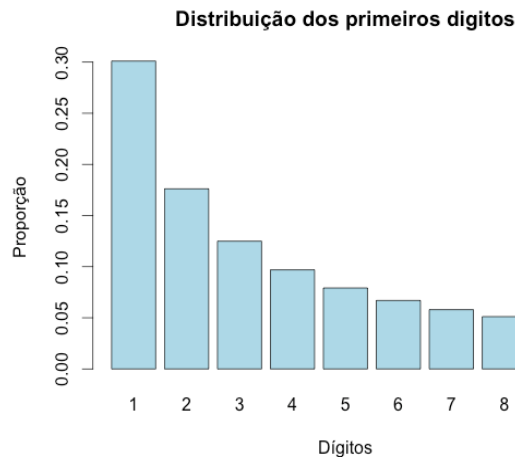
As sequências que são produzidas pela fonte, tem dimensão L=8 ou L=24, e são escritas num ficheiro de texto (uma sequência por linha).

Para tal, temos duas funções distintas:

- decimalSequence;
- alfaNumericSequence.

As sequencias decimais, com L=8, respeitam a Lei de Benford, em que para o primeiro dígito d pertence {1,...,9} com $p(d)=\log_{10}(1+1/d)$ e para os restantes dígitos, assume-se distribuição uniforme.

No nosso código geramos uma probabilidade aleatória e verificamos o seu respetivo número, segundo o seguinte gráfico:



Primeiro dígito	1	2	3	4	5	6	7	8	9	Total
Porcentagem (%)	30,00	17,67	13,00	9,00	8,33	6,33	5,67	5,67	4,33	100,00

Para os restantes números da sequência é gerado um valor aleatório de 0 a 9.

Quanto às sequências alfanuméricas, com $L=24$, a nossa função consiste em gerar valor aleatório, entre 1 e 36, referentes ao índice do nosso vetor. Vetor este que vai de A-Z e de 0-9. De seguida só temos que ir buscar o valor que está no índice (índice calculado aleatoriamente). `idx = sequence[i]; AZ09[idx]`.

Exercício 4

➤ Descrição da metodologia

Para a realização deste exercício foi utilizada a linguagem java.

Etapas:

1. Criar dois objetos que suportam os dados de um individuo e de uma aposta.
2. Ler os ficheiros .txt (Nomes.txt, Apelidos.txt, Concelhos.txt e Profissões.txt), guardando, para cada ficheiro, cada uma das suas linhas numa lista simplesmente ligada respetiva ao ficheiro.
3. Consultar cada uma das listas geradas para obter os atributos de um individuo.
 - 3.1. Gerar aleatoriamente um inteiro, 1 ou 2, para saber se o individuo terá 1 ou 2 nomes próprios. Repetir o processo para os apelidos.
 - 3.2. Escolher aleatoriamente da lista que contém os nomes, o número de calculado na alínea anterior, de nomes. Realizar o mesmo para a lista que contém os apelidos. Escolher aleatoriamente das listas concelhos e profissões um concelho e uma profissão.
 - 3.3. Criar um objeto individuo com os parâmetros obtidos. Este objeto tem a capacidade de gerar um número de cidadão com 8 dígitos único.
4. Repetir o ponto 3 pelo menos 1000, guardando os indivíduos numa lista própria para tal. O número de vezes que o ponto 3 é realizado é gerado aleatoriamente, no mínimo será feito um 1000 vezes).
5. Para criar uma aposta, obtém-se uma posição aleatória da lista de indivíduos e enquanto o individuo obtido já tiver realizado 4 apostas continua-se a ir buscar a aleatoriamente um individuo. A partir desse individuo obtém-se o número do cidadão apostador. O número da aposta é gerado de forma aleatória em que os primeiros 5 números inteiros estão no intervalo [1,50] e os restantes 2 estão no intervalo [1,11]. A data da aposta respeita os anos bissextos no formato dd-mm-aaaa.
 - 5.1. O número máximo de aposta geradas pode ser o mesmo número de indivíduos existentes.
6. Para gerar tabelas lêem-se as listas construindo uma *string* com o formato de tabela preenchida.

7. Para gerar o ficheiro que contém uma tabela cria-se um “Writer” que está encarregue que escrever linha a linha a *string* criada no ponto 6.
8. Os ficheiros gerados apresentam os seguintes nomes: TabelaIndividuos.txt e TabelaApostas.txt.

REGRAS DO SISTEMA DE INFORMAÇÃO

Para que não existam indivíduos com números de cidadão iguais ou apostas com números iguais é usado uma lista *HashSet* que não permitem objetos iguais.

Exercício 5

➤ Apresentação de resultados

Ficheiros testados:	Taxa de compressão:
fraseSimples.txt	1.3522727272727273
alice29.txt	0.6577776828041476
cp.htm	0.7205117262122506
legendas1.txt	0.65027848839092
Person.java	0.609511243689766
progc.c	0.7871204664193231
watermark.cs	0.5998478051906441
world192.txt	0.7797181776354996

➤ Comentários

O algoritmo LZ78 se baseia na construção de um dicionário com os dados encontrados anteriormente no arquivo a ser comprimido. No início o dicionário se encontra vazio. A medida que o arquivo vai sendo lido, caractere por caractere, cada sequência de caracteres não encontrada no dicionário é introduzida no dicionário e ganha um código (posição, símbolo).

Quando o dicionário fica totalmente preenchido optou-se por mante-lo, sem se fazer quaisquer alterações a este.

Os resultados obtidos podem ser visualizados em:

.\LZ78Tokenize\(\nome do ficheiro testado)-LZ78.txt

A taxa de compressão com o algoritmo LZ78 é geralmente 0,5 – 0,7, excepto no ficheiro fraseSimples.txt que contém uma sequência de 22 símbolos e a taxa de compressão é superior a 1.

Os resultados obtidos levam a concluir que este método não se poderá aplicar a ficheiros de pequenas dimensões, pois poderão ser utilizar mais bits para codificar (do token), do que os bits que o ficheiro original contem.

Nota: Existe um ficheiro chamado ficheiros.txt que contém o nome dos ficheiros a ser testados, os quais são apresentados na tabela acima.