



**Departamento de Engenharia de Eletrónica e Telecomunicações
e de Computadores**

Comunicação Digital – Módulo 1

Autores:	50536	Fábio Silva
	50553	Bruno Raposo

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

Docente: Artur Ferreira

22 – 04 – 2024

<< Esta página foi intencionalmente deixada em branco >

Índice

Exercício 1	2
Função de Contagem de Bits de um Inteiro	2
Função de Fibonacci	2
Função de Ocorrência de Símbolos	3
Função do Histograma de Símbolos	4
Função do Ficheiro Revertido	5
Exercício 2	6
Função de Progressão Aritmética	6
Função Fatorial	8
Função Mínimo Múltiplo Comum (MMC)	8
Função de Números Primos	9
Função de Frequência de Símbolos em Arquivos	10
Exercício 3	11
Análise à informação própria e à entropia e recolha do histograma	11
Estimativas de ocorrências de símbolos e pares de símbolos	13
Exercício 4	14
Implementação Genérica de Fonte de Símbolos	14
Geradores de Símbolos Específicos	16
Códigos PIN	16
Chaves do Euromilhões	16
Palavras-passe robustas	17
Compressão de Dados e Taxa de Compressão	18
Exercício 5	18
Sistemas Criptográficos	19
Cifra de Vernam	19
Realização do Exercício	19
Exercício 6	21
Binary Symmetric Channel (BSC)	21
Outros Exemplos de Transmissão de Sequências de Bits	22
Transmissões de ficheiros	22
Bibliografia	23

Exercício 1

Este exercício consiste num conjunto de funções, que deverão ser implementadas em linguagem C, com simples objetivos de forma a sujeitar uma familiarização com a linguagem e com conceitos básicos de comunicação digital.

Função de Contagem de Bits de um Inteiro

Esta função tem como objetivo contar e mostrar o **número de bits a 0 e a 1** de um valor inteiro, passado como parâmetro. Para tal, recorreu-se a um ciclo que verifica, bit a bit, se o mesmo tem valor 0 ou valor 1.

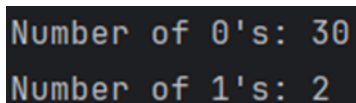
Exemplo de chamada à função:

Parâmetros:

- **Valor inteiro (val):** 17

Resultados Teóricos: O valor inteiro 17 será representado em binário por 0000 0000 0000 0000 0000 0000 0001 0001, resultando em um total de 30 bits a 0 e 2 bits a 1.

Com isto, podemos verificar os **resultados experimentais** representados por meio da **Figura 1**, que ilustra os valores efetivos retornados pela nossa função.



```
Number of 0's: 30
Number of 1's: 2
```

Figura 1 - Número de bits a 0 e a 1 do valor inteiro 17

Função de Fibonacci

O objetivo desta função consiste na **representação dos N primeiros termos da sequência de Fibonacci**. A sequência de Fibonacci é uma sequência numérica em que cada termo a partir do terceiro é a soma dos dois antecessores, sendo o primeiro e o segundo termo os **números 0 e 1**, respetivamente. Em outros casos, também se considera que os dois primeiros termos desta sequência são o número 1, no entanto, a nossa função foi implementada assumindo o primeiro caso.

Exemplo de chamada à função:

Parâmetros:

- **Número de termos a representar (N):** 16

Resultados Teóricos: Sendo que N é 16, então a função deverá imprimir na consola os 16 primeiros termos da sequência de Fibonacci, que são: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610.

Com isto, podemos verificar os **resultados experimentais** representados por meio da **Figura 2**, que ilustra os valores efetivos retornados pela nossa função.

```
16 first numbers from fibonnaci sequence: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

Figura 2 - 16 primeiros termos da sequência de Fibonacci

Função de Ocorrência de Símbolos

Esta função visa a calcular a **frequência de ocorrência de um único símbolo** em um ficheiro, retornando -1 no caso do ficheiro escolhido não apresentar nenhuma ocorrência do símbolo.

Exemplo de chamada à função com sucesso:

Parâmetros:

- **Símbolo cuja frequência de ocorrência vai ser calculada (symbol):** 's'
- **Ficheiro no qual o cálculo vai ser efetuado:** "alice29.txt"

Resultados Teóricos: Uma vez que o ficheiro escolhido consiste na famosa obra infantil "Alice no País das Maravilhas", é possível concluir que o símbolo 's' irá ocorrer pelo menos 1 vez.

Com isto, podemos verificar os **resultados experimentais** representados por meio da **Figura 3**, que ilustra o número de ocorrências do símbolo escolhido.

```
Number of s instances: 6277
```

Figura 3 - Número de ocorrências do símbolo 's' no ficheiro "alice29.txt"

Exemplo de chamada à função sem sucesso:

Parâmetros:

- **Símbolo cuja frequência de ocorrência vai ser calculada (symbol):** '!
- **Ficheiro no qual o cálculo vai ser efetuado:** "a.txt"

Resultados Teóricos: Tendo em conta que o pequeno ficheiro escolhido contém o seguinte conteúdo:

"Comunicação Digital - início de ficheiro de teste.

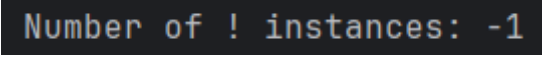
1234567890

The quick brown fox jumps over the lazy dog.

Comunicação Digital - final de ficheiro de teste."

É possível concluir que o valor retornado pela nossa função será -1, que representará o caso em que não existem ocorrências do símbolo '!' no ficheiro escolhido.

Com isto, podemos verificar os **resultados experimentais** representados por meio da **Figura 4**, que ilustra o caso em que não são encontradas ocorrências do símbolo escolhido.



```
Number of ! instances: -1
```

Figura 4 - Retorno da função em análise no caso do ficheiro não apresentar ocorrências do símbolo escolhido

Função do Histograma de Símbolos

Esta função tem como objetivo **representar o histograma de símbolos de um certo ficheiro** escolhido. Um histograma de símbolos consiste numa representação gráfica que demonstra a frequência de ocorrência de todos os símbolos presentes em um ficheiro.

Exemplo de chamada à função:

Parâmetros:

- **Ficheiro do qual se vai determinar o histograma de símbolos:** "alice29.txt"

Resultados Teóricos: Nesta função não é possível apresentar resultados concretos devido à dimensão do ficheiro em análise. Contudo, é possível concluir que, uma vez que se está a analisar a obra "Alice e o País das Maravilhas", todos os símbolos correspondentes a pontuação e às letras do alfabeto ocorrem pelo menos uma vez.

Com isto, podemos verificar o **histograma obtido** através da chamada da função, por meio da **Figura 5**. Com este histograma é possível concluir que o valor da entropia não será muito elevado, devido à grande diferença de ocorrência dos símbolos, sendo que o símbolo ' ' apresenta o maior número de ocorrências em comparação com os outros símbolos.

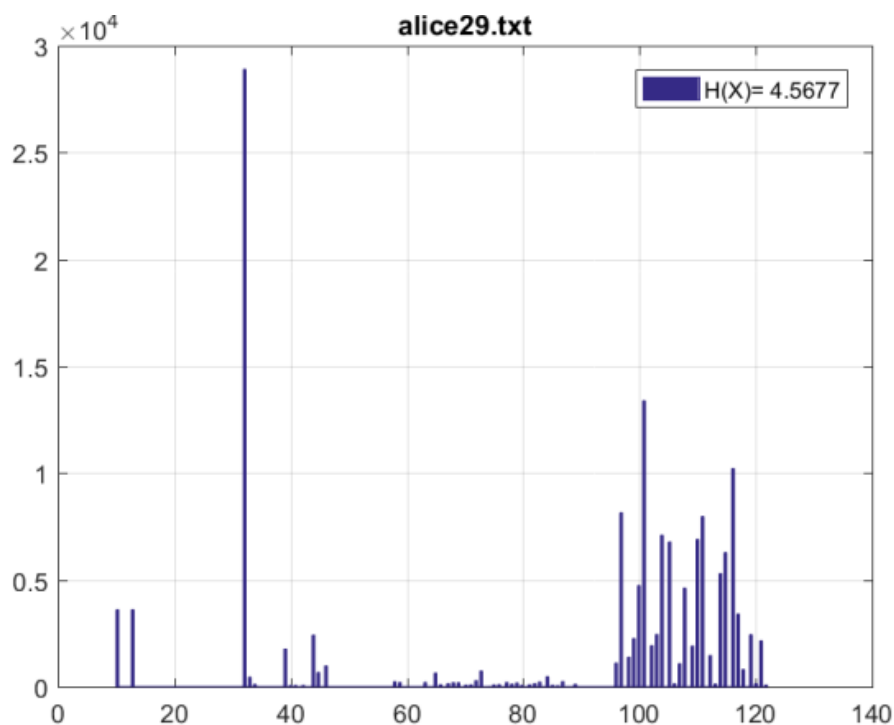


Figura 5 - Histograma do ficheiro alice29.txt

Função do Ficheiro Revertido

O propósito desta função consiste em **gerar um novo ficheiro, cujo conteúdo consiste no conteúdo de um outro ficheiro, invertido**. Para tal, recorreu-se a operações da biblioteca C como **ftell** e **fseek** que permitem uma fácil navegação no conteúdo de ficheiros.

Exemplo de chamada à função:

Parâmetros:

- **Ficheiro de input:** a.txt
- **Ficheiro de output:** testFile.txt

Resultados Teóricos: O resultado esperado com a chamada desta função será a geração de um novo ficheiro de texto designado de "testFile.txt" que irá conter todo o conteúdo presente no ficheiro "a.txt" invertido.

Com isto, podemos verificar os **resultados experimentais** por meio das **Figura 6 e 7**, que ilustram o conteúdo do ficheiro de input e o conteúdo do ficheiro de output, respetivamente.


```
Comunicao Digital - inicio de ficheiro de teste.
1234567890
The quick brown fox jumps over the lazy dog.
Comunicao Digital - final de ficheiro de teste.
```

Figura 6 - Conteúdo do ficheiro de input

```
.etset ed oriehcif ed lanif - latigiD oacinumoc
.god yzal eht revo spmuj xof nworb kciug ehT
0987654321
.etset ed oriehcif ed oiconi - latigiD oacinumoc
```

Figura 7 - Conteúdo do ficheiro de output

Exercício 2

Nestes exercícios, vamos explorar a implementação de várias funções em Python. Cada função aborda um conceito específico e será acompanhada pelos respetivos resultados teóricos e experimentais.

Função de Progressão Aritmética

Esta função irá gerar os **N primeiros termos** de uma progressão aritmética com um **primeiro termo (u)** e uma **razão (r)** especificados como parâmetros. A PA é uma sequência numérica em que cada termo é obtido somando-se a razão ao termo anterior. Demonstraremos como esta função se comporta com diferentes valores de N, u e r.

Variante 1:

- **Número de Termos (N):** 5
- **Primeiro Termo (u):** 10
- **Razão (r):** 3

Resultados teóricos: Os primeiros 5 termos da PA são: **10, 13, 16, 19 e 22**. Observamos que a razão positiva de 3 resulta em um aumento constante nos valores dos termos.

Variante 2:

- **Número de Termos (N):** 8
- **Primeiro Termo (u):** 5
- **Razão (r):** 2

Resultados teóricos: Os primeiros 8 termos da PA são: **5, 7, 9, 11, 13, 15, 17 e 19**. A razão de 2 leva a um crescimento constante, mas com um aumento menor a cada termo.

Variante 3:

- **Número de Termos (N):** 6
- **Primeiro Termo (u):** 2
- **Razão (r):** -1

Resultados teóricos: Os primeiros 6 termos da PA são: **2, 1, 0, -1, -2 e -3**. A razão negativa de -1 resulta em um decréscimo constante nos valores dos termos.

Agora, podemos verificar os **resultados experimentais** representados por meio da **Figura 8**, que ilustra os valores efetivos retornados pela nossa função. Esses valores foram obtidos quando a função recebeu os parâmetros descritos nas três variantes mencionadas anteriormente. Esta figura confirma os outputs esperados e permite-nos analisar o comportamento da progressão aritmética com diferentes valores de **N**, **u** e **r**.

```
vscode →/workspaces/CD $ python ex2.py
Progressão aritmética 1:
10, 13, 16, 19, 22,
Progressão aritmética 2:
5, 7, 9, 11, 13, 15, 17, 19,
Progressão aritmética 3:
2, 1, 0, -1, -2, -3,
```

Figura 8 - Resultados da Função de Progressão Aritmética (PA)

Função Fatorial

Aqui, criamos uma função que calcula o **fatorial** de um número inteiro **N**. O fatorial de um número é o produto de todos os inteiros positivos menores ou iguais a ele. Demonstraremos mais à frente como esta função lida com números maiores e menores.

Segue-se abaixo a **Função Fatorial** e a respetiva análise com 3 variantes distintas, que exemplificam a maneira como a função lida com números maiores e menores.

Variante 1: Fatorial de 3

- **Número Natural (N): 3**

Resultados teóricos: $3! = 3 \times 2 \times 1 = 6$

Variante 2: Fatorial de 5

- **Número Natural (N): 5**

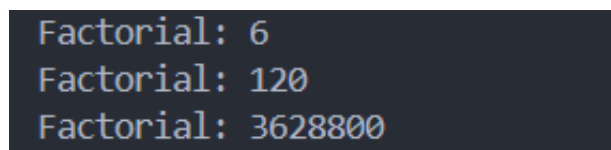
Resultados teóricos: $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Variante 3: Fatorial de 10

- **Número Natural (N): 10**

Resultados teóricos: $10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3628800$

Na **Figura 9**, apresentamos os **resultados experimentais** obtidos ao executar a **Função Fatorial** com diferentes valores de **N**. Esses valores foram diretamente comparados com os resultados teóricos esperados, e coincidem. Com esta análise podemos também, verificar a curva ascendente que demonstra o crescimento exponencial do fatorial à medida que o número natural aumenta.



```
Factorial: 6
Factorial: 120
Factorial: 3628800
```

Figura 9 - Resultados Experimentais da Função Fatorial

Função Mínimo Múltiplo Comum (MMC)

A terceira função determina o **MMC** entre dois números inteiros **a** e **b**. O **MMC** é o menor múltiplo comum a ambos os números. Analisaremos como esta função se comporta com diferentes pares de valores.

Variante 1: MMC entre 10 e 15

- **a:** 10
- **b:** 15

Resultados teóricos: $\text{MMC}(10, 15) = 30$

Variante 2: MMC entre 11 e 20

- **a:** 11
- **b:** 20

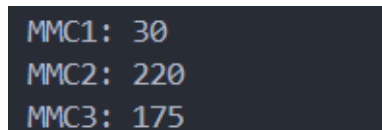
Resultados teóricos: $\text{MMC}(11, 20) = 220$

Variante 3: MMC entre 7 e 25

- **a:** 7
- **b:** 25

Resultados teóricos: $\text{MMC}(7, 25) = 175$

Os resultados teóricos obtidos para o **Mínimo Múltiplo Comum (MMC)** entre diferentes pares de valores foram consistentes com os resultados experimentais. Na **Figura 10**, podemos observar a coerência entre os valores teóricos e práticos.



```
MMC1: 30
MMC2: 220
MMC3: 175
```

Figura 10 - Resultados Experimentais da Função MMC

Função de Números Primos

A próxima função (**Função de Geração de Números Primos**) apresenta todos os **números primos** dentro de um intervalo definido pelos limites **left** e **right**, inclusivamente.

Vamos analisar três variantes distintas. Cada variante será acompanhada pelos resultados teóricos e experimentais obtidos.

Variante 1: Números Primos entre 1 e 25

- **Limite esquerdo (left):** 1
- **Limite direito (right):** 25
- Percorre todos os números no intervalo de 1 a 25.
- Testa individualmente cada número para verificar se é primo ou não. Exibe os números primos encontrados.

Resultados teóricos: 2, 3, 5, 7, 11, 13, 17 e 19.

Variante 2: Números Primos entre 1 e 50

- **Limite esquerdo (left):** 1
- **Limite direito (right):** 50
- Percorre todos os números no intervalo de 1 a 50.
- Testa individualmente cada número para verificar se é primo ou não. Exibe os números primos encontrados.

Resultados teóricos: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 e 47.

Variante 3: Números Primos entre 10 e 100

- **Limite esquerdo (left):** 10
- **Limite direito (right):** 100
- Inicia a procura a partir do número 10 e termina no 100.
- Filtra e exibe apenas os números primos encontrados.

Resultados teóricos: 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 e 97.

Na **Figura 11**, apresentamos os resultados experimentais obtidos pela nossa função. Esta figura confirma que os resultados práticos, exibidos na consola do nosso IDE, correspondem exatamente aos valores teóricos previstos.

```
Primes1: [2, 3, 5, 7, 11, 13, 17, 19, 23]
Primes2: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
Primes3: [11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

Figura 11 - Resultados Experimentais da Função de Geração Números Primos

Função de Frequência de Símbolos em Arquivos

Por fim, criámos uma função que exibe todos os **símbolos** presentes em um arquivo, cuja **frequência de ocorrência** é superior a uma **percentagem** passada como parâmetro.

Para obter a ocorrência de símbolos em um arquivo, lemos o conteúdo, contamos cada símbolo e no final filtramos pela frequência desejada.

Ao aplicarmos a função a um ficheiro de texto com a história “**Alice’s Adventures in Wonderland**” e uma **percentagem de 5%**, escrita originalmente em inglês, obtivemos os resultados ilustrados na **Figura 12**. Esses resultados têm implicações na **compreensão textual** e na **otimização de processos digitais**. Por exemplo, ao criar algoritmos de busca ou processadores de linguagem natural, considerar a frequência de símbolos pode melhorar a eficiência e a precisão. Além disso, a análise de símbolos pode revelar padrões culturais e estilísticos.

```
Symbol: ' ' - Occurencies: 28901
Symbol: 'e' - Occurencies: 13381
Symbol: 'a' - Occurencies: 8149
Symbol: 'o' - Occurencies: 7965
Symbol: 't' - Occurencies: 10212
```

Figura 12 - Resultados Experimentais da Função de Frequência de Símbolos

Exercício 3

Neste exercício, é proposta a implementação de funções capazes de analisar fontes de símbolos. A análise realizada deverá retornar o valor da informação própria de cada símbolo da fonte, o valor da entropia, o histograma e estimativas de ocorrências de símbolos e pares de símbolos.

Análise à informação própria e à entropia e recolha do histograma

Para a realização das funções responsáveis pela recolha destes dados, será necessário explorar alguns conceitos básicos antes.

A cada ocorrência de um símbolo está associada a **informação própria** ou auto-informação. Esta é definida por:

$$I(x_i) = \log_2 \frac{1}{p(x_i)} = -\log_2 p(x_i) \text{ [bit]}$$

Sendo **$p(x_i)$** correspondente à probabilidade de o símbolo ocorrer no ficheiro, que pode ser determinada por: nº de ocorrências do símbolo / nº de símbolos no ficheiro.

A **informação própria** de um símbolo é inversamente proporcional à probabilidade de ocorrência do mesmo, isto é, quanto maior for **$p(x_i)$** menor será **$I(x_i)$** .

A **entropia** de um certo ficheiro consiste no **valor esperado (médio)** da informação própria de cada símbolo, sendo determinada através do seguinte cálculo:

$$H(X) = E_p \left[\log_2 \frac{1}{p(X)} \right] = \sum_{x_i} p(x_i) \log_2 \frac{1}{p(x_i)} \text{ [bit/simb]}$$

O valor desta, deverá ser **sempre** maior ou igual que 0 e menor ou igual que $\log_2(M)$, sendo M o número de símbolos presentes no ficheiro. No caso do valor da entropia ser zero, significa que a ocorrência dos símbolos é o mais previsível possível. Um bom exemplo será um ficheiro que apresenta apenas um único símbolo. No caso da entropia ser igual a $\log_2(M)$, significa que a ocorrência dos símbolos é o mais imprevisível possível. Este caso é alcançado quando todos os símbolos apresentam uma probabilidade de ocorrência e, conseqüentemente, um valor de informação própria iguais, entre si.

Por fim, tal como mencionado no exercício 1 deste módulo, um histograma de símbolos consiste numa representação gráfica que demonstra a frequência de ocorrência de todos os símbolos presentes em um ficheiro.

Para concretizar esta análise, foram criadas funções para determinar a probabilidade de ocorrência de cada símbolo do ficheiro, determinar a informação própria de cada símbolo com base na sua probabilidade de ocorrência previamente determinada, determinar o valor da entropia do ficheiro em análise recorrendo aos dois atributos previamente determinados e, por fim, estabelecer o histograma dos símbolos do ficheiro.

As seguintes figuras ilustram os valores dos atributos, mencionados, de uma pequena porção de símbolos do ficheiro fibonacci.kt.

```
Symbol: 'p' - Histogram: 17
Symbol: 'a' - Histogram: 68
Symbol: 'c' - Histogram: 28
Symbol: 'k' - Histogram: 1
Symbol: 'g' - Histogram: 4
Symbol: 'e' - Histogram: 49
Symbol: ' ' - Histogram: 277
```

Figura 13 - Porção do histograma de símbolos do ficheiro fibonacci.kt

```
Symbol: 'p' - Probability: 0.013732
Symbol: 'a' - Probability: 0.054927
Symbol: 'c' - Probability: 0.022617
Symbol: 'k' - Probability: 0.000808
Symbol: 'g' - Probability: 0.003231
Symbol: 'e' - Probability: 0.039580
Symbol: ' ' - Probability: 0.223748
```

Figura 14 - Probabilidades de ocorrência de uma porção de símbolos do ficheiro fibonacci.kt

```
Symbol: 'p' - Own Information: 6
Symbol: 'a' - Own Information: 4
Symbol: 'c' - Own Information: 5
Symbol: 'k' - Own Information: 10
Symbol: 'g' - Own Information: 8
Symbol: 'e' - Own Information: 4
Symbol: ' ' - Own Information: 2
```

Figura 15 - Informação própria de uma porção de símbolos do ficheiro fibonacci.kt

```
Symbols Entropy: 4
```

Figura 16 - Valor da entropia do ficheiro fibonacci.kt

Estimativas de ocorrências de símbolos e pares de símbolos

De forma a determinar, em percentagem, a estimativa de ocorrência dos símbolos ou pares de símbolos de um ficheiro, é essencial obter dois histogramas. Um histograma de símbolos e um histograma de pares de símbolos. Com estes, é realizada uma iteração por cada valor seu, tornando o mesmo em percentagem, recolhendo, por fim, os cinco símbolos e pares de símbolos com maior percentagem de ocorrência.

As **Figuras 17 e 18** ilustram os cinco símbolos e pares de símbolos com maior percentagem de ocorrência no ficheiro **ListaPalavrasPT.txt**. Dado que estamos a lidar com um ficheiro extenso e as funções implementadas terão de percorrer o mesmo ficheiro duas vezes, é expectável que os resultados demorem algum tempo a ser apresentados.


```

Top 5 symbols

Symbol: 'a' - Percentage: 10.519902
Symbol: 'e' - Percentage: 9.406010
Symbol: 'r' - Percentage: 8.174621
Symbol: 's' - Percentage: 7.953767
Symbol: '-' - Percentage: 6.888728

```

Figura 17 - Top 5 símbolos com maior percentagem de ocorrência em ListaPalavrasPT.txt

```

Top 5 symbol pairs

Symbol: '
' - Percentage: 6.683453
Symbol: 'ar' - Percentage: 3.848231
' - Percentage: 3.284557
Symbol: 'os' - Percentage: 2.298176
Symbol: 'r-' - Percentage: 3.020154

```

Figura 18 - Top 5 pares de símbolos com maior percentagem de ocorrência em ListaPalavrasPT.txt

Exercício 4

No âmbito deste exercício, propõe-se a implementação de **fontes de símbolos**. Essas fontes são essenciais para a geração de sequências de símbolos com base em probabilidades associadas a cada símbolo. O objetivo é explorar a teoria da informação e a entropia, avaliando como diferentes fontes se comportam e como a aleatoriedade influencia a geração de conteúdo.

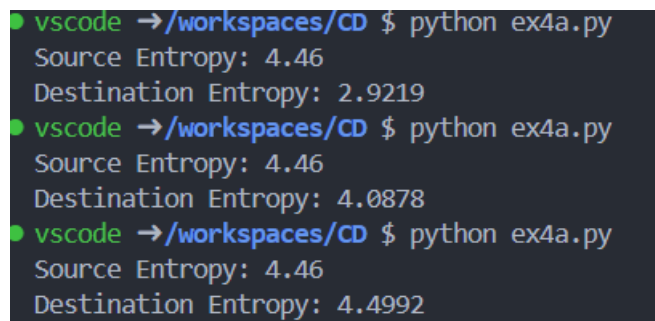
Implementação Genérica de Fonte de Símbolos

Criaremos uma fonte de símbolos genérica que gera arquivos contendo **N símbolos**, seguindo a **Função Massa de Probabilidade (FMP)** associada a um alfabeto de **M símbolos**. A implementação incluirá o cálculo da **entropia da fonte** e da sequência gerada. A **entropia** mede a incerteza ou imprevisibilidade da fonte.

Para concretizar a implementação da fonte de símbolos, começamos por definir uma função com o objetivo de criar **limites** para cada símbolo. Esses limites são estabelecidos com base na **Função Massa de Probabilidade (FMP)** fornecida e no **valor r**, que representa o limite máximo. Esta abordagem permitiu-nos criar uma estrutura que nos permite mapear valores aleatórios (gerados, por exemplo, por um

gerador de números aleatórios) para símbolos específicos, considerando as suas probabilidades. Em seguida, para gerar o símbolo aleatório, recorremos a um ciclo *for* que itera de **zero a N** (número de símbolos que se pretende gerar). Em cada iteração, é gerado um número aleatório, e posteriormente verificamos em qual intervalo (definido pelos limites) esse número aleatório se encontra e associamos o respetivo símbolo. Por fim, criámos uma função que efetua a escrita do ficheiro, dado uma *string* que é passada como parâmetro.

Na **Figura 19**, apresentam-se três variantes da execução da fonte de símbolos implementada. Para os testes, utilizamos as funções desenvolvidas no exercício 3 a fim de gerar o alfabeto de símbolos e as suas respetivas probabilidades. Note que o alfabeto é o mesmo nas três execuções, variámos apenas o número de símbolos gerados em cada uma delas. A seguir, prosseguiremos à análise detalhada dos resultados obtidos.



```
vscode → /workspaces/CD $ python ex4a.py
Source Entropy: 4.46
Destination Entropy: 2.9219
vscode → /workspaces/CD $ python ex4a.py
Source Entropy: 4.46
Destination Entropy: 4.0878
vscode → /workspaces/CD $ python ex4a.py
Source Entropy: 4.46
Destination Entropy: 4.4992
```

Figura 19 - Resultados experimentais de 3 execuções da fonte de símbolos

Execução com 10 Símbolos:

- A **entropia** da fonte é de **4,46**.
- A **entropia** do destino é de **2,9219**.

Análise: Com apenas 10 símbolos gerados, a entropia do destino é menor do que a entropia da fonte. Isso significa que a sequência gerada é menos imprevisível do que a fonte original. Isto aconteceu por termos utilizado um numero baixo de símbolos gerados. Pois o esperado é que a entropia do ficheiro destino se assemelhe à entropia do ficheiro destino.

Execução com 100 Símbolos:

- A **entropia** da fonte é de **4,46**.
- A **entropia** do destino é de **4,0878**.

Análise: Com 100 símbolos, a entropia do destino está mais próxima da entropia da fonte. Isto indica que a sequência gerada está a aproximar-se da entropia da fonte original.

Execução com 1000 Símbolos:

- A **entropia** da fonte é de **4,46**.
- A **entropia** do destino é de **4,4992**.

Análise: Com 1000 símbolos, a entropia do destino está mais próxima da entropia da fonte. Isto indica que a sequência gerada está mais aproximada da entropia da fonte original.

Geradores de Símbolos Específicos

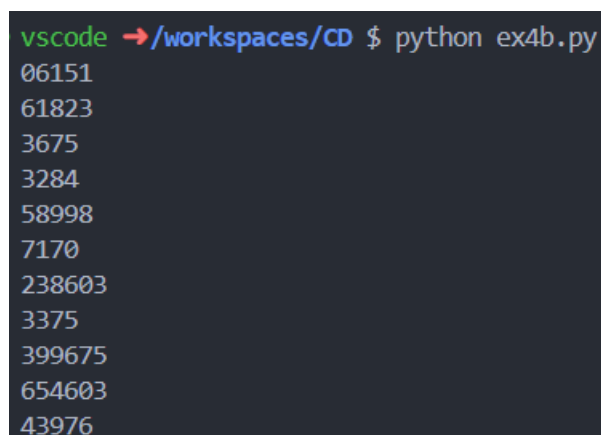
Utilizaremos a fonte de símbolos para criar três tipos de geradores:

- **Códigos PIN:** Geração de códigos PIN com **4 a 6 algarismos**.
- **Chaves do Euromilhões:** Geração de chaves para o **Euromilhões**.
- **Palavras-passe robustas:** Geração de senhas robustas com **8 a 12 caracteres**.

Apresentaremos **dez exemplos** de conteúdos gerados para cada tipo de gerador.

Códigos PIN

Implementámos uma função `pinGenerator` que tem como objetivo gerar códigos PIN de forma aleatória. Começámos por definir uma **Função Massa de Probabilidade (FMP)** para os **dígitos de 0 a 9**, atribuindo a cada dígito uma probabilidade de ocorrência de **10%**. Em seguida, determinámos aleatoriamente o número de dígitos no PIN (entre 4 e 6). Por fim, chamamos a fonte de símbolos genérica para gerar a sequência de símbolos (dígitos) com base na **FMP** definida e no número de dígitos. A **Figura 20** ilustra o resultado de 10 exemplos de PINs gerados.



```
vscode →/workspaces/CD $ python ex4b.py
06151
61823
3675
3284
58998
7170
238603
3375
399675
654603
43976
```

Figura 20 - Resultados Experimentais de 10 gerações de PINs

Chaves do Euromilhões

Nesta implementação, são gerados números para o jogo de sorte Euromilhões. Numa fase inicial, a função cria duas **Funções Massa Probabilidade (FMP)**. Uma para as chaves principais, compostas por 5 números aleatórios entre 1 e 50, e outra para as

estrelas que consistem em 2 números aleatórios entre 1 e 12. A distribuição dos números é uniforme, garantindo que todos tenham a mesma probabilidade de serem selecionados. Além disso, a função evita repetições nos números gerados modificando a **Função Massa Probabilidade (FMP)** para que o número que sai, seja removido do alfabeto e as restantes probabilidades dos números que ainda não saíram sejam ajustadas. A **Figura 21** ilustra o resultado de 10 exemplos de chaves geradas.

```
vscode → /workspaces/CD $ python ex4b.py
16 34 49 29 25 - 4 2
28 7 4 20 21 - 9 12
49 30 32 12 26 - 2 4
26 27 45 21 28 - 6 5
22 48 15 7 31 - 2 10
9 49 21 17 4 - 9 7
3 14 31 24 30 - 12 3
5 11 41 45 43 - 10 7
3 36 27 13 46 - 12 3
15 11 33 32 27 - 8 7
46 17 37 41 36 - 10 12
```

Figura 21 - Resultados Experimentais de 10 gerações de chaves Euromilhões

Palavras-passe robustas

Nesta secção abordaremos a geração de palavras-passe de forma aleatória. Além disso, explicaremos a implementação que fizemos para garantir, uma distribuição uniforme e a prevenção de repetições. Esta abordagem é fundamental para garantir a segurança e a robustez das palavras-passe.

Na implementação que fizemos, considerámos 4 categorias de caracteres diferentes:

- **Dígitos**
- **Letras Maiúsculas**
- **Letras Minúsculas**
- **Caracteres Especiais**

Nesta abordagem, destacamos a importância de garantir tanto a distribuição uniforme dos caracteres quanto a prevenção de repetições. No nosso código, definimos uma **Função Massa de Probabilidade (FMP)** que inclui as **4 categorias** de caracteres (números, letras maiúsculas, letras minúsculas e caracteres especiais). A probabilidade inicial para cada categoria é de **25%**. Além disso, criamos uma **FMP** específica para cada categoria, distribuindo a probabilidade de sair um símbolo dessa categoria de forma igualitária. À medida que os símbolos são gerados, reajustamos as probabilidades das quatro categorias. À categoria que foi selecionada é atribuída uma probabilidade de **1%**, enquanto as restantes recebem uma probabilidade de **33%**. Desta forma, procuramos evitar a repetição de caracteres pertencentes à mesma categoria, pois só assim é

possível garantir uma palavra-passe robusta. Na **Figura 22** representamos a geração de 10 palavras-passe seguindo estes critérios.

```
vscode →/workspaces/CD $ python ex4b.py
(4w"7zIg9<
~ABi0SvQ4e
w2;T@n&G
R1|0^T7K.
h6h/5a4"i?H7
]4z0E2\2X[
Jn.6w^2FJ
f/5k2D]5b
v8Ur8B~h7"2
7I5^9jR\
+d"tHo{A_2-L
```

Figura 22 - Resultados Experimentais de 10 gerações de palavras-passe

Compressão de Dados e Taxa de Compressão

Verificamos que a taxa de compressão do ficheiro de palavras-passe foi de apenas **18%**. Este valor sugere que a **entropia** do ficheiro é relativamente alta. Em outras palavras, os dados apresentam maior imprevisibilidade, pois estão bem distribuídos. Essa observação reforça o nosso esforço em evitar a repetição de símbolos pertencentes à mesma categoria e garantir que todas as categorias tivessem igual probabilidade de ocorrência. A entropia, neste contexto, desempenha um papel importante na análise da eficácia da compressão e na compreensão da distribuição dos dados.

1kpasswordGenerated	20/04/2024 18:31	Documento de Te...	11 KB
1kpasswordGenerated	20/04/2024 18:32	Pasta comprimida ...	9 KB

Figura 23 - Resultado da Compressão do ficheiro de palavras-passe

Exercício 5

O objetivo do seguinte exercício consiste em implementar um programa capaz de aplicar uma cifra de Vernam a uma imagem, tendo esta qualquer formato, e, de seguida, decifrar a mesma. Para tal, é necessário aprofundar os conhecimentos sobre sistemas criptográficos e cifra de Vernam.

Sistemas Criptográficos

Um sistema criptográfico consiste num sistema que, dado uma mensagem e uma chave, aplica essa chave na mensagem de forma a gerar uma nova mensagem ilegível que pode ser transmitida via canais desprotegidos, sem correr o risco de poder ser compreendida por terceiros.

Um sistema criptográfico apresenta 5 conceitos importantes:

1. **Plain Text (P)** – Consiste na mensagem em claro que se pretende enviar
2. **Cipher Text (C)** – Consiste na mensagem cifrada que vai ser transmitida via um canal desprotegido
3. **Key Space (K)** – Chave usada na cifra e/ou na decifra da mensagem
4. **Encipher (E)** – Cifra aplicada no **Plain Text** pelo emissor
5. **Decipher (D)** – Decifra aplicada no **Cipher Text** pelo recetor

Um sistema criptográfico pode apresentar dois tipos distintos de cifra:

1. **Cifra Simétrica** – A mesma chave, secreta, é utilizada na cifra do *Plain Text* e na decifra do *Cipher Text*
2. **Cifra Assimétrica** – Uma chave, pública, é utilizada na cifra do *Plain Text* e uma outra chave, secreta, é utilizada na decifra do *Cipher Text*

Cifra de Vernam

Em 1919, Gilbert Vernam propôs o Sistema de cifra One-Time-Pad, que consiste numa técnica de criptografia simétrica que oferece segurança perfeita quando aplicada nas seguintes condições:

- A chave utilizada é do mesmo comprimento que o texto em claro (*Plain Text*)
- A chave não é reutilizada
- A chave:
 - Deve ser aleatória
 - Deve ser pelo menos do comprimento do texto em claro (*Plain Text*)
 - Nunca deve ser reutilizada
 - Deve ser mantida secreta

Na cifra de Vernam, o texto em claro é cifrado através da operação **XOR** entre os símbolos deste e um conjunto de chaves secretas, aleatórias e distintas. Já a decifra é realizada recorrendo à mesma operação entre os símbolos do texto cifrado e as chaves correspondentes, pertencentes ao mesmo conjunto usado na cifra.

Realização do Exercício

Nesta secção, apresentaremos informações relevantes para a compreensão da solução proposta pelo nosso grupo para este exercício, bem como os resultados obtidos.

Na realização da nossa aplicação de cifra e decifra de uma imagem, recorreu-se à biblioteca **Pillow** da linguagem **Python** de forma a conseguir realizar operações de obtenção da informação de uma imagem e manipulação da mesma.

Neste exercício nem todas as condições para obter uma cifra que apresente segurança perfeita foram respeitadas, isto porque, a geração do conjunto de chaves utilizadas na cifra da imagem é aleatória, o que não garante a unicidade destas.

Por fim, todas as chaves utilizadas na cifra de cada pixel da imagem, ou de uma porção da imagem, foram guardadas nos pixels correspondentes num outro ficheiro de imagem nomeado **cypherKeys.png**, apresentando este formato de forma a não existir perda de informação.

As figuras seguintes apresentam os resultados da cifra e decifra de uma imagem **monocromática** com formato **bmp** e de uma imagem **colorida** com formato **jpg**.



Figura 24 – lena.bmp



Figura 25 – Ciphared
lena.bmp

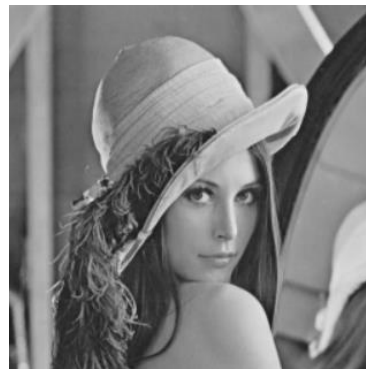


Figura 26 – Deciphared
lena.bmp



Figura 27 – barries.jpg

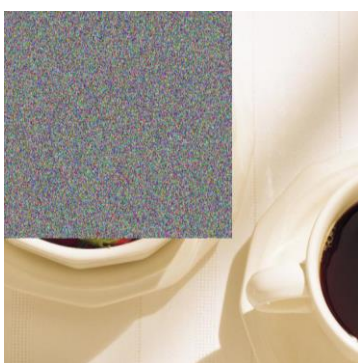


Figura 28 – Ciphared
barries.jpg



Figura 29 – Deciphared
barries.jpg

Exercício 6

Neste exercício, abordaremos a simulação de transmissão de dados em **canais binários simétricos (BSC)**. O BSC é um modelo utilizado para representar a transmissão de informações em sistemas de comunicação sujeitos a erros.

Binary Symmetric Channel (BSC)

"O BSC é um canal binário, ou seja, pode transmitir apenas um dos dois símbolos possíveis (normalmente 0 e 1). A transmissão não é perfeita e eventualmente o recetor recebe um bit com erro. Deve-se salientar que outros tipos de canal são capazes de transmitir mais de dois símbolos e até mesmo um número infinito de símbolos.

Este canal é frequentemente utilizado por ser um dos canais com ruído mais simples de ser analisado. Muitos problemas em teoria de comunicação podem ser simplificados para o BSC. Por outro lado, a capacidade de transmissão eficaz através do BSC pode originar soluções para canais de maior complexidade." [1]

Para a implementação de um **canal binário simétrico**, utilizamos um gerador de símbolos aleatórios que desenvolvemos para o [Exercício 4](#). Consoante o bit recebido, aplicamos uma **Função de Massa de Probabilidade (FMP)** a esse símbolo. Por exemplo, se o bit lido for '0', utilizamos uma FMP em que a probabilidade de sair '0' é $1 - p$, e a probabilidade de sair '1' é p . No caso em que o bit lido é '1', aplicamos a FMP inversa. Esta abordagem permite simular o que acontece num canal binário sujeito a erros, em que o p representa o **BER (Bit error rate)**.

Na **Figura 30**, apresentamos os resultados experimentais obtidos após a execução do nosso programa em três iterações. Utilizámos uma **probabilidade de erro elevada (50%)** para facilitar a simulação da troca de bits. Por meio desse exemplo, podemos observar que ocorreu a alteração de alguns bits entre o número binário inicial e o resultado após a passagem pelo **Canal Binário Simétrico (BSC)**. Essa representação reflete o que acontece na realidade durante a comunicação digital.

```
● vscode →/workspaces/CD $ python ex6a.py
  Ini:  10101111
  Res:  00001111
● vscode →/workspaces/CD $ python ex6a.py
  Ini:  10101111
  Res:  10000110
● vscode →/workspaces/CD $ python ex6a.py
  Ini:  10101111
  Res:  00011110
```

Figura 30 - Resultados experimentais da transmissão de uma sequência binária por BSC

Outros Exemplos de Transmissão de Sequências de Bits

Na **Figura 31**, são apresentados **três exemplos de transmissões sequenciais de bits**, com um número de bits de **1024**, **10240** e **102400**, respectivamente. A partir dos resultados experimentais, podemos inferir que o **BER (Bit Error Rate)** original passado à função é muito semelhante ao **BER** resultante da passagem de cada sequência de bits pelo canal. Essa aproximação é mais notável na terceira iteração, em que o número de bits é maior e reflete de forma mais precisa essa troca de bits. A análise dos resultados experimentais demonstra como a troca de bits afeta a transmissão de dados e como o **BER (Taxa de Erro de Bits)** se comporta nesse contexto.

```
vscode →/workspaces/CD $ python ex6b.py
Original BER: 0.3
New BER: 0.2919921875
Original BER: 0.3
New BER: 0.29423828125
Original BER: 0.3
New BER: 0.300224609375
```

Figura 31 - Outros resultados experimentais da transmissão sequencial por BSC

Transmissões de ficheiros

Na transmissão de ficheiros, observamos algo relativamente semelhante ao que ocorre nas transmissões de sequências binárias. O valor do **BER (Bit Error Rate)** para ficheiros com muitos bits é muito próximo ao BER original. No exemplo da **Figura 32**, verificamos que, para um BER de 10%, tivemos um total de **10736 erros**, o que ainda é um valor significativo e não corresponde à realidade da comunicação digital. Se observarmos excertos do ficheiro inicial (conforme a **Figura 33**) e compararmos com o ficheiro gerado após a passagem pelo BSC (conforme a **Figura 34**), podemos verificar uma dificuldade significativa na leitura do texto devido às várias trocas de bits.

```
vscode →/workspaces/CD $ python ex6c.py
Number of bits: 1187856
Number of errors: 10736
Original BER: 0.01
New BER: 0.009038132568257431
vscode →/workspaces/CD $
```

Figura 32 - Resultados experimentais da transmissão de ficheiros por BSC

```
Alice was beginning to get very tired of sitting by her sister  
on the bank, and of having nothing to do: once or twice she had  
peeped into the book her sister was reading, but it had no  
pictures or conversations in it, `and what is the use of a book,'  
thought Alice `without pictures or conversation?'
```

Figura 33 - Texto antes da transmissão por BSC

```
A|lice Was(beginning to get very tkred of sit4ing b{ her sister  
on the bank, and of iaving nothing to do: oncE or twice she had  
pueped"into the book `er sister was reading, bu4 it had no  
ryctures /r!contersations iN iT, `a^d uhat is the qse of a book,'  
thought Aliae"~withouv pictures or coîvârsathon='
```

Figura 34 - Texto depois da transmissão por BSC

Bibliografia

- [1] "https://pt.wikipedia.org/wiki/Canal_binário_simétrico," [Online].