

**Trabalho Prático de Comunicação Digital**

**Módulo 1**

|  |
| --- |
| David Costa 45935  Miguel Almeida 47249  Tiago Silva 46005 |
|  |

|  |  |
| --- | --- |
| Orientador | Artur Ferreira |
|  |  |

Maio de 2022

**Exercício 1**

**Aula prática exercício 4**

1. **Função int count\_ones( int val ), a qual retorna o número de bits a 1 no valor inteiro val, passado como parâmetro**

Esta função usa uma máscara com o valor 1 para verificar o valor lógico and entre o valor e a máscara bit a bit. O valor e afetado por um Shift à direita para cada um dos 32 bits de um número inteiro. Cade vez que esta operação lógica retorna true incrementamos o contador que é retornado no final da execução.

**Exemplo:**

Input val = 10 (1010 binário) => Output = 2

Input val = 32 (100000 binário) => Output = 1

Input val = 1152 (1111 binário) => Output = 4

1. **Função void print\_bits( int array[], size\_t array\_size ), a qual imprime como caracteres os valores dos bits de todos os elementos de array, com array\_size inteiros.**

Para cada elemento do array utilizamos uma máscara com o valor 0x80 e verificamos o valor logico da operação and da mascara com o valor. Esta operação é feita para cada bit do valor lido do array e para cada bit lido se o resultado da operação for true imprime 1 ou false imprime 0.

**Exemplo:**

Val = 10 =>bin = 00000000000000000000000000001010

Val = 1024 => bin = 00000000000000000000010000000000

Val = 32 => bin = 00000000000000000000000000100000

Val = 1152 => bin = 00000000000000000000010010000000

Val = 1 => bin = 00000000000000000000000000000001

Val = 138 => bin = 00000000000000000000000010001010

1. **Função int count\_symbol( char \*file\_name, char symbol ), a qual retorna o número de vezes que o símbolo symbol, ocorre no ficheiro file\_name, passado como parâmetro.**

A função abre o ficheiro em modo de leitura e verifica cada caracter até End of File, por cada caracter lido se este corresponder ao caracter recebido como parâmetro é incrementado uma variável de contador que é retornada no fim da leitura.

**Exemplo:**

Ficheiro exemplo.txt => 1234abcd

[]{}ABCDabcd

Input = ‘1’ => Symbol Count = 1

Input = ‘b’ => Symbol Count = 2

Input = ‘D’ => Symbol Count = 1

**Aula prática exercício 5**

1. **Função que apresenta os primeiros N termos da sequência de Fibonnaci.**

Os primeiros dois membros da sequência fibonnaci são sempre 0 e 1, o seguintes são calculados através da soma dois dois anteriores até N iterações.

**Exemplo:**

Input => 7 Output => 0, 1, 1, 2, 3, 5, 8, 13

1. **Função que calcula e apresenta os primeiros N termos da progressão aritmética, com primeiro termo u e razão r. Os valores de N, u e r são passados como parâmetro.**

Esta função escreve o primeiro termo u e os seguintes são calculados através do anterior multiplicado pela razão r durante N iterações.

**Exemplo:**

Input => N = 5, u = 1, r = 2

Output => 1, 2, 4, 8, 16

1. **Função que identifica o símbolo mais frequente num ficheiro passado como parâmetro, indicando a frequência de ocorrência desse símbolo.**

Nesta função é usado um array de tamanho 128 inicializado a zero onde cada posição corresponde ao número de ocorrências de um caracter, em que o índice do array será o seu código ASCII.

O ficheiro é aberto e lido caracter a caracter, sendo incrementado o índice do array correspondente ao caracter lido.

No fim da leitura percorremos o array de ocorrências para encontrar o mais ocorrido e escrevemos no output o seu valor correspondente ao código do se índice.

**Exemplo:**

Ficheiro test => ccc

adadadadadadada

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

Output = b

**Exercício 2**

1. Implementamos a função string\_generator, que recebe como parâmetros o alfabeto de strings (dictionary), a função massa de probabilidade (probability), a dimensão das strings geradas (repeat) e uma flag (hist) para mostrar o histograma.

É retornado o valor da entropia e um ficheiro de output onde é escrita a sequência de strings geradas.

É utilizada a função random.choices que gera uma lista de strings aleatórias de acordo com os valores fornecidos e de acordo com a probabilidade de cada caracter ocorrer (guardada na variável result).

Contamos o número de ocorrências das strings geradas e calculamos a entropia utilizando a sua probabilidade em função das ocorrências.

Para gerar o histograma utilizamos a biblioteca matplotlib para mostrar as strings geradas em função das ocorrências.

1. **Geração de sequências de símbolos**
2. **Geração automática de palavras-passe com número variável de caracteres (entre o mínimo e o máximo estabelecidos). A palavra-passe deverá ser alfanumérica, contendo obrigatoriamente caracteres minúsculos, maiúsculos, símbolos de pontuação e algarismos.**

Foi implementada a função pass\_gen que recebe como parâmetros a dimensão mínima (min\_size) e máxima (max\_size).

Utilizamos o método random.randint para gerar um numero aleatório para o tamanho da palavra-passe entre os valores min\_size e max\_size.

Geramos uma lista dos caracteres possíveis para serem utilizados na palavra-passe (caracteres, dígitos e pontuação).

Definimos uma probabilidade igual para cada símbolo gerado e utilizamos a função desenvolvida na alínea a) (string\_generator) para gerar a palavra-passe.

De seguida usamos a função check\_password para verificar se a passe gerada contem pelo menos uma minúscula, uma maiúscula, um dígito e um caracter de pontuação.

1. **Geração automática de sequências alfanuméricas, ou seja, algarismos decimais e letras maiúsculas, com L = 24, semelhantes a chaves de ativação e registo de software (por exemplo: RTY9 GHUI 1JER 82TY SGJP IUDS).**

Implementamos a função key\_gen sem parâmetros.

O funcionamento é idêntico ao pass\_gen mas a lista de valores usados apenas contem letras maiúsculas e dígitos e a função check\_key confirma se chave gerada respeita os critérios.

Para medir a robustez das chaves geradas recorremos à entropia calculada por cada chave através da função strength.

**c)** **Utilize convenientemente a implementação, para a geração automática de conteúdos de tabelas a utilizar num sistema de informação de um jogo de sorte. Pretende-se preencher duas tabelas com dados, gerados aleatoriamente, sobre indivíduos (apostadores) e as respetivas apostas.**

Para este problema criamos variáveis associadas ao stream de cada ficheiro de texto fornecido em modo de leitura (nomes -> fileNames, apelidos -> fileSurnames, concelhos -> fileLocals, profissões -> fileProfessions) em que cada foi posteriormente afetada por uma lista com todas as linhas do ficheiro respectivo e todos os caracteres de espaço removidos.

Criamos um ficheiro para os indivíduos (“Individuos”) e outro para as apostas (“Apostas”).

Os números dos cartãos de cidadão foram gerados através do método random.sample que gera valores 1499 valores únicos entre 10000000 e 99999999 (8 algarismos decimais) e armazenados na variável idList.

Usamos a função string\_generator da alínea a) do exercício 2, para gerar os valores possíveis de nomes, apelidos, concelhos, profissões, o método random.sample para gerar a os valores possíveis das apostas e o método random.randint para gerar uma data e escolher uma aposta aleatória das geradas anteriormente.

**Exercício 3**

1. **Implemente o par codificador/descodificador de código unário, a funcionar em modo semi-adaptativo.**
2. **Apresente resultados experimentais que comprovem o correto funcionamento do par codificador/descodificador, recorrendo a ficheiros do conjunto CD\_TestFiles.zip.**
3. **Para ficheiros produzidos pela fonte do exercício 2, alínea c), apresente a taxa de compressão obtida. Apresente resultados experimentais tais que comprovem o cumprimento do primeiro Teorema de Shannon**

**Exercício 4**

**(Python) A função LZ77\_Tokenizer descreve o ficheiro de entrada através de tokens LZ77, obtidos com search window (SW) e look-ahead-buffer (LAB) de dimensões indicadas pelo utilizador.**

**Gera um ficheiro de texto com os tokens obtidos no processamento do ficheiro de entrada, sendo que cada linha contém um token e apresenta os histogramas e a entropia dos campos position e length.**

1. Os ficheiros gerados com os valores dos tokens foram guardados na diretoria ‘Test Files’ e têm o nome ‘TOKENS\_(nome do ficheiro fornecido)’.

Na função LZ77\_Tokenizer definimos duas variáveis para o índice do início da janela de pesquisa (dictIterator) e para o índice do fim/início da janela de codificação (labIterator).

Com estes índices criamos a janela de pesquisa (dictionary) e a janela de codificação (lab) com os valores do texto de entrada compreendidos entre os índices respectivos.

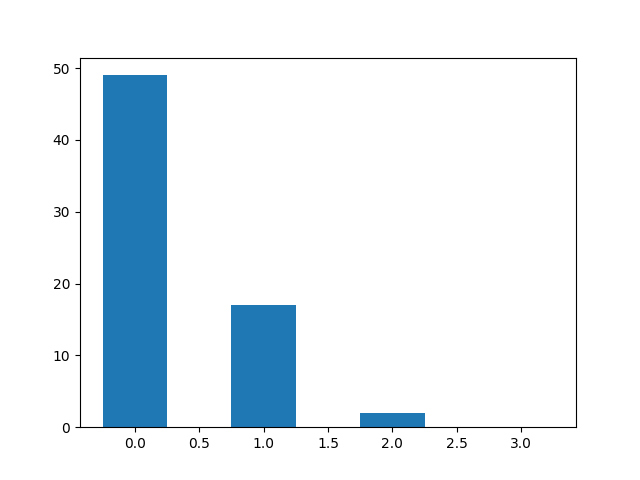
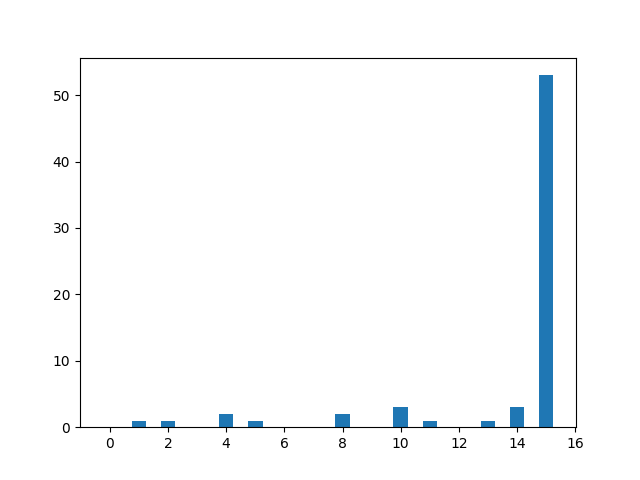
Passamos à chamada da função LZ77\_search recebe como parâmetros a janela de pesquisa (dictionary) e a janela de codificação (lab), retornando todos os valores necessários à criação de um token.

Atualizamos as ocorrências dos campos position (histPosition) e length (histLength) para posterior apresentação dos histogramas, e atualizamos os valores dos índices dictIterator e labIterator para obtermos o próximo token.

No fim da leitura dos ficheiros apresentamos os histogramas e a entropia dos campos position e length.

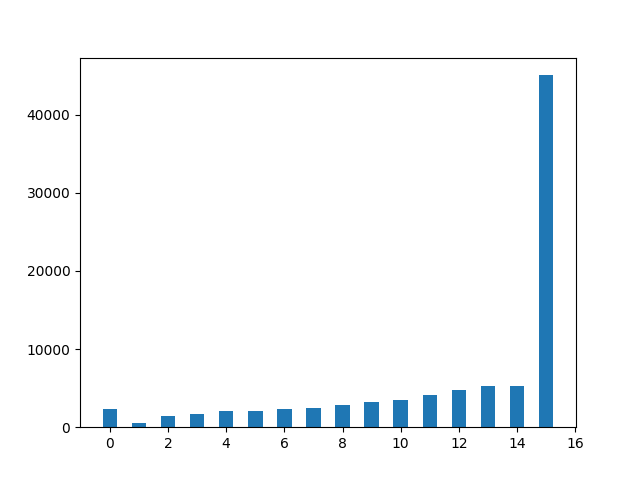
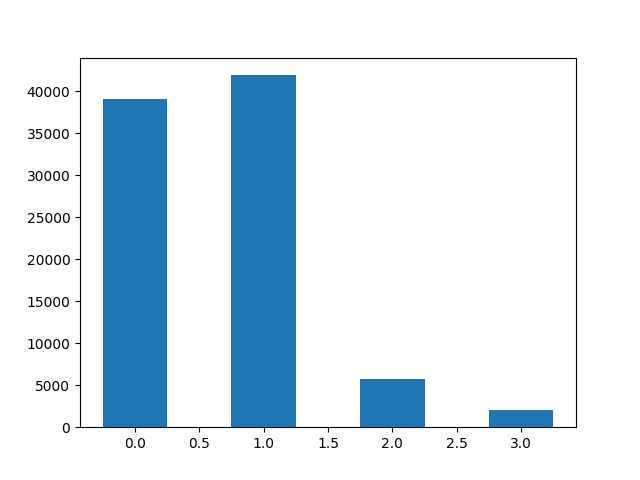
1. **Apresente resultados experimentais obtidos com ficheiros do conjunto CD\_TestFiles.zip e gerados pela fonte do exercício 2, alínea c).**

**a.txt**

****

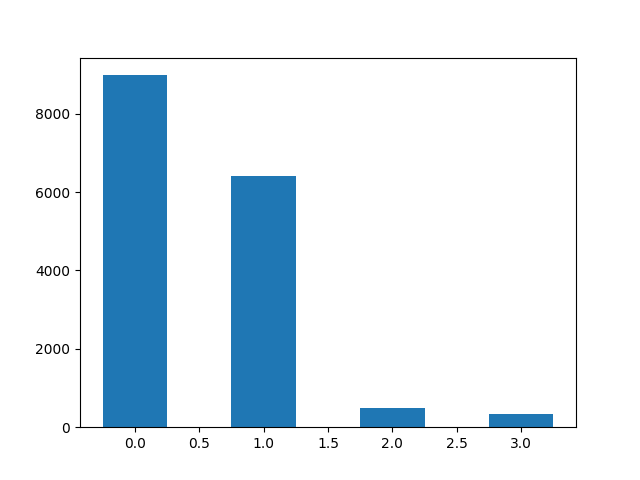
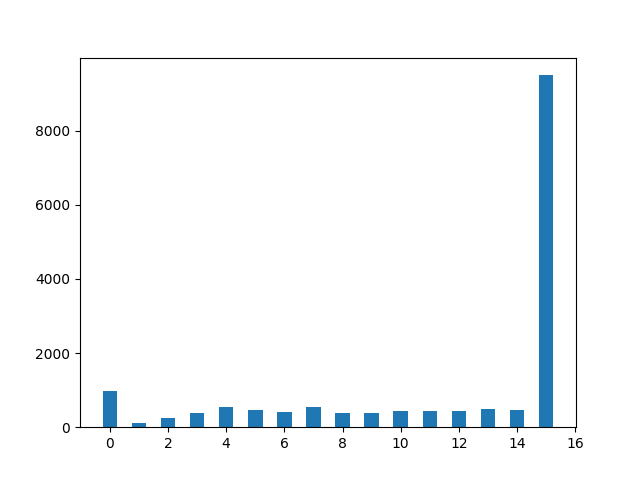
Histogramas position (entropia = 1.4244) e length (entropia = 0.9903)

**alice29.txt**

****

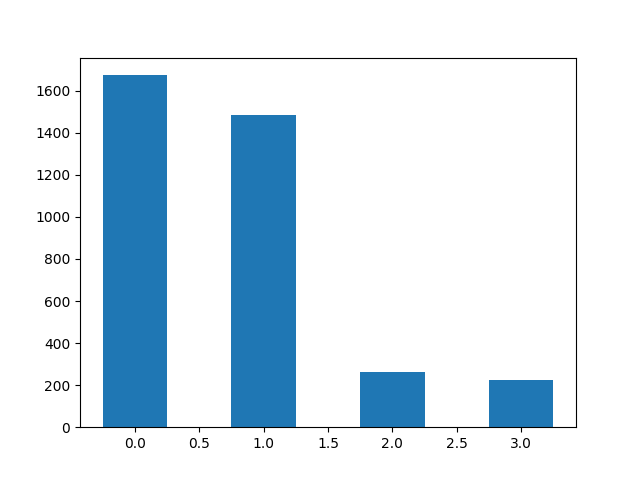
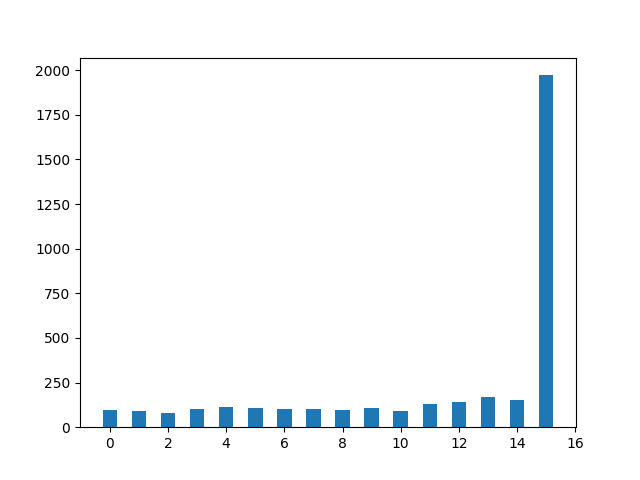
Histogramas position (entropia = 2.8449) e length (entropia = 1.4142)

**cp.htm**

****

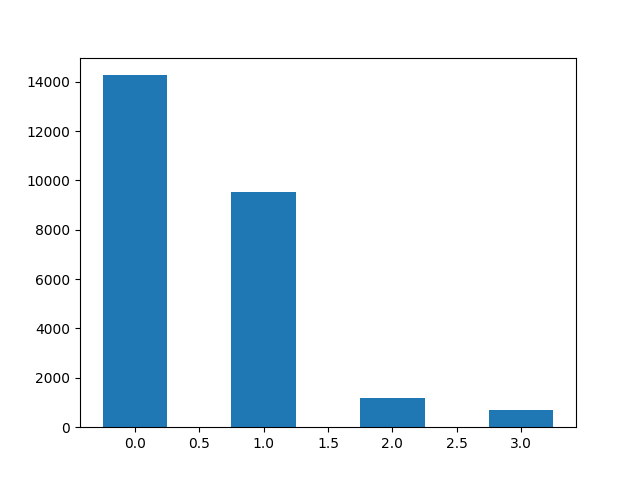
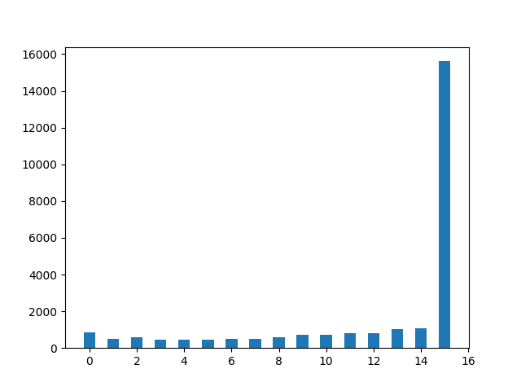
Histogramas position (entropia = 2.5521) e length (entropia = 1.2700)

**Person.java**

****

Histogramas position (entropia = 2.7762) e length (entropia = 1.5662)

**prog.c**

****

Histogramas position (entropia = 2.4716) e length (entropia = 1.3462)

**lena.bmp**

Não conseguimos codificar este ficheiro.

**2.c)**

Histograma position (entropia = ) e length (entropia = )