Universidade De Évora
Departamento de Informática
Teoria de Informação
Ano letivo 2019 - 2020

# Trabalho de Teoria de Informação

Alunos:
Luís Ressonha - 35003
Rúben Teimas - 39868

Docentes:
Miguel Barão

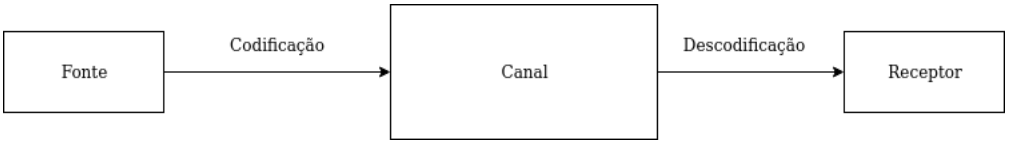
19 de Janeiro de 2020

# 1 Introdução

Este trabalho consiste na construção de um sistema de comunicação para enviar mensagens por um canal ruídoso.

Para a implementação deste sistema foram-nos fornecidos 2 ficheiros executáveis:
- Fonte;
- Canal.

A fonte gera mensagens que posteriormente serão codificadas e enviadas para o canal. Já no canal estas mensagens poderão ser corrompidas, sendo o nosso objetivo corrigir os ditos erros para que a mensagem possa chegar corretamente.




Representação do funcionamento do sistema

Para a compressão da mensagem utilizámos o algoritmo *LZW* e para a deteção e correção de erros o algoritmo de *Hamming (7,4)*.

# 2 Desenvolvimento

## 2.1 Fonte

Após uma análise da fonte concluímos que a esta pode gerar quatro mensagens diferentes:

- unknown
- broken-
- off----
- on-----

Como inicialmente pensámos em utilizar o algoritmo *Shannon Fano Elias* criámos um pequeno script para conhecer as probabilidades de cada palavra aparecer. Para ter uma amostra viável testámos a fonte a gerar 1 milhão de mensagens e o resultados foram os seguintes:

| **Mensagem** | **P(Mensagem)** |
|---|---|
| unknown | 0.10 |
| broken- | 0.21 |
| off---- | 0.14 |
| on----- | 0.55 |

Probabilidade da fonte gerar uma mensagem x

## 2.2 Canal

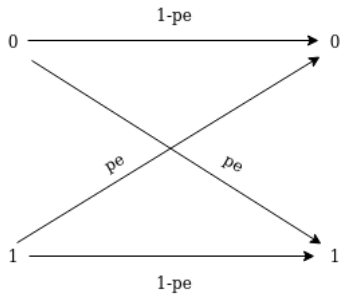
Um dos primeiros passos do trabalho foi analisar o comportamento do canal dado. Após alguns testes conseguimos perceber que o canal iria percorrer a string recebida e, poderia, alterar o último bit de cada letra, fazendo com que esta se transfomasse no caracter da tabela *ASCII* imediatamente acima ou abaixo.

Assim sendo o canal é portanto um **Canal Binário Simétrico** e a sua capacidade será

$$C = 1 - H(p_e)$$

em que

$$H(p_e) = -p_e log_2(p_e) - (1 - p_e) log_2(1 - p_e)$$




Representação do Canal

## 2.3 Compressão e Codificação

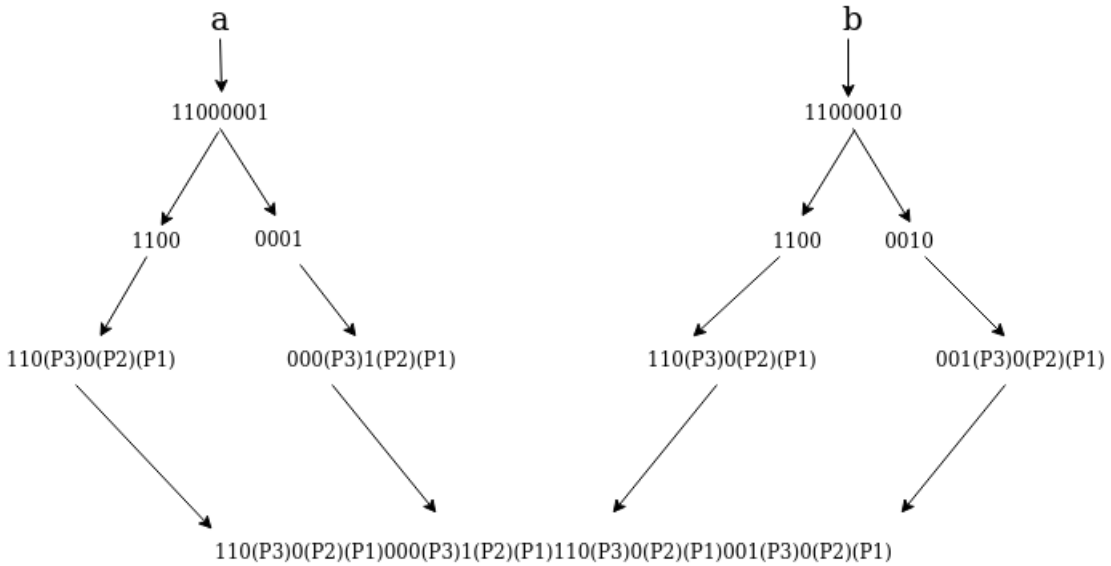
Inicialmente pensámos em comprimir a mensagem palavra a palavra, sendo cada palavra um simbolo da fonte e existindo assim 4 simbolos, mas ao vermos que pode existir mais do que um erro por palavra optámos por comprimir a mensagem letra a letra, ficando assim a fonte com 11 simbolos(as diferentes letras, o hífen e o *newline*).

Para a compressão/descompressão da mensagem utilizámos o algoritmo ***LZW*** pois devido à repetição de caractéres pareceu-nos ser uma boa opção.

Na compressão utilizámos um dicionário em *python* que inicialmente tem os caracteres *ASCII* até 122 (código da ultima letra minúscula). O algoritmo irá receber a mensagem e percorrê-la char a char até que encontre uma sequência de char's que ainda não exista no dicionário. Ao encontrá-la irá colocar essa mesma sequência no dicionário e ler a sequência de char's anterior a esta, colocando-a numa lista. O algoritmo irá fazer isto até que toda a mensagem esteja comprimida.

Após a compressão da mensagem, esta será codificada utilizando o código de ***Hamming***. Optámos por escolher um código de ***Hamming (7,4)***, i.e palavra de tamanho 7 com 4 *bits* de dados e 3 de paridade.

Para que o ***Hamming*** funcione optámos por codificar os números vindo do ***LZW*** para binário. A mensagem será então separada em grupos de 4, sendo posteriormente adicionados 3 bits de paridade. Os bits de paridade são calculados fazendo *XOR* entre os bits de dados. Para o primeiro bit de paridade fazemos *XOR* entre os bits da posição com **1** na posição menos significativas. Para o segundo bit fazemos o mesmo mas com os bits que têm *1* na segunda posição menos significativa e assim sucessivamente.




Representação da Codificação

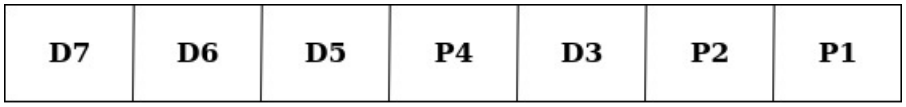
Após a codificação, uma mensagem que inicialmente tivesse $n$ palavras e teria um tamanho de $n*8$ terá agora um tamanho de

$$T = ((4+3)+(4+3)) * n$$

## 2.4 Descodificação e Descompressão

Após a passagem pelo canal os *bit's* poderão vir errados, como tal utilizamos o ***Hamming*** para detetar esses erros e corrigi-los.

Para descodificar a mensagem iremos dividi-la em grupos de 7. Em cada grupo iremos fazer *XOR* entre o respetivo bit de paridade e os bits que foram utilizados para o calcular na codificação. Fazemos isto para todos os bits de paridade do grupo e ao juntar os resultados desses *XOR's*, obtemos um número entre 0 e 7. Caso o número seja 0, signfica que não existe nenhum erro e podemos usar os 4 bits de dados. Se o número resultante for diferente de 0, então esse será o bit que está corrompido e que devemos inverter.




Representação de uma palavra do Código de Hamming

No final da descodificação de *Hamming* ficaremos com uma string com a representação binária da string comprimida inicialmente.

Para procedermos à descompressão da String utilizámos o algoritmo LZW conforme é explicado nos slides das aulas teóricas.

# 3 Conclusão

Embora os algoritmos de Compressão/Descompressão e Deteção/Correção de Erros nos pareçam estar bem implementados, nem sempre a mensagem que sai da fonte é a mesma recebida.

Acreditamos que isso se deva às propriedades do ***Código de Hamming*** e do **Canal**. Na nossa análise do canal, tal como dito anteriormente, reparamos que o canal poderia corromper o ultimo bit de um char, ora estando nós a mandar uma string com a representação binária do da compressão da mensagem, o canal poderá corromper mais do que um bit da letra, fazendo a correção de erros impossiveis, pois o ***Código de Hamming*** só permite corrigir um bit.

# 4 Referências

https://rosettacode.org/wiki/LZW_compression#Python