

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

**Comunicação Digital – Módulo 2**

|  |  |  |
| --- | --- | --- |
| 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

03 – 05 – 2024

<< Esta página foi intencionalmente deixada em branco >

Índice

[Exercício 1 2](#_Toc167726709)

[**Códigos de controlo de erros** 2](#_Toc167726710)

[**Ausência de código de controlo de erros (Alínea i)** 3](#_Toc167726711)

[**Código de repetição (Alínea ii)** 5](#_Toc167726712)

[**Código de Hamming (Alínea iii)** 7](#_Toc167726713)

[Exercício 2 10](#_Toc167726714)

[Função de Progressão Aritmética 10](#_Toc167726715)

[Função Fatorial 11](#_Toc167726716)

[Função Mínimo Múltiplo Comum (MMC) 12](#_Toc167726717)

[Função de Números Primos 13](#_Toc167726718)

[Função de Frequência de Símbolos em Arquivos 14](#_Toc167726719)

[Exercício 3 14](#_Toc167726720)

[Análise à informação própria e à entropia e recolha do histograma 15](#_Toc167726721)

[Estimativas de ocorrências de símbolos e pares de símbolos 17](#_Toc167726722)

[Exercício 4 17](#_Toc167726723)

[Implementação Genérica de Fonte de Símbolos 18](#_Toc167726724)

[Geradores de Símbolos Específicos 19](#_Toc167726725)

[Códigos PIN 19](#_Toc167726726)

[Chaves do Euromilhões 20](#_Toc167726727)

[Palavras-passe robustas 20](#_Toc167726728)

[Compressão de Dados e Taxa de Compressão 21](#_Toc167726729)

[Exercício 5 22](#_Toc167726730)

[Sistemas Criptográficos 22](#_Toc167726731)

[Cifra de Vernam 22](#_Toc167726732)

[Realização do Exercício 23](#_Toc167726733)

[Exercício 6 24](#_Toc167726734)

[Binary Symmetric Channel (BSC) 24](#_Toc167726735)

[Outros Exemplos de Transmissão de Sequências de Bits 25](#_Toc167726736)

[Transmissões de ficheiros 26](#_Toc167726737)

[Bibliografia 27](#_Toc167726738)

# Exercício 1

Neste exercício, é proposta a **simulação da transmissão de ficheiros sobre o BSC implementado** no último exercício do módulo anterior. Adicionalmente, os testes realizados ao BSC recorrerão a **3 configurações distintas**, sendo estas:

1. Ausência de códigos de controlo de erros.
2. Código de repetição (3,1), em modo correção.
3. Código de Hamming (7,4), em modo correção.

Para cada configuração serão apresentados **4 exemplos com valores de *p* crescentes**, onde, para cada exemplo, serão apresentados os **valores BER e BER’**, tal como o **número de palavras diferentes** entre o ficheiro recebido e o ficheiro original.

**Códigos de controlo de erros**

Os códigos de controlo de erros têm como função a deteção e, por vezes, a correção de erros que possam surgir em uma mensagem resultante de uma transmissão sujeita a erros. Esta deteção e correção são obtidas através da introdução de redundância na mensagem original, isto é, na adição de bits adicionais aos bits originais da mensagem.

Os códigos de controlo de erros lecionados em aula são: o código de repetição, o código de bit paridade par, o código de Hamming e o CRC (Cyclic Redundant Check). Destes códigos apresentados, recorreremos ao código de repetição e ao código de Hamming na resolução deste exercício, ambos em modo de correção.

Todos os códigos apresentados acima, com exceção do CRC, pertencem a um conjunto de códigos de controlo de erros designado de códigos lineares de bloco. Alguns conceitos importantes sobre este conjunto são:

* Cada *bloco*, isto é, cada conjunto de bits aos quais vão ser adicionados bits redundantes, de *k* bits de mensagem origina uma palavra de código com *n* bits (*k* bits da mensagem + *q* bits redundantes);
* O vetor nulo pertence ao código;
* A soma modular de quaisquer duas palavras do código resulta em uma outra palavra do código;
* Uma propriedade destes códigos é a distância mínima (dmin) que consiste no número de bits distintos entre palavras de código diferentes;
* O limite de bits errados que a mensagem pode ter até o código não ser capaz de indicar que ocorreu um erro na transmissão pode ser determinado através da seguinte fórmula:

Uma imagem com Tipo de letra, tipografia, texto, branco

Descrição gerada automaticamente

Figura 1 – Fórmula de determinação do limite máximo de bits errados para existir deteção de erros na transmissão (l = nº de bits errados; dmin = distância mínima)

* Por fim, o limite de bits errados até o código não ser capaz de corrigir qualquer erro quando detetado pode ser determinado através da seguinte fórmula:

Uma imagem com Tipo de letra, file, diagrama, branco

Descrição gerada automaticamente

Figura 2 – Fórmula de determinação do limite máximo de bits errados para existir correção de erros na transmissão (t = nº de bits errados; dmin = distância mínima)

**Ausência de código de controlo de erros (Alínea i)**

Esta configuração do BSC tem como objetivo simular a transmissão de um ficheiro com a obra “Alice e o País das Maravilhas”, sujeita a erros, **sem recorrer a nenhum código de controlo de erros**.

Para a realização desta simulação recorreu-se a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura ilustra os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

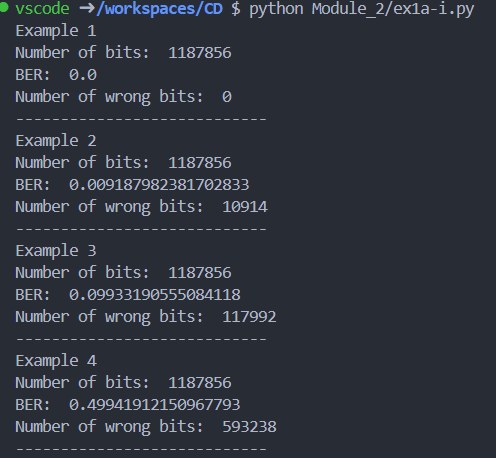


Figura 3 - Resultados da simulação sem código de controlo de erros

Uma vez que estes exemplos de simulação não recorrem a nenhum código de controlo de erros, não existe valor de BER’ a ser apresentado.

Ao observarmos a imagem, é possível compreender que quanto maior for o valor de *p*, ou seja, o BER, maior será o número de erros observados no ficheiro.

As seguintes figuras mostram um excerto dos ficheiros resultantes dos exemplos de transmissão do ficheiro com a obra “Alice e o País das Maravilhas”, pelo nosso BSC.

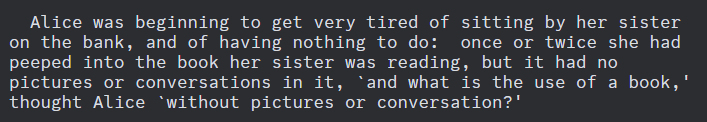


Figura 4 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,001

Uma imagem com texto, Tipo de letra, captura de ecrã, informação

Descrição gerada automaticamente

Figura 5 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,01

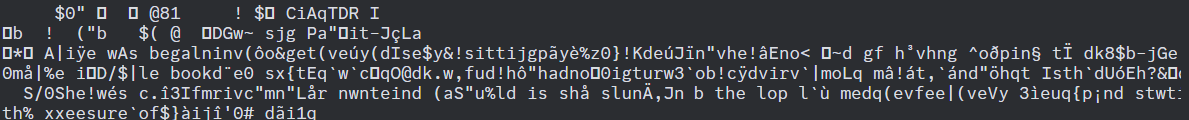


Figura 6 – Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,1

Uma imagem com texto, captura de ecrã, Tipo de letra, informação

Descrição gerada automaticamente

Figura 7 - Excerto do ficheiro resultante de uma transmissão ausente de controlo de erros com BER = 0,5

**Código de repetição (Alínea ii)**

O objetivo desta configuração do BSC consiste em, novamente, simular a transmissão da obra “Alice e o País das Maravilhas”, **recorrendo ao código de repetição (3,1) em modo de correção**, como forma de deteção e correção de erros durante a transmissão.

O código de repetição (3,1) consiste em um **código de controlo de erros** que, de forma a conseguir detetar uma possível troca de bits na transmissão, **adiciona a cada bit da mensagem duas repetições do mesmo**, antes desta ser transmitida. Com esta codificação, uma mensagem do tipo “0 1 1 0” resultará no código “000 111 111 000”.

Após a transmissão da mensagem codificada pelo BSC, o descodificador irá verificar, para cada **conjunto de 3 bits**, se algum destes foi modificado, indicando que houve um erro de transmissão quando **todos os bits do conjunto não são iguais**. Desta forma, este código de controlo de erros é capaz de **detetar erros de 1 e 2 bits na mensagem codificada**. Contudo, a sua **capacidade de correção de erros já se encontra limitada a apenas 1 bit**. Isto porque, após a transmissão da mensagem codificada, o descodificador vai verificar os 3 bits de cada conjunto e descodificar, os mesmos, no bit que se encontra em maioria, descodificando de forma errada, conjuntos de bits que tenham sofrido alterações em 2 ou 3 dos seus bits. Com esta descodificação, um código sujeito a erros do tipo “001 101 001 000” resultará na mensagem “0 1 0 0”.

Para a realização desta simulação recorreu-se, novamente, a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura apresenta os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

Uma imagem com texto, menu, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, menu, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 8 – Resultados da simulação com código de repetição (3,1)

Com as imagens apresentadas acima conseguimos perceber que:

* A implementação com o código de repetição irá resultar em um **menor número de erros na mensagem** recebida;
* Quanto maior o valor de *p*:
* **Maior será o número de erros na mensagem recebida**, quer a transmissão aplique o código de repetição, quer não;
* **Menor será a diferença entre o número de bits errados da transmissão com código de repetição e o número de bits errados da transmissão sem código de controlo de erros**. Isto porque, quanto maior for o BER da transmissão, maior será a probabilidade de cada bloco de bits apresentar erros em 2 ou 3 bits, impossibilitando a correção dos mesmos.

As figuras seguintes demonstram um excerto dos ficheiros resultantes dos exemplos de transmissão com código de repetição (3,1), da obra “Alice e o País das Maravilhas”, pelo nosso BSC.

Uma imagem com texto, captura de ecrã, Tipo de letra, informação

Descrição gerada automaticamente

Figura 9 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,001

Uma imagem com texto, Tipo de letra, captura de ecrã, informação

Descrição gerada automaticamente

Figura 10 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,01

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

Figura 11 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,1

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

Figura 12 - Excerto do ficheiro resultante de uma transmissão com código de repetição (3,1) e BER = 0,5

**Código de Hamming (Alínea iii)**

Nesta configuração do BSC pretende-se simular, de novo, a transmissão da obra “Alice e o País das Maravilhas”, **recorrendo ao código de Hamming (7,4) em modo de correção**, como forma de deteção e correção de erros durante a transmissão.

O código de Hamming (7,4) consiste em um outro **código de controlo de erros** que, de forma a conseguir detetar erros na transmissão, **realiza 3 equações de forma a gerar os 3 bits de paridade do código**, antes desta ser transmitida. Estas equações são:

Uma imagem com Tipo de letra, texto, branco, tipografia

Descrição gerada automaticamente

Figura 13 - Equações de paridade

Uma vez que o cálculo dos bits de paridade é fixo, este código de controlo de erros apresenta apenas 16 palavras de código possíveis:

* 0000 000
* 0001 111
* 0010 101
* 0011 010
* 0100 110
* 0101 001
* 0110 011
* 0111 100
* 1000 011
* 1001 100
* 1010 110
* 1011 001
* 1100 101
* 1101 010
* 1110 000
* 1111 111

O código de Hamming é **capaz de detetar erros até 2 bits e corrigir erros de 1 bit**, isto porque **todos os seus códigos apresentam uma distância mínima igual a 3**.

Após a transmissão da mensagem codificada pelo BSC, o descodificador vai, para **cada** **conjunto de 7 bits**, aplicar, novamente, as equações de paridade para os bits da mensagem, comparando estes com os bits de paridade da mensagem resultante da transmissão. Caso esta comparação conclua que os bits são diferentes, então ocorreu um erro na transmissão da mensagem.

De forma a corrigir quaisquer erros de 1 bit na mensagem recebida, o descodificador vai recorrer a uma tabela de síndromas com o objetivo de detetar o bit errado e corrigir o mesmo. Esta tabela de síndromas, específica para este código de controlo de erros, consiste em uma tabela que relaciona uma codificação de 3 bits (a síndroma) com o número do bit da mensagem que se encontra errado, sendo que se a codificação for igual ao vetor nulo, então a mensagem não apresenta nenhum erro. A síndroma poderá ser determinada através da seguinte operação:

**(e0 XOR r0) +(e1 XOR r1) + (e2 XOR r2)**,

sendo ***e*** os bits resultantes da aplicação das equações de paridade nos bits de mensagem do código recebido e ***r*** os bits de paridade do código recebido.

A tabela de síndromas mencionada é ilustrada na seguinte figura:

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 14 – Tabela de síndromas específica para o código de Hamming (7,4)

Para a realização desta simulação recorreu-se, novamente, a implementações realizadas no módulo anterior, nomeadamente, o BSC e a função de contagem de erros entre a mensagem recebida e a mensagem original.

A seguinte figura apresenta os resultados obtidos de 4 exemplos de simulação distintos com valores de *p* iguais a 0,001; 0,01; 0,1 e 0,5, respetivamente.

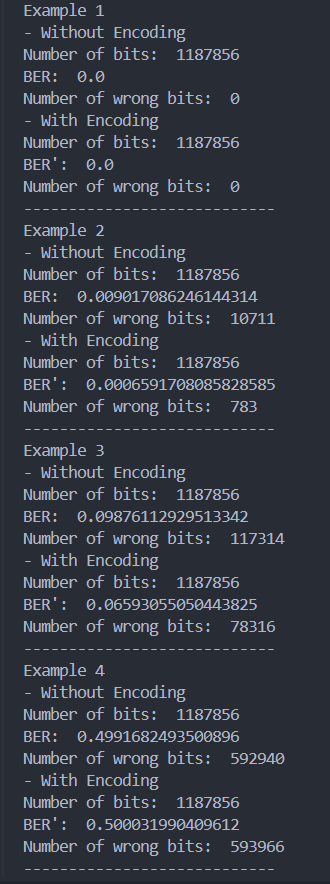
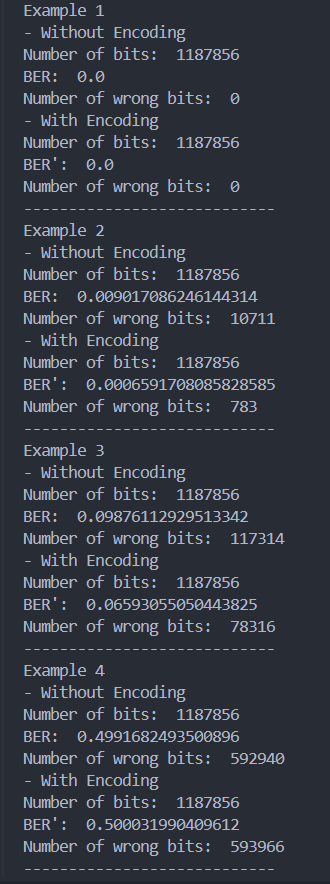


Figura 15 - Resultados da simulação com código de Hamming (7,4)

Com as imagens apresentadas acima conseguimos perceber que:

* A implementação com o código de Hamming irá resultar em um **menor número de erros na mensagem** recebida;
* Quanto maior o valor de *p*:
* **Maior será o número de erros na mensagem recebida**, quer a transmissão aplique o código de Hamming, quer não;
* **Menor será a diferença entre o número de bits errados da transmissão com código de Hamming e o número de bits errados da transmissão sem código de controlo de erros**. Isto porque, quanto maior for o BER da transmissão, maior será a probabilidade de cada bloco de bits apresentar erros em mais do que 1 bit, impossibilitando a correção dos mesmos.

As figuras seguintes demonstram um excerto dos ficheiros resultantes dos exemplos de transmissão com código de Hamming (7,4), da obra “Alice e o País das Maravilhas”, pelo nosso BSC.

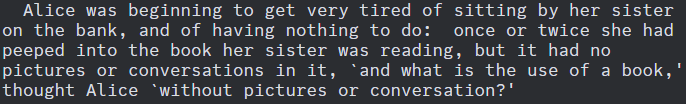


Figura 16 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,001

Uma imagem com texto, Tipo de letra, captura de ecrã, informação

Descrição gerada automaticamente

Figura 17 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,01

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 18 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,1

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 19 - Excerto do ficheiro resultante de uma transmissão com código de Hamming (7,4) e BER = 0,5

# 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**.

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

Figura 3 - Resultados da Função de Progressão Aritmética (PA)

## Função Fatorial

Aqui, criámos 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 analise 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 × 2 × 1 = 6

**Variante 2: Fatorial de 5**

* + **Número Natural (N)**: 5

**Resultados teóricos:** 5! = 5 × 4 × 3 × 2 × 1 = 120

**Variante 3: Fatorial de 10**

* + **Número Natural (N)**: 10

**Resultados teóricos:** 10! = 10 × 9 × 8 × 7 × 6 × 5 × 4 × 3 × 2 × 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.

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

Figura 4 - 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:** MMC(10, 15) = 30

**Variante 2: MMC entre 11 e 20**

* + **a:** 11
  + **b:** 20

**Resultados teóricos:** MMC(11, 20) = 220

**Variante 3: MMC entre 7 e 25**

* + **a:** 7
  + **b:** 25

**Resultados teóricos:** 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.

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

Figura 5 - 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.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 6 - 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.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 7 - 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 definada por:

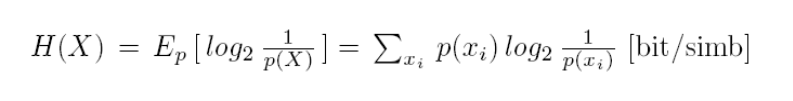
Uma imagem com Tipo de letra, tipografia, escrita à mão, caligrafia

Descrição gerada automaticamente

Sendo ***p(xi)*** 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 símboloé inversamente proporcional à probabilidade de ocorrência do mesmo, isto é, quanto maior for ***p(xi)*** menor será ***I(xi)***.

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



O valor desta, deverá ser **sempre** maior ou igual que 0 e menor ou igual que *log2(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 *log2(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, consequentemente, 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 8 - Porção do histograma de símbolos do ficheiro fibonacci.kt

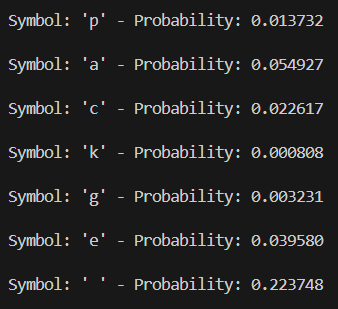
****

Figura 9 - Probabilidades de ocorrência de uma porção de símbolos do ficheiro fibonacci.kt

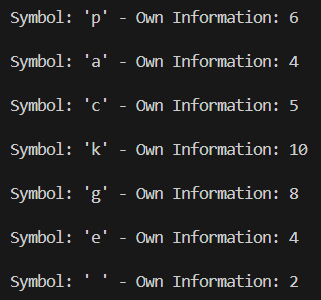
****

Figura 10 - Informação própria de uma porção de símbolos do ficheiro fibonacci.kt

****

Figura 11 - 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.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 12 - Top 5 símbolos com maior percentagem de ocorrência em ListaPalavrasPT.txt

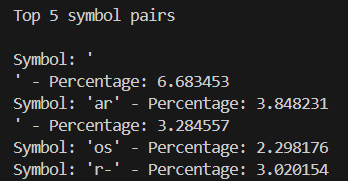


Figura 13 - Top 5 pares de símbolos com maior percentagem de ocorrência em ListaPalavrasPT.txt

# 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, começámos 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 14 - 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**.

**Analise**: 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 e 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**.

**Analise**: 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**.

**Analise**: 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.

**Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente**

Figura 15 - 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)**. Umapara 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 numero 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.

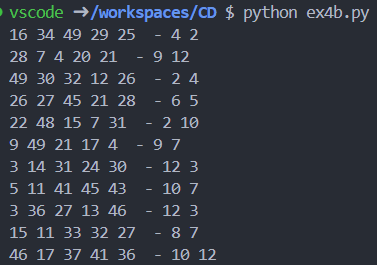
****

Figura 16 - 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.

**Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente**

Figura 17 - 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.

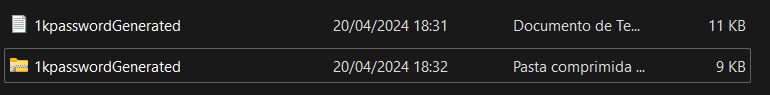


Figura 18 - Resultado da Compressão do ficheiro de palavras-passe

# 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***.

Uma imagem com acessório para a cabeça, vestuário, Cara humana, chapéu

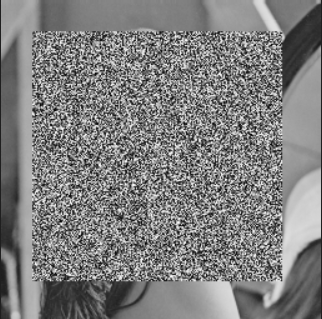
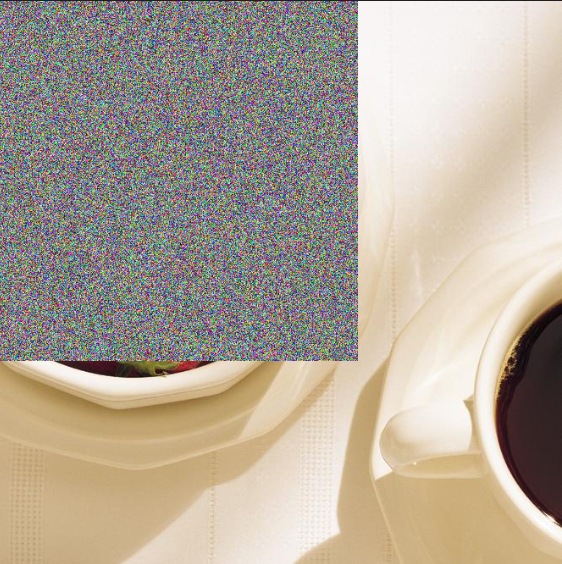
Descrição gerada automaticamente

Figura 26 – Deciphered lena.bmp

Figura 24 – lena.bmp

Figura 25 – Ciphered lena.bmp

Uma imagem com fruta, Morangos, Morango silvestre, Louça de servir

Descrição gerada automaticamente

Figura 28 – Ciphered barries.jpg

Figura 29 – Deciphered barries.jpg

Uma imagem com fruta, Morangos, Morango silvestre, Morango-silvestre

Descrição gerada automaticamente

Figura 27 – 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]**](#_Bibliografia_1)

Para a implementação de um **canal binário simétrico**, utilizamos um gerador de símbolos aleatórios que desenvolvemos para o [Exercício 4](#_4.1_Implementação_Genérica). 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.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

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**, respetivamente. 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

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.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 32 - Resultados experimentais da transmissão de ficheiros por BSC

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 33 - Texto antes da transmissão por BSC

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 34 - Texto depois da transmissão por BSC

# Bibliografia

|  |  |
| --- | --- |
| [1] | “https://pt.wikipedia.org/wiki/Canal\_binário\_simétrico,” [Online]. |