ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

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

a) **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 é afetado por um shift à direita para cada um dos 32 bits de um número inteiro. Cada 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

b) **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 máscara 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

c) **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

a) Função que apresenta os primeiros N termos da sequência de Fibonnaci.

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

Exemplo:

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

b) 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

c) 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 seu índice.

Exemplo:

Ficheiro test => ccc
adadadadadadada
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

Output = b

Exercício 2

a) 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.

b) Geração de sequências de símbolos

i) 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 número aleatório para o tamanho da palavra-passe entre *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.

ii) 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 enunciados.

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 respetivo e todos os caracteres de espaço removidos.

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

Os números dos cartões 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

a) Implemente o par codificador/descodificador de código unário, a funcionar em modo semi-adaptativo.

Para a codificação, a função *encoder* começa por contar o número de símbolos e de ocorrências de cada símbolo por ficheiro.
Os símbolos são ordenados pela ordem crescente de ocorrências e é calculada a função massa de probabilidade.
Na primeira linha do ficheiro de output escrevemos o modelo que vai ser posteriormente ser usado pelo descodificador, e na segunda linha escrevemos o valor codificado de acordo com o índice do seu símbolo.

Na descodificação, a função *decoder* lê o modelo do ficheiro codificado e lê de seguida traduz os valores codificado de acordo com o número de bits a '1' até ao bit de separação '0'.

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.

a) 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 respetivos.

Passamos à chamada da função *LZ77_search* que 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.

b) Apresente resultados experimentais obtidos com ficheiros do conjunto CD_TestFiles.zip e gerados pela fonte do exercício 2, alínea c).

a.txt

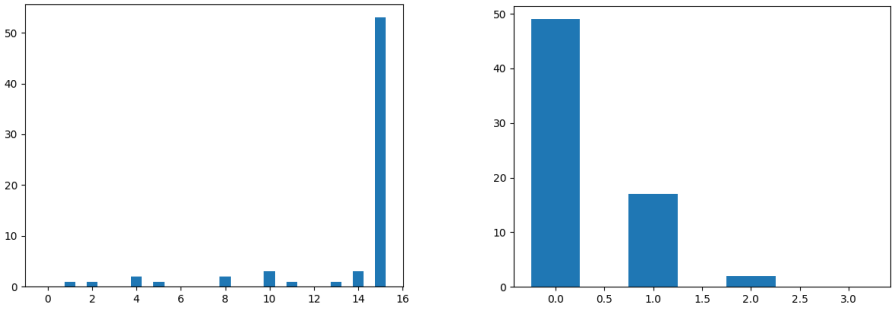

Figura 1 - Histogramas position (entropia = 1.4244) e length (entropia = 0.9903)

alice29.txt

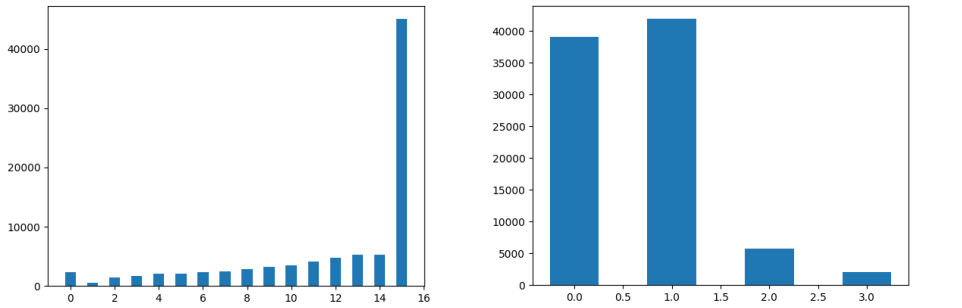

Figura 2 - Histogramas position (entropia = 2.8449) e length (entropia = 1.4142)

cp.htm

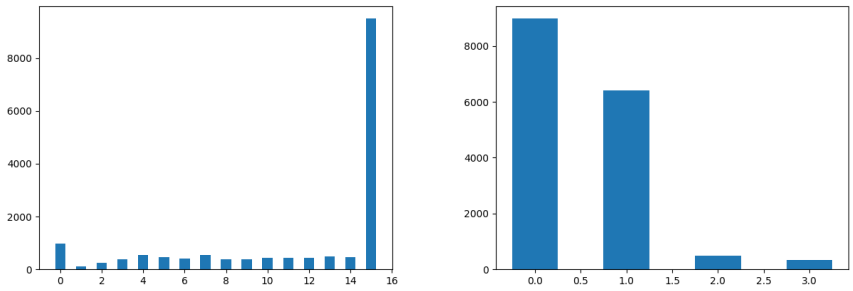

Figura 3 - Histogramas position (entropia = 2.5521) e length (entropia = 1.2700)

Person.java

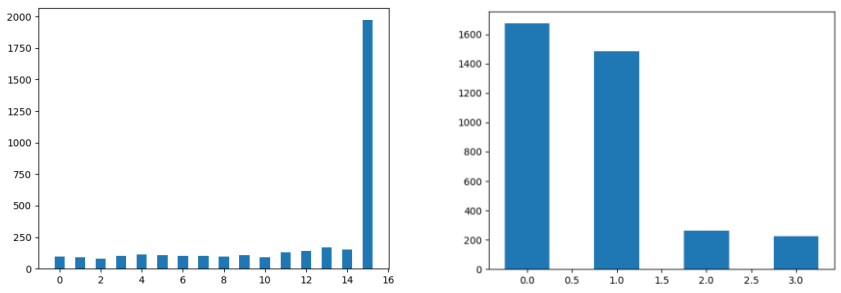

Figura 4 - Histogramas position (entropia = 2.7762) e length (entropia = 1.5662)

prog.c

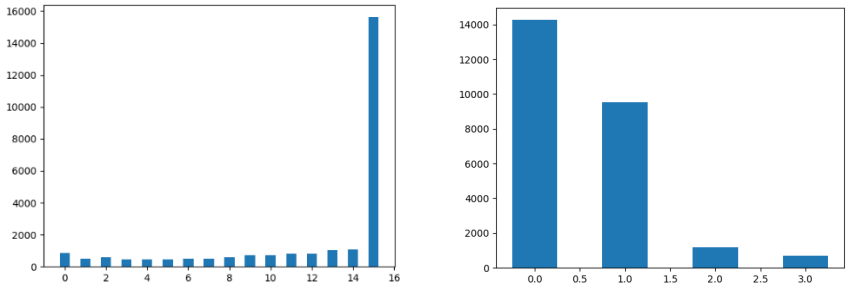

Figura 5 - Histogramas position (entropia = 2.4716) e length (entropia = 1.3462)

lena.bmp

Não conseguimos codificar este ficheiro.

2.c)

Indivíduos

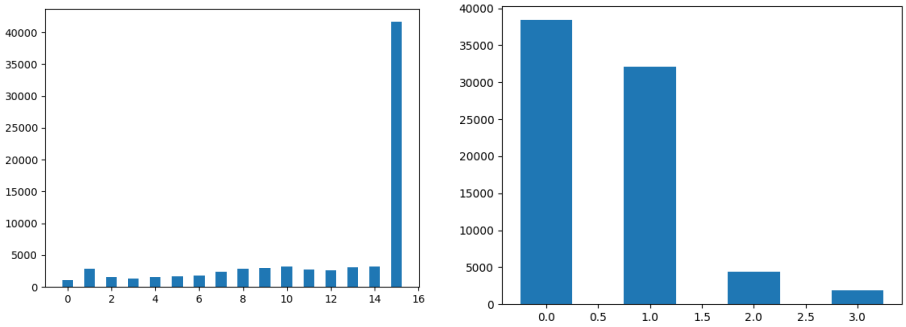

Figura 6 - Histograma position (entropia = 2.7498) e length (entropia = 1.3960)

Apostas

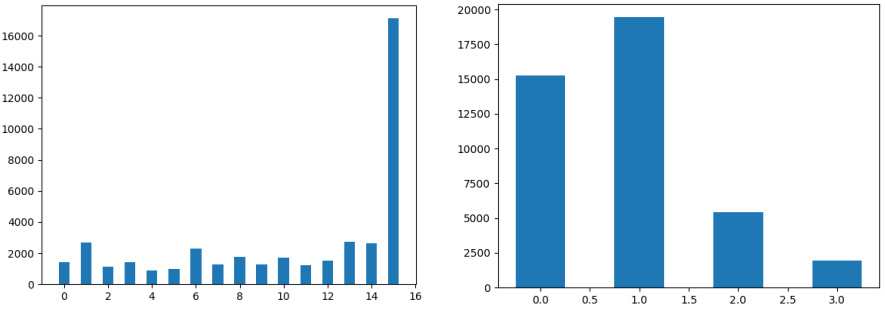

Figura 7 - Histograma position (entropia = 3.2399) e length (entropia = 1.6315)