



**Departamento de Engenharia de Electrónica e
Telecomunicações e de Computadores**

Trabalho Prático – Módulo 1

Autores:	48089	André Filipe Pina Páscoa
	48280	André Filipe do Pilar de Jesus
	48287	Nyckollas Brandão

LEIC41D Grupo 15

Relatório para a Unidade Curricular de Comunicação Digital da
Licenciatura em Engenharia Informática e de Computadores

Professor: Engenheiro Artur Ferreira

17 – 04 – 2022

<< Esta página foi intencionalmente deixada em branco >>

Resumo

O presente trabalho incide sobre a primeira parte do programa da unidade curricular de Comunicação Digital: sistemas de comunicação digital (SCD), teoria da informação e codificação de fonte.

Foram realizados quatro exercícios distintos, com recurso às linguagens de programação “C” e “Python”. O primeiro exercício consistiu na realização de pequenos programas básicos com o objetivo de realizar um primeiro contacto com a linguagem “Python”, e uma revisão da linguagem “C”. No segundo exercício foi desenvolvida uma fonte de strings e alguns programas para testar o funcionamento da mesma. O terceiro exercício teve como objetivo o desenvolvimento de um par codificador/descodificador de código unário. O quarto e último exercício consistiu no desenvolvimento do algoritmo de codificação LZ77 (Lempel-Ziv 1977).

Após a realização do trabalho, concluiu-se que os objetivos de aprendizagem foram alcançados, produzindo os resultados pretendidos e adquirindo conhecimentos da matéria em estudo.

Palavras-chave: sistema de comunicação digital, teoria da informação, codificação de fonte, código unário, codificação baseada em dicionário

Abstract

This project focuses on the first phase of the Digital Communications curricular unit (SCD), Information theory and source coding.

Four exercises were implemented, with the help of C and Python programming languages. The first exercise consisted in developing programs with the objective of learning the basics of Python and revising the C programming language. In the second exercise we developed a strings source and a few tests for it. The third exercise a encoding/decoding pair of comma code was developed. The fourth exercise consisted in the development of a LZ77 (Lempel-Ziv 1977) encoding algorithm.

After the project was completed, all the objectives were achieved, producing the intended results while acquiring the knowledge that was pretended.

Keywords: digital communication systems, information theory, font encoding, comma code, dictionary coder

Índice

1	Introdução	1
2	Resolução dos Exercícios	2
2.1	Exercício 1	3
2.2	Exercício 2	4
2.3	Exercício 3	7
2.4	Exercício 4	11
3	Conclusão	13
4	Software utilizado	14
5	Referências	15

Lista de Figuras

Figura 1 – Diagrama do funcionamento da função <i>strings_source</i>	4
Figura 2 – Exemplo de histograma gerado pela função <i>strings_source</i>	5
Figura 3 – Exemplo de histograma do campo “length” dos tokens, gerado pela função <i>LZ77_Tokenizer</i> com o ficheiro a.txt	12
Figura 4 - Exemplo de histograma do campo “position” dos tokens, gerado pela função <i>LZ77_Tokenizer</i> com o ficheiro a.txt	12

Listagens

Listagem 1 – Geração da sequência de strings na função <i>strings_source</i>	4
Listagem 2 – Resultado do teste com o ficheiro <i>alice29.txt</i>	8
Listagem 3 – Resultados dos testes com os ficheiros <i>bets.txt</i> e <i>citizens.txt</i>	9
Listagem 4 – Número de ocorrências por símbolo do ficheiro <i>bets.txt</i>	10
Listagem 5 – Geração dos tokens na função <i>LZ77_Tokenizer</i> , com chamadas consecutivas da função <i>LZ77_get_token</i>	11

1 Introdução

Este trabalho teve como principal objetivo o estudo e aplicação de conceitos fundamentais sobre SCD (Sistemas de Comunicação Digital), teoria de informação e codificação de fonte.

Inicialmente foi realizada uma revisão destes conceitos, com recurso às folhas de apoio e aos slides disponibilizados pelo professor.

É de ressaltar que todos os resultados produzidos, incluindo comentários no código fonte, estão em língua inglesa, à exceção do presente documento escrito em português.

Após a revisão dos conceitos em estudo, iniciou-se o desenvolvimento do exercício 1. Os programas do primeiro exercício já tinham sido desenvolvidos na primeira aula prática da unidade curricular, por isso apenas se adicionaram os comentários e os testes dos programas às soluções.

Com o primeiro exercício resolvido, passou-se à implementação do exercício 2, que teve como principal objetivo o desenvolvimento de uma fonte de strings. Nesse exercício também foram aplicados conceitos da teoria da informação, como o cálculo da entropia ou a construção de histogramas.

O terceiro exercício incidiu sobre o código unário (*comma code*). Teve como principal objetivo o desenvolvimento de um par codificador/descodificador de código unário, a funcionar em modo semi-adaptativo. Neste exercício foram aplicados conhecimentos sobre codificação de fonte.

O quarto e último exercício, teve como objetivo o estudo de técnicas de codificação de fonte baseadas em dicionário, especificamente o algoritmo de codificação LZ77 (Lempel-Ziv 1977). Com esse objetivo, a vertente de codificação desse algoritmo foi implementado com recurso à linguagem “Python”.

No seguinte capítulo estão apresentados os processos de resolução dos quatro exercícios.

2 Resolução dos Exercícios

Este capítulo contém a explicação das decisões tomadas na resolução dos exercícios.

Todos os exercícios foram devidamente comentados e testados.

Para poder executar os programas em Python, é necessário ter instalado uma versão da linguagem igual ou superior à 3.9, e as bibliotecas utilizadas no seu desenvolvimento como, por exemplo, matplotlib. Para executar os programas em C é necessário ter um ambiente com um compilador da linguagem.

2.1 Exercício 1

O exercício 1 consistiu na realização de pequenos programas básicos nas linguagens “C” e “Python”. Estes programas foram implementados na primeira aula prática da disciplina e tiveram como objetivo realizar um primeiro contacto com a linguagem “Python”, e uma revisão da linguagem “C”.

O exercício é constituído por seis programas, três de cada linguagem.

A diretoria do exercício 1 está dividida em duas sub-diretórias:

- *c* – contém os ficheiros dos exercícios em na linguagem “C”, um por ficheiro;
- *python* - contém os ficheiros dos exercícios em na linguagem “Python”, um por ficheiro;

Funções na linguagem “C”:

- *count_ones* – retorna o número de bits com o valor “1” num inteiro;
- *print_bits* – imprime como caracteres os valores dos bits dos elementos de um array;
- *count_symbol* – retorna o número de vezes que um determinado símbolo ocorre num ficheiro;

Para testar o funcionamento das funções em “C”, foi desenvolvida uma função *main* para cada uma. Cada função *main* tem chamadas de cada função com parâmetros diferentes, que permitem comprovar o correto funcionamento da mesma.

Funções na linguagem “Python”:

- *fibonacci* – apresenta os primeiros N termos da sequência de Fibonacci;
- *arithmetic_sequence* – apresenta os primeiros N termos de uma progressão aritmética;
- *most_frequent_symbol* – apresenta o símbolo mais frequente num determinado ficheiro, indicando a sua frequência.

Para testar o funcionamento das funções em “Python”, foi utilizado a framewrok “PyTest”, que permite o desenvolvimento de testes unitários. Para cada função, foram implementados alguns testes unitários com chamadas com parâmetros diferentes, que permitem comprovar o correto funcionamento de cada função.

2.2 Exercício 2

O exercício 2 teve como principal objetivo o desenvolvimento de uma fonte de strings. Neste exercício são aplicados conceitos da teoria da informação, como o cálculo da entropia e a construção de histogramas. Neste exercício foi utilizada a linguagem “Python”.

A diretoria do exercício 2 está dividida em quatro sub-diretorias:

- strings_source – resolução da alínea (a);
- generate_password – resolução da alínea (bi);
- generate_sequence – resolução da alínea (bii);
- generate_table_content – resolução da alínea (c).

Começou-se por desenvolver a função *strings_source*, que recebe como parâmetro um alfabeto de strings e uma função massa de probabilidade (fmp) associada a esse alfabeto e retorna uma sequência de L strings. Na Figura 1 está ilustrado num diagrama o funcionamento desta função.

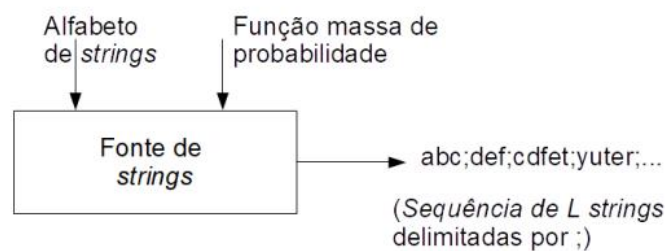


Figura 1 – Diagrama do funcionamento da função *strings_source*

O código apresentado na Listagem 1 é a porção da função *strings_source*, que tem como função a geração da sequência de strings retornada. É utilizada a função *choices* da biblioteca *random*, passando o alfabeto, a função massa de probabilidade e o número de strings a retornar. Na segunda linha é utilizada a função *join* da biblioteca *standard* do Python, para juntar a sequência numa string, separada pelo carácter “;”.

```
# Get string sequence
strs = random.choices(alphabet, fmp, k=L)
res = ";".join(strs)
```

Listagem 1 – Geração da sequência de strings na função *strings_source*

Para além da geração da sequência, esta função também apresenta a entropia da fmp e o histograma da sequência gerada, para isso, recorreram-se às bibliotecas do Python *scipy.stats* e *matplotlib*, respetivamente.

Em baixo, na Figura 2, apresenta-se um exemplo de um histograma gerado na execução da função *strings_source*, com um alfabeto de sete palavras.

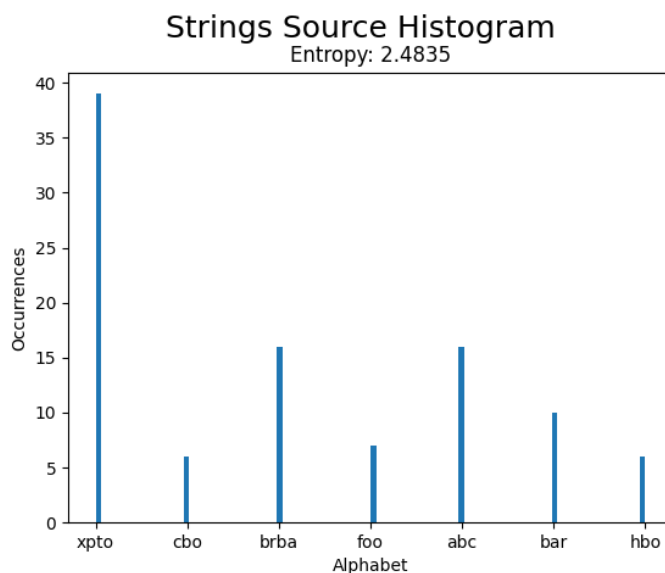


Figura 2 – Exemplo de histograma gerado pela função *strings_source*

Com o objetivo de testar a implementação da fonte de strings para gerar as sequências de símbolos, foram realizados dois programas: *generate_password*, que consiste na geração automática de palavras-passe; e *generate_sequence*, que realiza a geração automática de sequências alfanuméricas.

A função *generate_password*, gera uma palavra-passe, recebendo a dimensão mínima e máxima desta, e chamando a função *strings_source*. Este programa tem quatro alfabetos e respetivas fmps: uma para maiúsculas, minúsculas, algarismos e símbolos. Uma palavra-passe é gerada chamando *strings_source* com cada um destes alfabetos, e juntando os valores de retorno numa variável *password*. No final, a palavra-passe é “embaralhada” com recurso à função do Python *shuffle*.

Para medir a robustez das palavra-passe geradas, foi desenvolvida a função *check_password_strength*, que recebe uma palavra-passe e imprime o nível da sua robustez.

A função *generate_sequence* é muito mais simples, tendo como objetivo a geração de e sequências alfanuméricas, semelhantes a chaves de ativação e registo de software (por exemplo: RTY9 GHUI 1JER 82TY SGJP IUDS). Para gerar essa sequência, foi criado um alfabeto com letras maiúsculas e algarismos e a respetiva fmp, que são passados como parâmetro à função *strings_source*. Após isso é aplicada a função *join* do Python, que faz a separação da sequência em grupos de quatro caracteres, como apresentada no exemplo acima.

Para testar o funcionamento de ambas as funções, foram gerados cinco ficheiros com mil palavras-passe e cinco ficheiros com mil sequências alfanuméricas. Após a observação desses ficheiros concluiu-se que, tanto as palavras-passe como as sequências geradas, são robustas e aleatórias, tal como esperado.

Por fim, implementou-se o programa *generate_table_content*, que realiza a geração automática de conteúdos de tabelas a utilizar num sistema de informação. Esta função gera duas tabelas, preenchidas com dados gerados aleatoriamente: uma sobre indivíduos apostadores e outra com as respetivas apostas. Cada apostador tem a seguinte informação:

Número de Cidadão | Nome(s) Próprio(s) e Apelido(s) | Concelho de Residência | Profissão

Cada aposta tem a seguinte informação:

Número de Cidadão | Aposta | Data

A tabela dos indivíduos é gerada através da leitura de nomes, apelidos, concelhos de residência e profissões de ficheiros auxiliares disponibilizados pelo professor. Após ler cada ficheiro, é chamada a função *strings_source*, com cada um desses alfabetos e uma fmp uniforme. Após isso, é construído o array de indivíduos com as sequências retornadas pela função *strings_source*.

As apostas são compostas pelo número do cidadão apostador, valor da aposta e data da aposta. Para obter o valor da aposta, foi utilizada a função *random.sample* do Python, enquanto que para obter os números de cidadão e as datas, foram realizadas mais chamadas à função *strings_source*.

Para testar o funcionamento do programa, foi chamada a função com dois mil apostadores e respetivas apostas, gerando dois ficheiros, um com cada tabela.

2.3 Exercício 3

O exercício 3 incidiu sobre a codificação de código unário (*comma code*). Teve como principal objetivo o desenvolvimento de um par codificador/descodificador de código unário, a funcionar em modo semi-adaptativo. Neste exercício foram aplicados conhecimentos sobre codificação de fonte. Para a resolução deste exercício foi utilizada a linguagem “C”.

O exercício 3 está dividido em três sub-diretórias:

- include – contém os *header files* da resolução do exercício;
- src – contém os *source files* da resolução do exercício;
- test_files – contém os ficheiros utilizados nos testes.

Começou-se por desenvolver o codificador de código unário. O objetivo do codificador é escrever no ficheiro de output, a informação com o seguinte formato:

<Número de bytes do ficheiro original>\n

<Número de símbolos do ficheiro original>\n

<Símbolos ordenados pelas suas respetivas ocorrências no ficheiro original><Dados codificados bit-a-bit com código unário>

A primeira etapa do codificador é calcular o modelo baseado no ficheiro original, que é constituído pelo número de bytes do ficheiro original, número de símbolos e os símbolos ordenados pelas suas respetivas ocorrências. Este modelo, posteriormente, é utilizado na descodificação do ficheiro comprimido.

No modelo é necessário incluir o número de bytes do ficheiro original, porque os dados são codificados bit-a-bit, devido aos ficheiros serem baseados em bytes, se o número de bits não for múltiplo de 8 então irá existir ambiguidade no último byte.

Exemplo 1:

Bytes originais-> 0x88, 0x88, 0x66

Comprimido:

Modelo -> 0x88, 0x66

Dados comprimidos -> 0b0010

Como os dados codificados só possuem 4 bits seria impossível saber se o número de bytes originais era 3 ou 7 (0b0010000).

O número de símbolos do ficheiro original é incluído para saber quando é que o modelo acaba.

Para fazer a escrita dos dados codificados, itera-se sobre todos os bytes do ficheiro original e escreve-se o código unário associado a cada um.

Exemplo 2:

No caso do Exemplo 1, para o símbolo 0x88 o código unário é 0b0 e para 0x66 é 0b10, ou seja, o número de bits com o valor “1” é o respetivo índice no array de símbolos ordenados.

No caso do decodificador, é necessário ler o modelo do ficheiro codificado, que será usado para decodificar os dados codificados bit-a-bit. Para realizar essa decodificação é efetuada a contagem do número de bits a 1 consecutivos, e utilizada essa contagem como índice no array de símbolos ordenados do modelo.

Para provar que o par codificador/decodificador se encontra funcional, recorreu-se aos ficheiros do conjunto *CD_TestFiles.zip*. Para cada ficheiro foi calculado o tamanho previsto do ficheiro codificado, de seguida foi executado o codificador e, finalmente, feita a comparação dos dois números. Para garantir que o ficheiro original é igual ao decodificado, foram comparados o número de bytes de ambos.

No caso do ficheiro *alice29.txt*, o teste apresentou o resultado apresentado na Listagem 2.

```
File ../test_files/alice29.txt
File size: 152090 bytes
Decoded size: 152090 bytes
Expected encoded size: 182960 bytes
Encoded size: 182960 bytes
```

Listagem 2 – Resultado do teste com o ficheiro *alice29.txt*

Como é possível observar, o tamanho previsto do ficheiro codificado corresponde ao tamanho do ficheiro codificado, e o número de bytes do ficheiro original é igual ao do descodificado.

Também foram efetuados testes com os ficheiros *bets.txt* e *citizens.txt* do exercício 2c), neste caso foi calculada a taxa de compressão obtida e testado o comprimento do primeiro Teorema de Shannon. O resultado desses testes está apresentado na Listagem 3.

```
File ../test_files/bets.txt

File size: 84748 bytes
Decoded size: 84748 bytes
Expected encoded size: 54298 bytes
Encoded size: 54298 bytes
Compression ratio: 64.07%
Average length: 5.12 bits/symbol
Entropy: 3.52 bits/symbol
Encoding efficiency: 68.74%
-----
File ../test_files/citizens.txt

File size: 173125 bytes
Decoded size: 173125 bytes
Expected encoded size: 263423 bytes
Encoded size: 263423 bytes
Compression ratio: 152.16%
Average length: 12.17 bits/symbol
Entropy: 4.88 bits/symbol
Encoding efficiency: 40.14%
```

Listagem 3 – Resultados dos testes com os ficheiros *bets.txt* e *citizens.txt*

Ao analisar o resultado dos testes para ambos os ficheiros, é possível observar que o ficheiro *bets.txt* apresenta uma melhor taxa de compressão que o ficheiro *citizens.txt*, isto acontece porque o número de bits médio por símbolo do ficheiro *bets.txt* (5.12 bits/símbolo) é significativamente mais baixo que o do ficheiro *citizens.txt* (12.17 bits/símbolo). Este facto deve-se ao ficheiro *bets.txt* possuir maioritariamente números, que são constituídos por apenas nove símbolos, ao contrário do *citizens.txt* que possui noventa e nove símbolos distintos.

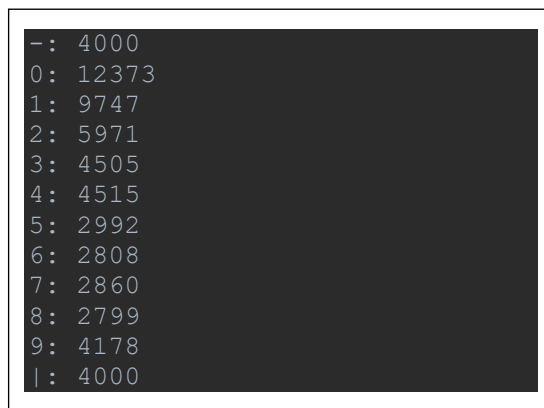
Para provar que o par codificador/descodificador segue o primeiro teorema de Shannon, foi calculada a entropia e o número de bits médio por símbolo dos ficheiros.

Segundo o primeiro teorema de Shannon, é possível codificar, sem distorção, um ficheiro de entropia H bits/símbolo, usando em média $L = H + e$ bits/símbolo.

Para provarmos este teorema calculámos a eficiência de codificação dada por:

$$\frac{H(X)}{L} = \frac{H(X)}{H(X)+e}$$

O número de ocorrências por símbolo do ficheiro *bets.txt* está apresentado na Listagem 4.



```
-: 4000
0: 12373
1: 9747
2: 5971
3: 4505
4: 4515
5: 2992
6: 2808
7: 2860
8: 2799
9: 4178
|: 4000
```

Listagem 4 – Número de ocorrências por símbolo do ficheiro *bets.txt*

No caso do ficheiro *bets.txt*, a eficiência de codificação é relativamente alta porque os dígitos ocorrem de forma quase uniforme (número de ocorrências dos dígitos é semelhante), ou seja, a entropia vai ser elevada. Como a entropia é elevada e o número médio de bits por símbolos é baixo, então este ficheiro terá uma boa eficiência de codificação utilizando a técnica de código unário.

No ficheiro *citizens.txt*, a entropia é baixa, porque a distribuição das ocorrências dos símbolos não é uniforme. Como a entropia é baixa e o número médio de bits por símbolo é elevado (devido a possuir um número elevado de símbolos), a eficiência de codificação não é boa.

Como é possível observar, o código unário segue o primeiro teorema de Shannon, mas só nos casos em que o número médio de símbolos é baixo (< 8) e a entropia elevada (distribuição uniforme).

2.4 Exercício 4

O exercício 4 teve como objetivo o estudo de técnicas de codificação de fonte baseadas em dicionário, especificamente o algoritmo de codificação LZ77 (Lempel - Ziv 1977). Foi pedido para implementar uma função *LZ77_Tokenizer*, que implementa esse algoritmo na vertente da codificação.

Para a resolução deste exercício foi escolhida a linguagem “Python”, por ser uma linguagem de mais alto nível, comparada com “C”, e que tem bibliotecas e funcionalidades que nos permitem desenvolver código mais simples e rapidamente.

Começou-se por implementar a função auxiliar *LZ77_get_token*, que recebe uma *search-window* e um *look-ahead-buffer*, e retorna um token com a melhor posição e comprimento encontrados. Esta função é a que tem a implementação do algoritmo em si, utilizando a técnica de *sliding-window*.

Na implementação do exercício, um token é representado por um tuplo de 3 propriedades: (*position*, *length*, *innovation_symbol*). Tanto a *search-window* como o *look-ahead-buffer* são representados por arrays de caracteres.

A função *LZ77_Tokenizer*, chama múltiplas vezes *LZ77_get_token*, até o ficheiro estar totalmente codificado. Na Listagem 5, abaixo, é possível observar a porção do código dessa função em que todos os tokens são obtidos.

```
# Get tokens
while i < len(data):
    sw = data[sw_i:i]
    lab = data[i:i + lab_length]

    token = LZ77_get_token(sw, lab)
    tokens.append(token)

    i += token[1] + 1
    sw_i = i - sw_length

    if sw_i < 0:
        sw_i = 0
```

Listagem 5 – Geração dos tokens na função *LZ77_Tokenizer*, com chamadas consecutivas da função *LZ77_get_token*

Para calcular a entropia das posições e dos comprimentos dos tokens, foi criada a função auxiliar *entropy_from_list*. Para apresentar o histograma desses mesmos campos, foi implementada a função auxiliar *show_histogram*.

Com o objetivo de testar a funcionalidade da função, esta foi chamada com diversos ficheiros de teste. Em baixo, na Figura 3 e na Figura 4, estão apresentados exemplos de histogramas, gerados pela função *LZ77_Tokenizer*, chamada com o ficheiro de teste “a.txt”. É possível observar na Figura 3, que a dimensão dos tokens mais frequente foi 0, significando que na geração desses tokens, não foram encontradas correspondências na *search-window*.

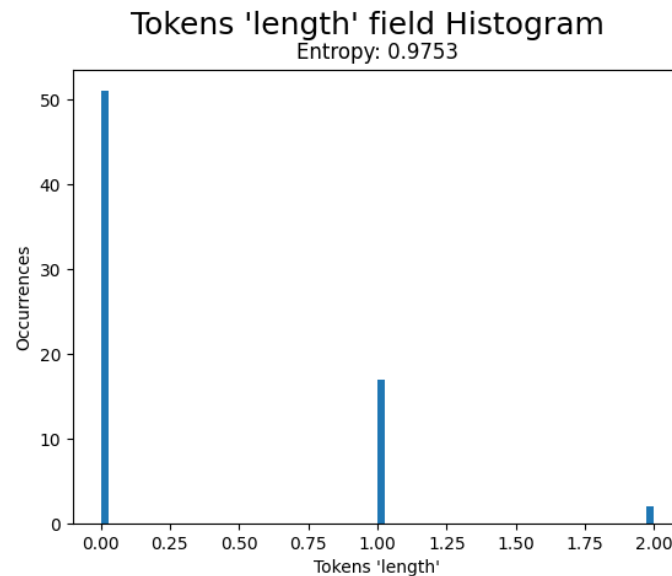


Figura 3 – Exemplo de histograma do campo “length” dos tokens, gerado pela função *LZ77_Tokenizer* com o ficheiro a.txt

É possível observar na Figura 4, que o campo “posição” dos tokens mais frequente foi 0, significando que na geração desses tokens, não foram encontradas correspondências na *search-window*. Também houve uma distribuição menor de outras ocorrências nos valores de posição 2, 4, 5 e 9.

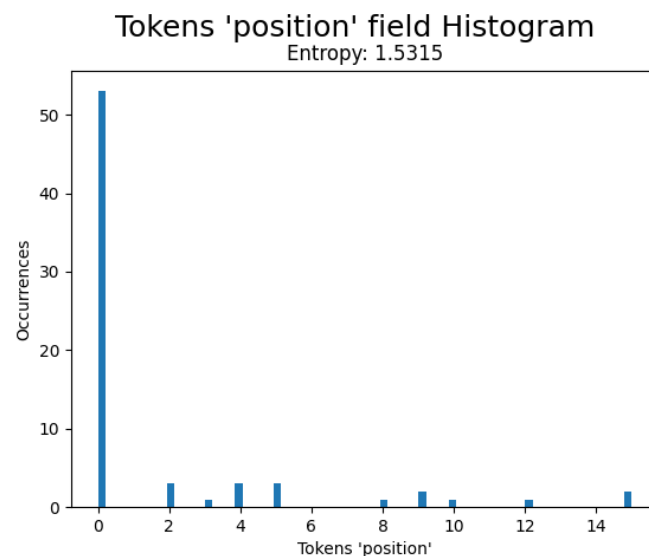


Figura 4 - Exemplo de histograma do campo “position” dos tokens, gerado pela função *LZ77_Tokenizer* com o ficheiro a.txt

3 Conclusão

Em suma, podemos concluir que os objetivos definidos para a realização deste trabalho foram atingidos, e consideramos que as resoluções dos exercícios apresentadas, são adequadas e cumprem com os requisitos do enunciado

Durante a implementação, adquirimos conhecimentos relativos aos sistemas de comunicação digital (SCD), a teoria da informação e às técnicas de codificação de fonte, através de código unário e de técnicas baseadas em dicionário. Este trabalho também nos possibilitou o desenvolvimento de competências na utilização das linguagens “C” e “Python” na construção de programas de baixa/médica complexidade.

Esta série de problemas proporcionou-nos a oportunidade de utilizar conhecimentos que viemos a adquirir nas aulas da unidade curricular, e durante o nosso estudo autónomo.

4 Software utilizado

Segue-se uma lista do software utilizado na realização deste trabalho, juntamente com uma breve descrição sobre a sua utilização:

- **Python 3.10**: linguagem utilizada no desenvolvimento dos exercícios 1, 2 e 4;
- **PyCharm 2021.3.3 (Professional Edition)**: ambiente de desenvolvimento de programas em Python;
- **C**: linguagem utilizada no desenvolvimento dos exercícios 1 e 3;
- **CLion 2022.1**: ambiente de desenvolvimento de programas em C;
- **Visual Studio Code 1.66.1**: ambiente de desenvolvimento de programas em C;
- **Git/GitHub**: controlo de versões e armazenamento do projeto num repositório;
- **Microsoft Word**: escrita do presente documento.

5 Referências

- [1] Slides 5.Teoria da Informação (21 de março de 2022) Retrieved April 3, 2022, from https://2122moodle.isel.pt/pluginfile.php/1155280/mod_resource/content/2/5.Teoria%20da%20Informacao.pdf
- [2] Slides 6.Codificação de Fonte - Técnicas Estatísticas ou Entrópicas (4 de abril de 2022). Retrieved April 3, 2022, from https://2122moodle.isel.pt/pluginfile.php/1156343/mod_resource/content/3/6.Codificacao%20de%20Fonte%20-%20Estat%C3%ADstica.pdf
- [3] Wikimedia Foundation. (2022, March 30). LZ77 and LZ78. Wikipedia. Retrieved April 12, 2022, from https://en.wikipedia.org/wiki/LZ77_and_LZ78
- [4] Budhrani, D. (2019, December 28). How data compression works: Exploring LZ77. Medium. Retrieved April 12, 2022, from <https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097>
- [5] Slides 7.Codificação de Fonte - Técnicas Baseadas em Dicionário (12 de abril de 2022). Retrieved April 12, 2022, from https://2122moodle.isel.pt/pluginfile.php/1156344/mod_resource/content/3/7.Codificacao%20de%20Fonte%20-%20Dicion%C3%A1rio.pdf