Universidade de Coimbra

SEBENTA
TEORIA DA INFORMAÇÃO

SEBENTA COM A MATÉRIA RELATIVA À CADEIRA "TEORIA DA INFORMAÇÃO"

PEDRO SECO

2024/2025

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA, FCTUC

Sebenta - Teoria da Informação - 2024/2025

Pedro Seco

A presente sebenta visa auxiliar no estudo da cadeira "Teoria da Informação" do 2ºano do 1º semestre da Licenciatura em Engenharia Informática. Trata-se de um material não oficial, criado apenas com o intuito de ajudar o estudo. Qualquer gralha avisa-me!

1. Introdução *(powerpoint 1)*

- **Compressão de dados**

Motivos para compressão de dados: diminuição dos requisitos de armazenamento e uso adequado da largura de banda disponível

Objetivos para compressão de dados: Identificar e remover a redundância/irrelevância da fonte

- **Tipos de compressão de dados:**

  - **Não-destrutiva**: a reconstrução é exata. Usada em Textos, Ficheiros Binários, etc…
  - **Destrutiva**: A reconstrução é aproximada (exploração do facto de que certos detalhes não são percebidos pelo ser humano ou que são menos importantes em termos de qualidade percebida)

Mas como reduzimos o tamanho dos dados? Com modelos de compressão

- **Tipos de compressão de dados:**

  - **Modelação Física (ou Processo)**: através de equações matemáticas. A parte difícil é descobrir/conhecer a equação
  - **Modelação preditiva**: através de modelos de previsão (Ex. método de compressão do PNG) Em cada linha, cada byte é previsto com base nos valores de bytes anteriores (vê se os pixels vizinhos são semelhantes entre amostras consecutivas). Este tipo de modelação envolve Redundância espacial e temporal.
  - **Modelação estatística**: Análise estatística dos símbolos mais frequentes (que terão menos bits ao serem codificados) ⇒ Huffman codes

De forma a protegermos os dados, recorremos à criptografia que os deixam de forma ilegível

- **Criptografia**

  - **Confidencialidade**: manter os dados secretos realizando encriptação de dados, garantir que os dados só podem ser desencriptados pela pessoa autorizada
  - **Integridade**: garantir que os dados não sofreram alteração
  - **Autenticação**: Saber a origem/destino dos dados/emissor
  - **Não repúdio**: Garantia de origem dos dados

- **Tipos de algoritmos de criptografía**

    - **Chave simétrica** (privada)
    - **Chave assimétrica** (pública e privada)
    - **Funções de Hashing**

2. Teoria da Informação e Codificação Entrópica *(powerpoint 2)*

- **Fonte de Informação**: Processo onde é gerada informação
    - Pode ser **Contínua** no tempo. Ex: Tensão Elétrica, ECG, etc…
    - Pode ser **Discreta**. Ex: Resultado de um inquérito, texto, etc…
    - Pode ser **Uni ou Multi-Dimensional**. Ex: Áudio, Imagem, etc…

De um modo simplificado, uma fonte é um gerador de símbolos que pertencem a um alfabeto. Cada um destes símbolos dispõe de uma probabilidade associada: $\{P(a_1), P(a_2), \ldots, P(a_n)\}$. Eis alguns exemplos de fontes: Fonte Binária ($A = \{0, 1\}$), Imagem Monocromática ($A = \{0, 1, 2, 3, \ldots, 255\}$).

- **Relação entre Informação e Incerteza**: Quanto maior for a Incerteza, maior será a Informação (Ex: Dizer que o Sporting sem o Gyokeres iria ser campeão no ano passado, era bastante improvável! Logo gera bastante informação).
- **Relação entre Informação e Probabilidade**: Quanto maior for a probabilidade (menor a incerteza), menor a informação. São inversamente proporcionais!
- Informação de dois eventos independentes = soma das informações individuais.

Fórmula para a Informação:

$$i(a) = -\log_b P(a) = \log_b \frac{1}{P(a)}$$

Outras propriedades:

$i(a, c) = i(a) + i(c)$

- **Entropia ou Informação Média**: Número médio de bits para codificar uma fonte de informação (AKA Entropia de 1ª ordem)

Fórmula para a Entropia:

$$H(a) = \sum_{i=1}^{n} P(a_i) i(a_i) = -\sum_{i=1}^{n} P(a_i) \log_2 P(a_i)$$

Nota: Entropia é uma medida de dispersão de distribuição!

Se a nossa distribuição for uniforme, ou seja, com probabilidades equiprováveis, isto quer dizer que todos os acontecimentos têm a mesma probabilidade, assim, a incerteza é máxima, logo, neste caso, conclui-se que a Entropia é máxima!

Eis alguns exemplos para um melhor entendimento, relativos à probabilidade de certas equipas ganharem o campeonato português:

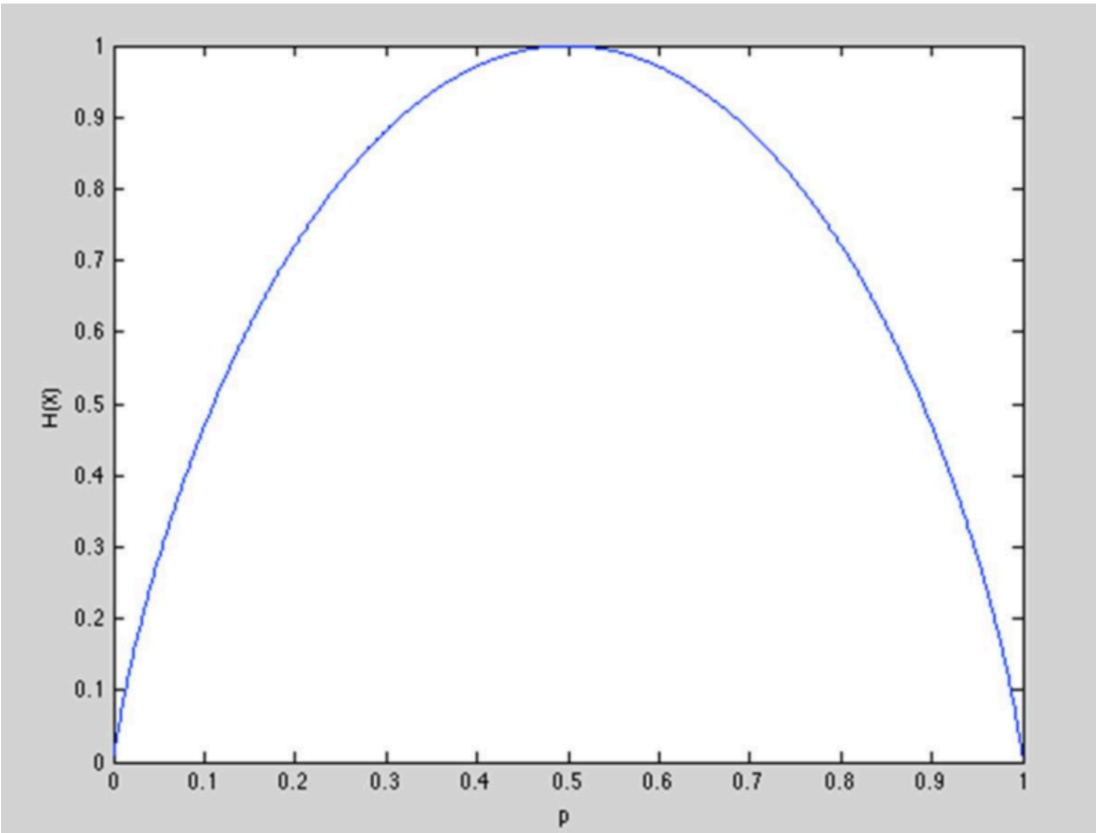
- ***Exemplo 1***:
  {Estoril, Nacional, Arouca, Farense}
  {0.2,0.35,0.2,0.25}

Neste exemplo, podemos observar que os elementos possuem todos probabilidades bastante semelhantes, logo, é de esperar que a entropia seja elevada. Calculando a entropia, H = 1.96.

- ***Exemplo 2***:
  {Benfica, Braga, Porto, Sporting}
  {0.6,0.15,0.2,0.05}

Neste exemplo, podemos observar que o mesmo não acontece como no Exemplo 1. Aqui, observamos que as probabilidades estão mais desequilibradas, com o Benfica a ter a maior parte do bolo das probabilidades, como é expectável. Assim, é de esperar que a Entropia seja mais baixa em relação ao exemplo anterior pois a incerteza, neste exemplo, é bem menor, pois temos mais certeza de quem poderá vir a ganhar o campeonato.

Gráfico que relaciona a Entropia e Incerteza:



Propriedades Importantes de Entropia:

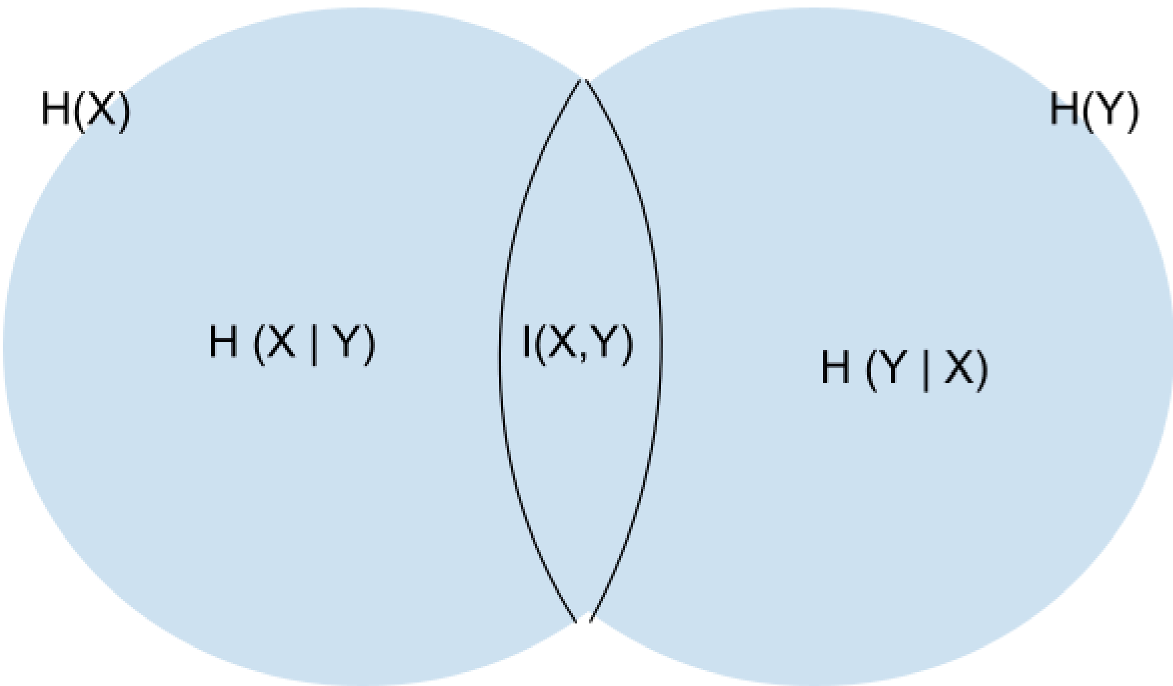
- $H(X) \in [0, log_2(\#A)]$, sendo #A o número de elementos do nosso Alfabeto
- Acontecimentos equiprováveis ⇒ Hmáx = $log_2(\#A)$
- Acontecimentos certos (Probabilidade = 1) ⇒ H(X) = 0

- **Entropia Conjunta** : H(X,Y) = $-\sum_i \sum_j P(X = x_i, Y = y_i) log_2 P(X = x_i, Y = y_i)$
  ou $H(X, Y)$ = H(X) + H(Y | X)
- **Entropia Condicional** :

  H(X|Y) = $-\sum_i \sum_j P(X = x_i, Y = y_i) log_2 P(X = x_i | Y = y_i)$

- $H(X) \geq 0$ se e só se $p_i = 1$
- $H(X, Y) = H(X) + H(Y)$ *e* $H(Y|X) = H(Y)$ se e só se os acontecimentos forem independentes
- Entropia no numpy: $H = -np.sum(p * np.log2(p))$, sendo p um array com as probabilidades

Nota: Em H(X | Y), estamos a restringir informação, logo a incerteza será menor, logo a entropia será menor do que em H(X).



$H(X, Y)$ = H(X) + H(Y | X)

I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) ⇒ **Informação Mútua**

Para concluir, eis as definições de Entropia Conjunta e Condicional:

- **Entropia Conjunta**: Informação dada por dois meios diferentes
- **Entropia Condicional**: Entropia de uma fonte de informação quando se conhece outra

No caso de acontecimentos independentes: P(X,Y) = P(X).P(Y) ⇒ I(X,Y) = 0
No caso de acontecimentos totalmente dependentes: I(X,Y) = H(X) ou H(Y)

Divergência Kullback-Leibler

↳ Trata-se de uma "distância" (não é rigorosamente uma distância porque não é simétrica) entre duas distribuições P(X) e Q(X) sobre o mesmo alfabeto. $D_{KL}(P, Q) \neq D_{KL}(Q, P)$

Fórmula para a Divergência Kullback-Leibler:

$$D_{KL}(P, Q) = \sum_{x \in Ax} P(x) log_2 \frac{P(X)}{Q(X)}$$

Nota: Se P(X) = Q(X), então a distância entre as duas distribuições é nula, logo $D_{KL}(P, Q) = 0$

Nota: A DLK toma sempre valores não-negativos

A entropia que temos vindo a falar trata-se da Entropia de 1ª ordem (existem outras ordens). A pergunta que se faz a seguir é: Será que é possível reduzir o valor da Entropia de alguma maneira? A resposta a esta pergunta é sim, e fazemos isto de várias maneiras, incluindo com modelações.

Modelações permitem reduzir a Entropia!

Tipos de modelações:
- Modelo físico: pode descorrelacionar os dados.
- Modelação de contexto: Pela utilização de cadeias de Markov e dicionários, por exemplo (veremos mais à frente).

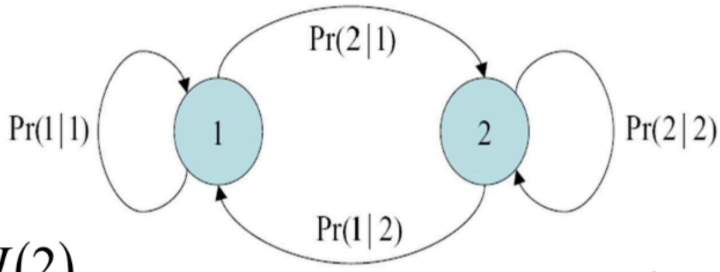
Agrupamento de Símbolos (Modelação do contexto):

1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2
Calculando a entropia normalmente, vemos que: P(1) = 0.25 = P(2) e P(3) = 0.5. H = 1.5bits
Fazendo um agrupamento de símbolos, juntando o 1 com o 2 e o 3 com 3 temos: P(1,2) = 0.5 = P(3,3). H = 0.5 bits

Vantagem e desvantagem do agrupamento de símbolos:

✅O agrupamento de símbolos permite GERALMENTE melhorar a estimação da entropia.
❌Número de arranjos possíveis aumenta exponencialmente com a dimensão do alfabeto (+ requisitos de memória e + complexidade).

Cadeias de Markov



$$H = P(1)H(1) + P(2)H(2)$$

Onde H(1) e H(2) são dados por:

$$H(1) \;=\; -\, P(1|1).\, log_2(P(1|1)) \,-\, P(2|1).\, log_2(P(2|1))$$
$$H(2) \;=\; -\, P(2|2).\, log_2(P(2|2)) \,-\, P(1|2).\, log_2(P(1|2))$$

Nota: Por norma, a modelação por cadeias de Markov permite reduzir a Entropia!

Regra da cadeia - Cultura geral, não sai no exame.

Propriedades e Conclusões - Entropia e Compressão

- H(X) ∈ $[0,\ log_2(\#A)],\ \#A\ corresponde\ ao\ número\ de\ elementos\ do\ alfabeto$
- $Quando\ os\ acontecimentos\ são\ equiprováveis,\ H(X) \;=\; log_2(\#A)$
- $A\ informação\ de\ contexto\ reduz\ a\ entropia\ H(X|Y) \;\leq\; H(X)\ (iguais\ se\ iid)$
- $É\ preferível\ estudar\ várias\ variáveis\ em\ simultâneo\ (agrupar)$:
  $$H(X, Y) \;=\; H(X) \;+\; H(Y|X) \;\leq\; H(X) \;+\; H(Y)$$

Teorema de Shannon

$$\boxed{\overline{L} \;\geq\; H(X)}$$

O que este teorema nos afirma é que o comprimento médio do código L nunca pode ser menor do que a Entropia H(X).

Códigos de prefixo - Códigos em que nenhuma palavra é prefixo de outra (**logo são unicamente descodificáveis e instantâneos**). Como saber se é código prefixo? 1. Desenhar a árvore binária com os códigos, 2. se as palavras forem folhas, então é código de prefixo, 3. Se um nó, que não seja folha, for uma palavra de código, não é um código de prefixo, logo não são instantâneos. São os códigos mais eficientes possíveis!

Códigos de prefixo ⇔ instantâneo ⇒ unicamente descodificável

O objetivo é obtermos códigos unicamente decodificáveis e instantâneos (são mais eficientes). Vejamos o que isto quer dizer:

- **Código unicamente descodificável:** Qualquer sequência só pode ser descodificada de uma e uma só forma (ausência de ambiguidade).
- **Código instantâneo:** O descodificador consegue determinar o momento em que o código está completo, não necessitando do próximo símbolo codificado para determinar o fim do presente.

Como verificamos isto?

Para verificar se um código pode ser unicamente descodificável (lembrar que não há relação de equivalência), tem de obedecer à seguinte desigualdade:

Desigualdade Kraft-McMillan

$$\sum 2^{-li} \leq 1$$

Para verificar se um código pode ser ótimo, tem de obedecer à seguinte igualdade:

$$\sum 2^{-li} = 1$$

Códigos ótimos ou de Shannon

Seja l e P(x) o comprimento e probabilidade de ocorrência. O comprimento de palavra verifica o seguinte:

$l_i = -\log_2 p_i$ e $\sum 2^{-li} = 1$

Desempenho dos códigos ótimos:

$$H(X) \leq \overline{L} \leq H(X) + 1$$

Quanto mais próximo L estiver de H(X) + 1, pior será o desempenho

Podemos concluir que quanto maior for *L*:
- Menor será a eficiência do código
- Maior será o desperdício de bits
- Menor será a taxa de compressão
- Mais recursos serão necessários para armazenamento/transmissão.

Além disso, para valores baixos de entropia (H(X)), o majorante é bastante elevado (H(X) + 1). Como resolvemos isto? Em vez de utilizarmos símbolos individuais, utilizamos agrupamento de símbolos:

$$\overline{L}(S^n) = n\overline{L}(S) \qquad H(S^n) = nH(S) \qquad H(S) \leq \overline{L} \leq H(S) + \frac{1}{n}$$

Ex:
$\overline{L}(S) = 2.4$
$\overline{L}(S^2) = 2.4 * 2$ => Agrupados 2 a 2

Códigos de Huffman e Códigos Adaptativos

Algoritmo para os Códigos de Huffman => Slides

(Pergunta exame?) Porquê utilizar códigos adaptativos de Huffman? R: Estes ajustam-se dinamicamente ao comprimento de código

Nos códigos adaptativos de Huffman, a árvore é sibilante (Peso dos nós maiores está sempre mais acima mais à direita ↑→), nos códigos iniciais temos de encontrar um "e" e "r" que satisfaçam a seguinte equação: $2^e + r = m$, sendo m o comprimento do alfabeto. Descobrimos primeiro o "e" máximo que verifica a equação e depois descobrimos "r".

- De $1 \leq k \leq 2r \Rightarrow$ Codificamos o número k-1 com e+1 bits (k é o indice da tabela, começar em 1)
- A partir de $k > 2r \Rightarrow$ Codificamos o número k-r-1 com e bits

Algoritmo => Ver slides

1. Codificação Lempel-Ziv *(slides 180-217, pp 1)*

Técnica de compressão de dados *lossless* que permite explorar padrões que se repetem. A utilização de um dicionário permite poupar alguns bits na transmissão. O dicionário é construído adaptativamente. Iremos abordar 3 algoritmos deste tipo de codificação: LZ77, LZ78 e LZW (que é uma variante do LZ78)

1. 1. LZ77

- Utiliza dicionário implícito
- Baseia-se em dois buffers: Search Buffer (SB) e Look Ahead Buffer (LAB).
- Ideia: Procurar no SB o maior padrão que ocorre no LAB
- Para codificar:
    - Próximo símbolo é enviado para o caso de não existir símbolo <0,0,símbolo>

- Codificar a posição relativa: <Offset, Length, código do próximo símbolo>, em que offset corresponde à distância da primeira letra do padrão reconhecido no LAB à primeira letra desse mesmo padrão no SB; Length corresponde ao comprimento do padrão encontrado no SB; Código do próximo símbolo é o próximo símbolo a ser codificado no LAB.

Exemplo: **Codificação**

- Usemos a mensagem = “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”
- Consideremos #SB = 8 e #LAB = 8
  (Verde - SB, Vermelho - LAB)

1. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   SB está antes da string e LAB corresponde inicialmente aos 8 primeiros caracteres da string. Vamos ao LAB e começamos por perguntar: “s” está no SB? Ora, o SB neste caso, está vazio, então “s” não está no SB, por isso temos de o codificar da seguinte forma:

   <0,0,”s”> = Não há nenhum padrão no SB, logo o offset é 0, como não há padrão então Length = 0, como não existe símbolo no SB, então pomos a letra “s”

   Avançamos o SB e LAB um caracter:

2. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos: “a” está no SB? Ora, o SB só contém agora a letra “s”, então “a” não está no SB, então temos de o codificar da seguinte forma:

   <0,0,”a”> = Não há nenhum padrão no SB, logo o offset é 0, como não há padrão então Length = 0, como não existe símbolo no SB, então pomos a letra “a”
   Avançamos o SB e LAB um caracter:

3. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos: “l” está no SB? Ora, o SB contém agora a sequencia “sa”, então “l” não está no SB, então temos de o codificar da seguinte forma:
   <0,0,”l”> = Não há nenhum padrão no SB, logo o offset é 0, como não há padrão então Length = 0, como não existe símbolo no SB, então pomos a letra “l”

   Avançamos o SB e LAB um caracter:

4. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos: “s” está no SB? Ora, “s” está no SB. Relembrando que o objetivo é sempre encontrar a maior sequencia possível, por isso vamos continuar a fazer a pergunta: já vimos que “s” está no SB, e “a” está no SB? Sim!, e “∅” ? Ora, “∅” já não está no SB, por isso a maior sequencia que encontramos foi “sa”. Então codificamos agora da seguinte forma:

<3,2,∅> = “sa” no LAB está à distância 3 de “sa” no SB. “sa” tem comprimento 2, logo Length = 2. Como a sequência existe, então pomos o código do próximo símbolo, ou seja, ∅. ∅ acabou de ser codificado, por isso avançamos o SB e LAB 2 caracteres:

5. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “s” está no SB? Sim!, “a” está? Sim!, “l” está? Sim!, “s” está? Sim!, “a” está? Sim!, “∅” está? Sim! **NOTA** “s” está? Sim! “a” está? Sim!

   NOTA: Podemos passar depois do Search Buffer, mas não podemos passar do LAB. no fundo o que nós fizemos foi estender ainda mais o SB, pois queremos a maior sequência possível

   Com isto a maior sequência que encontramos foi: “salsa ∅sa”. Então codificamos agora da seguinte forma:

   <6,8,”l”> = “salsa ∅sa” está no SB à distância 6. “salsa ∅sa” tem comprimento 8, logo Length = 8. Como a sequência existe, então pomos o código do próximo símbolo, ou seja, “l”

   Avançamos 8 caracteres no SB e LAB, não esquecendo que #SB = 8 e #LAB = 8:

6. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “s” está no SB? Sim!, “a” está? Sim!, “∅” está? Sim!, “s” está? Sim!, “a” está? Sim!, “*l*” está? Sim! **NOTA** “s” está? Sim! “a” está? Sim!

   NOTA: Podemos passar depois do Search Buffer, mas não podemos passar do LAB. no fundo o que nós fizemos foi estender ainda mais o SB, pois queremos a maior sequência possível

Com isto a maior sequencia que encontramos foi: “sa∅salsa”. Então codificamos agora da seguinte forma:

<6,8,∅> = “sa∅salsa” está no SB à distância 6. “sa∅salsa” tem comprimento 8, logo Length é 8. Como a sequência existe, então pomos o código do próximo símbolo do LAB, ou seja, ∅ .

Avançamos 8 caracteres no LAB e SB:

7. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “z” está no SB? Não, então codificamos da seguinte forma:

   <0,0,”z”> = Não há nenhum padrão no SB, logo o offset é 0, como não há padrão então Length = 0, como não existe símbolo no SB, então pomos a letra “z”

   Avançamos um caracter no LAB e SB:

8. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “o” está no SB? Não, então codificamos da seguinte forma:

   <0,0”o”> = Não há nenhum padrão no SB, logo o offset é 0, como não há padrão então Length = 0, como não existe símbolo no SB, então pomos a letra “o”

   Avançamos um caracter no LAB e SB:

9. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “o” está no SB? Sim! então codificamos da seguinte forma:

   <1,1,”∅”> = “o” está no SB à distância 1, “o” tem comprimento 1, logo Length é 1, como a sequência existe, então pomos o código do próximo símbolo do LAB, ou seja, ∅.

   Avançamos 1 no LAB e SB:

10. salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo

   Vamos ao LAB e perguntamos “z” está no SB? Sim! “o” está? Sim! “o” está? Sim! “∅” está? Sim! **NOTA** “z” está? Sim! “o” está? Sim! “o” está? Sim!

   NOTA: Podemos passar depois do Search Buffer, mas não podemos passar do LAB. no fundo o que nós fizemos foi estender ainda mais o SB, pois queremos a maior sequência possível

   Então codificamos da seguinte forma:

   <4,7,EOF> = Distância de 4, length 7 e o próximo caracter é o fim do ficheiro, ou seja EOF

Assim, a nossa codificação em LZ77 fica:

<0,0,”s”>
<0,0”a”>
<0,0,”l”>
<3,2,∅>
<6,8,”l”>
<6,8,∅>
<0,0,”z”>
<0,0,”o”>
<1,1,∅>
<4,7,EoF>

Para terminar, o LZ77 **pressupõe que os padrões ocorrem próximos uns dos outros** (razão pela qual temos 2 buffers).

Mas se os padrões estiverem longe uns dos outros, este método torna-se **ineficiente**. A solução passa por utilizar Dicionários explícitos: LZ78

LZ77 (Dicionário Implícito): Usa uma janela de pesquisa (Buffer) que armazena o pedaço recente da sequência. As referências aos padrões são baseadas na posição relativa.

LZ78 (Dicionário Explícito): Cria um dicionário permanente de padrões, em que cada entrada é identificada por um índice, e é atualizado cada vez que um novo padrão é encontrado.

1. 2. LZ78

- Utilização de um dicionário explícito
- Dicionário construído de forma adaptativa no codificador e descodificador
- Símbolos são codificados da seguinte maneira: <i,c>, em que i corresponde ao índice no dicionário (em particular se i = 0, quer dizer que não foi encontrada nenhuma entrada e c corresponde ao novo caracter) e c corresponde ao código do próximo caracter. Estes tuplos passam a ser novas entradas no dicionário.

Exemplo: **Codificação**

- Usemos de novo a mensagem = “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”

1. Começamos por ter uma tabela vazia. Começamos a ler a string. “s” está nas entradas da tabela? Não, então adicionamos à tabela, indicando o índice, qual a entrada e codificação:

Índice	Entrada	Codificação
1	s	<0,s>

Relembrando que quando não está na tabela a codificação fica <0,símbolo>

2. Avançamos na string “s**a**lsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “a” está em alguma das entradas da tabela? Não, então adicionar de forma semelhante:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>

3. Avançamos na string “sa**l**sa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “l” está em alguma das entradas da tabela? Não, então adicionar de forma semelhante:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,l>

4. Avançamos na string “sal**s**a∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “s” está em alguma das entradas da tabela? Sim, então vamos continuar a ler na string (Relembrando que aqui o objetivo é também encontrar a maior sequência possível): “sal**sa**∅salsa∅salsa∅salsa∅zoo ∅zoo∅zoo”, “sa” está na tabela? Não, então acrescentar:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,L>
4	sa	<1,a>

Relembrando que como “sa” possui “s” que já está na tabela, usamos o índice de “s” na codificação de “sa”. Então a codificação de “sa” fica <1,a>, pois utiliza o indice de “s” e o caracter a seguir é “a”.

5. Avançamos na string “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. ∅ não está na tabela:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,L>
4	sa	<1,a>
5	∅	<0,∅>

6. Avançamos na string “salsa∅**s**alsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “s” já está na tabela. “salsa∅**sa**lsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “sa” já está na tabela. “salsa∅**sal**sa∅salsa∅salsa∅zoo∅zoo∅zoo”. “sal” não está na tabela, temos de adicionar, e na codificação vamos utilizar o índice de “sa”:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,l>
4	sa	<1,a>
5	∅	<0,∅>
6	sal	<4,l>

7. Avançamos na string “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “s” já está na tabela. “salsa∅sal**sa**∅salsa∅salsa∅zoo∅zoo∅zoo”. “sa” também já está. “salsa∅sal**sa∅**salsa∅salsa∅zoo∅zoo∅zoo”. “sa∅” não está, então adicionar à tabela, aproveitando o índice de “sa”:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,l>
4	sa	<1,a>

5	∅	<0,∅>
6	sal	<4,l>
7	sa∅	<4,∅>

8. Avançamos na string “salsa∅salsa∅**s**alsa∅salsa∅zoo∅zoo∅zoo”. “s” já está na tabela. “salsa∅salsa∅**sa**lsa∅salsa∅zoo∅zoo∅zoo”. “sa” já está na tabela. “salsa∅salsa∅**sal**sa∅ salsa∅zoo∅zoo∅zoo”. “sal” já está na tabela. “salsa∅salsa∅**sals**a∅salsa∅zoo∅zoo∅ zoo”. “sals” não está na tabela, então vamos adicionar, e ao codificar vamos aproveitar o indíce de “sal”:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,l>
4	sa	<1,a>
5	∅	<0,∅>
6	sal	<4,l>
7	sa∅	<4,∅>
8	sals	<6,s>

9. Repetir este processo até ao fim da mensagem. Faz agora tu e como confirmação tens aqui a solução final:

Índice	Entrada	Codificação
1	s	<0,s>
2	a	<0,a>
3	l	<0,l>
4	sa	<1,a>
5	∅	<0,∅>
6	sal	<4,l>
7	sa∅	<4,∅>

8	sals	<6,s>
9	a∅	<2,∅>
10	salsa	<8,a>
11	∅z	<5,z>
12	o	<0,o>
13	o∅	<12,∅>
14	z	<0,z>
15	oo	<12,o>
16	∅zo	<11,o>
17	<o,EoF>	<12,EoF>

Exemplo: **Descodificação**

- Utilizando a coluna “Codificação” obtida acima.
- Começando com a string mensagem = “ ”
- O descodificador vai construir o mesmo dicionário do codificador
- Algoritmo:
    - Pegar na coluna “Codificação”
    - Se for do tipo <0,caracter> acrescentar à string vazia e colocar na coluna da entrada
    - Se for do tipo <i,caracter> ir ao índice e buscar a entrada e ainda adicionar o caracter. Colocar na nova string e numa nova entrada

Vantagem do LZ78: Não estamos limitados ao tamanho do SB no LZ77
Desvantagem do LZ78: O dicionário pode ficar muito grande e pode ser preciso dar reset ao dicionário .

1. 3. LZW

• Variante do LZ78 em que se evita o uso do duplo <i,caracter>. Aqui só são enviados os índices do dicionário, <i>
• O dicionário, no início, não começa vazio. Começa com todos os símbolos do alfabeto.

Exemplo: **Codificação**

Consideremos mais uma vez a mensagem = “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”

1. Como vimos, o nosso dicionário começa com todos os símbolos do alfabeto nas entradas, neste caso, por ordem alfabética (não tem de ser):

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z

2. Começamos a ler a string: "**s**alsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "s" existe no dicionário? Sim!, então vamos continuar a ler. "**sa**lsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "sa" existe no dicionário? Não, então vamos adicionar "sa" ao dicionário e vamos codificar o "s" , usando o seu índice (5):

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa

Codificação: 5

3. Continuamos a ler a string: "s**a**lsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "a" existe no dicionário? Sim!, então continuamos a ler: "s**al**sa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "al" existe no dicionário? Não. Então vamos adicionar "al" ao dicionário, com um novo índice e vamos codificar o "a" usando o índice do dicionário:

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa
8	al

Codificação: 5 2

4. Continuamos a ler a string: "sa**l**sa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "l" existe no dicionário? Sim!. "sa**ls**a∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "ls" existe no dicionário? Não! Então adicionamos "ls" ao dicionário e codificamos "l" com o índice no dicionário:

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa
8	al
9	ls

Codificação: 5 2 3

5. Continuamos a ler a string: "sal**s**a∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "s" existe no dicionário? Sim!. "sal**sa**∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "sa" existe no dicionário? Sim!. "sal**sa∅**salsa∅salsa∅salsa∅zoo∅zoo∅zoo". "sa∅" existe no dicionário? Não! Então adicionamos "sa∅" ao dicionário com um novo índice e codificamos "sa" com o seu indíce no dicionário:

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa
8	al
9	ls
10	sa∅

Codificação: 5 2 3 7

6. Continuamos a ler a string: “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “∅” existe no dicionário? Sim! “salsa∅salsa∅salsa∅salsa∅zoo∅zoo∅zoo”. “∅s” existe no dicionário? Não! Então adicionamos “∅s” ao dicionário com um novo índice e codificamos “∅” com o seu índice do dicionário.

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa
8	al
9	ls
10	sa∅
11	∅s

Codificação: 5 2 3 7 1

7. Repetir este processo até ao fim da mensagem. Faz agora tu e como confirmação tens aqui a solução final:

Índice	Entrada
1	∅
2	a
3	l
4	o
5	s
6	z
7	sa
8	al
9	ls
10	sa∅
11	∅s
12	sal
13	lsa
14	a∅
15	∅sa
16	als
17	sa∅s
18	sals
19	sa∅z
20	zo
21	oo
22	o∅
23	∅z
24	zoo
25	o∅z

Codificação: 5 2 3 7 1 7 9 2 11 8 10 12 10 6 4 4 1 20 22 24

2. Códigos Aritméticos *(slides 140-176, pp 1)*

- **Motivação para a utilização de códigos aritméticos (Vamos ver mais à frente o que são):**

Dado o alfabeto A = {a,b,c} e P(x=a) = 0.95, P(x=b) = 0.02 e P(x=c) = 0.03.
Podemos perceber que as probabilidades são bastantes desequilibradas.
Ao calcularmos a entropia temos que H(x) ≈ 0.335 bits/símbolo
Ao fazermos a codificação de Huffman temos que:

símbolo	length
a	0
b	10
c	11

$\overline{L} = 1 \cdot 0.95 + 2 \cdot 0.02 + 2 \cdot 0.03 = 1.05$ (nºbits * P(X=a))

H(x) ≤ $\overline{L}$ ≤ H(x) + 1. Tal ocorre, porém um ótimo código quer-se quando $\overline{L}$ está o mais próximo possível de H(x). Por isso, neste caso a codificação de Huffman é ineficiente. **(Quando as probabilidades são desequilibradas, o majorante (H(x)+1) é muito alto)**.

E se fizéssemos um agrupamento de símbolos? Neste caso, se fosse dada uma cadeia, o alfabeto poderia crescer de forma exponencial e a árvore de Huffman iria ficar muito profunda. Até poderíamos ter uma maior eficiência, mas o custo computacional iria ser enorme.

Então neste caso, vamos utilizar Códigos Aritméticos.

- **Definição Códigos Aritméticos:** Codificação de uma sequência particular de comprimento m sem ter que gerar todas as sequências possíveis (ou seja, já não temos o problema do agrupamento de símbolos em que se gerava um alfabeto enorme)

Nos códigos aritméticos, a nossa sequência de símbolos vai ficar numa TAG que representa um valor de [0,1[. Além disso, precisamos de uma função que mapeia uma sequência de símbolos: Função de distribuição cumulativa.

Exemplo da função cumulativa:

Dadas as probabilidades P(x=a) = 0.95, P(x=b) = 0.02 e P(x=c) = 0.03, a função cumulativa de a,b e c é: F(a) = 0.95, F(b) = 0.95 + 0.02 = 0.97 e F(c) = 0.95+0.02+0.03 = 1

- **Algoritmo para Códigos Aritméticos:**

$l^o = 0$ $(limite\ inferior)$
$u^o = 1$ $(limite\ superior)$
$l^n = l^{n-1} + amp^{n-1}.\ F(a_{k-1})$
$u^n = l^{n-1} + amp^{n-1}.\ F(a_k)$
Obs: O n não é necessariamente igual a k

TAG = $\frac{u^n + l^n}{2}$

Exemplo:

Codificação

Dada a sequência Seq = a_2, a_3, a_2, a_1 e P(x=a_1) = 0.7,P(x=a_2) = 0.1 e P(x=a_3) = 0.2, vamos fazer a codificação de Códigos aritméticos:

0) Escrevemos o limite superior e inferior, tal como o algoritmo nos diz, assim como os valores da nossa função de acumulação:

$l^o = 0$ $(limite\ inferior)$
$u^o = 1$ $(limite\ superior)$
F(a_0) = 0, F(a_1) = 0.7, F(a_2) = 0.8, F(a_3) = 1

1) Recebemos da sequência o a_2:

$l^1 = l^0 + amp^0 F(a_1) = 0 + (u^0 - l^0) . F(a_1) = 0 + 1 . 0.7 = 0.7$

$u^1 = l^0 + amp^0 F(a_2) = 0 + 1 . 0.8 = 0.8$

2) Recebemos da sequência o a_3:

$l^2 = l^1 + amp^1 F(a_2) = 0.7 + (u^1 - l^1) . F(a_2) = 0.7 + 0.1 . 0.8 = 0.78$

$u^2 = l^1 + amp^1 F(a_3) = 0.7 + 0.1 . 1 = 0.8$

3) Recebemos da sequência o a_2:

$l^3 = l^2 + amp^2 F(a_1) = 0.78 + (u^2 - l^2) . F(a_1) = 0.78 + 0.02 . 0.7 = 0.794$

$u^3 = l^2 + amp^2 F(a_2) = 0.78 + 0.02 . 0.8 = 0.796$

4) Recebemos da sequência o a_1:

$l^4 = l^3 + amp^3 F(a_0) = 0.794 + (u^3 - l^3) . F(a_0) = 0.794 + 0.002 . 0 = 0.794$

$u^4 = l^3 + amp^3 F(a_1)$ = 0.794 + 0.002 . 0.7 = 0.7954

Então a nossa TAG = $\frac{u^4 + l^4}{2} = \frac{0.7954+0.794}{2} = 0.7947$

Porém, a nossa TAG está em decimal, temos de a passar para binário. Mas quantos bits vai ter a nossa tag? Ora, pela fórmula $\left[- log_2(u^n - l^n)\right] + 1$. Fazendo a conta, neste caso são 11 bits. Convertendo a TAG decimal para TAG binária temos: TAG = 11001011011

Descodificação

Por exemplo, recebendo agora a TAG binária 11001011011, transformamos para uma TAG decimal: TAG = 0.7944335938 ∈ $[0.794, 0.7954]$

Relembrando o que conhecíamos a priori:

$l^o = 0\ (limite\ inferior)$

$u^o = 1\ (limite\ superior)$

F(a_0) = 0, F(a_1) = 0.7, F(a_2) = 0.8, F(a_3) = 1

Então agora para descodificar vamos aplicar a fórmula $t_n = \frac{TAG - l^{n-1}}{amp^{n-1}}$ a cada iteração:

1) $t_1 = \frac{0.7944335938 - l^0}{amp^0}$ = $0.7944335938 \in [F(a_1), F(a_2)]$, por isso vamos descodificar a_2

   Mensagem descodificada M = a_2

2) $t_2 = \frac{0.7944335938 - l^1}{amp^1}$ = 0.944335938 ∈ $[F(a_2), F(a_3)]$,por isso vamos descodificar a_3

   Mensagem descodificada M = a_2,a_3

3) $t_3 = \frac{0.7944335938 - l^2}{amp^2}$ = 0.72167969 ∈ $[F(a_1), F(a_2)]$, por isso vamos descodificar a_2

   Mensagem descodificada M = a_2,a_3, a_2

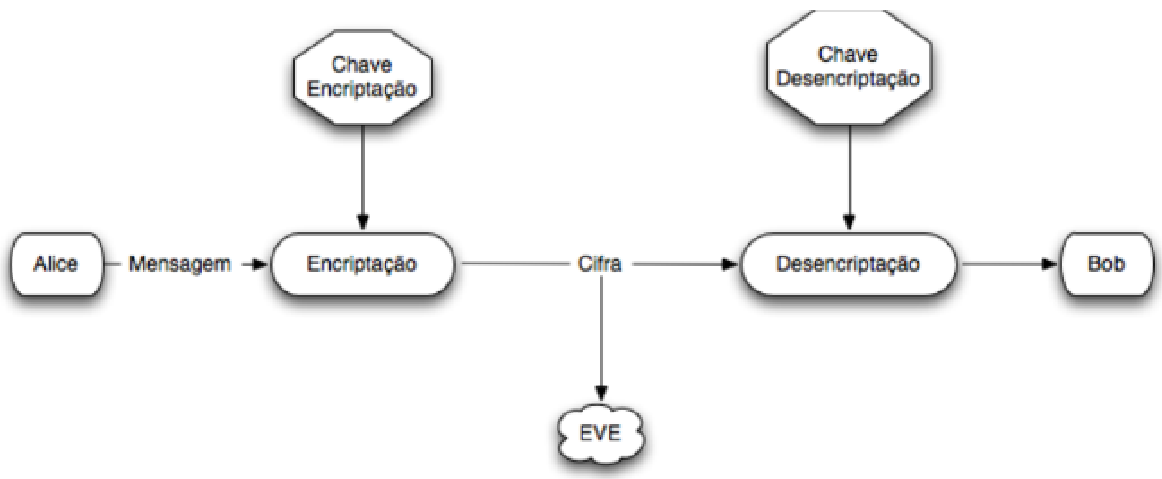
4) $t_4 = \frac{0.7944335938 - l^3}{amp^3}$ = 0.2167969 ∈ $[F(a_0), F(a_1)]$ por isso vamos descodificar a_1

   Mensagem descodificada M = a_2,a_3, a_2,a_1

3. Encriptação e segurança *(pp 2)*

Objetivos da Criptografia:

- **Autenticação**: Saber a origem/destino dos dados/emissor
- **Não-Repúdio:** Mecanismo de garantia de segurança que impede uma entidade participante numa dada operação de negar essa participação (Ex: **e-mails**: Um remetente pode usar uma assinatura digital para garantir que ele realmente enviou o e-mail, impedindo-o de negar isso mais tarde.)
- **Confidencialidade:** manter os dados secretos realizando encriptação de dados, garantir que os dados só podem ser desencriptados pela pessoa autorizada
- **Integridade:** Processo de garantia que a mensagem não foi alterada, utilizando, por exemplo, funções de hash. Cada mensagem possui uma função de hash associada e caso a mensagem for alterada, por mais mínima que seja a alteração, vai gerar uma função de hash completamente diferente da original.



Neste caso, a Alice possui uma mensagem que quer mandar ao Bob. Para isso, a Alice vai usar uma chave de encriptação para encriptar a sua mensagem. Ao encriptar, vai transformar a mensagem num formato de cifra. Como está em formato cifra, quem não possua uma chave de encriptação, não consegue entender a qual a mensagem que está a ser enviada.

No caso, o Bob possui a chave de encriptação, então consegue desencriptar a mensagem e ver o significado da mesma enviada pela Alice.

Como é visível, há ainda a introdução de uma terceira pessoa, a Eve, que vai tentar interceptar a mensagem a meio. A Eve pode ter estes objetivos: Pode querer **ler a mensagem enviada pela Alice**, pode estar a **tentar encontrar a chave que permite desencriptar todas as mensagens**, pode estar a **querer alterar a mensagem da Alice** ou pode estar a **querer fazer-se passar pela Alice.**

- **Tipos de Ataques**

    - **Cipher Text Only (Cifra)**: Eve tem apenas acesso a uma cópia da Cifra, sem conhecer a mensagem original ou a chave. **Perigos**: Eve pode fazer uma análise estatística, ou seja, por exemplo, sabendo que a letra mais comum do alfabeto inglês é a letra "e", a letra que aparecer com mais frequência na cifra, pode ser a letra "e" na mensagem original.
    - **Known plaintext (Mensagem conhecida):** Se Eve tiver acesso a pares de Mensagem e Cifra, facilmente consegue deduzir a chave de encriptação/desencriptação
    - **Chosen plaintext (Seleção de mensagem)**: Se Eve tiver acesso temporário ao Encriptador (não tem à chave), pode criar Mensagens planeadas para revelarem padrões na Cifra. Ao enviar várias mensagens planeadas, tem vários pares Mensagem, Cifra e pode realizar uma análise cuidadosa e revelar a chave de encriptação.
    - **Chosen cifer (Seleção da Cifra)**: Eve pode escolher uma Cifra e obter a mensagem original desencriptada sem conhecer a chave, ou seja, tem acesso ao desencriptador. Tal como o Chosen plaintext consegue enviar cifras planeadas, de modo a ter pares Mensagem,Cifra para depois fazer uma análise e revelar a chave de desencriptação.

RSA - Algoritmo de encriptação

Antes de avançarmos para o RSA, é preciso termos algumas noções de Teoria dos Números:

- $a \mid b$ => a divide b, b é múltiplo de a, b mod a = 0
- p é primo => divisível por p e 1
- $\pi(x)$ => Corresponde ao nº de primos < x, $\pi(x) = \frac{x}{ln(x)}$
- $mdc(a, b)$ pode ser feito pela **decomposição em fatores primos** ou pelo **Algoritmo de Euclides**.

Exemplo do **Algoritmo de Euclides**:

mdc(1180,482) = último resto diferente de 0

(Divisor para a ser dividido, Resto passa a dividir)
1180/482 = 216(resto), 2 (quociente)
482/216 = 50(resto), 2 (quociente)
216/50 = 16(resto), 4(quociente)
50/16 = **2 (resto)**, 3(quociente)
16/2 = 0(resto), 8(quociente)

2 corresponde ao último resto diferente de 0, logo mdc(1180,482) = 2

Algoritmo de Euclides Estendido

Útil para resolver equações do tipo $ax + by = d$
Dados a,b inteiros, calcular x,y inteiros

Regras do algoritmo:

$x_0 = 0$
$x_1 = 1$
$y_0 = 1$
$y_1 = 0$
$x_k = - q_{k-1} . x_{k-1} + x_{k-2}$
$y_k = - q_{k-1} . y_{k-1} + y_{k-2}$

Exemplo:
$482x + 1180y = 2$

Importante: (Pelo algoritmo de Euclides calculado à pouco vemos que temos 5 quocientes, logo a nossa solução para a equação vai estar no x_5 e y_5)

Comecemos por calcular x_2 até chegarmos à solução que é x_5 (lembrar que $x_0 = 0$ e $x_1 = 1$)
$x_2 = - q_1 . x_1 + x_0 = - 2 . 1 + 0 = - 2$
$x_3 = - q_2 . x_2 + x_1 = - 2 . - 2 + 1 = 5$
$x_4 = - q_3 . x_3 + x_2 = - 4 . 5 - 2 = - 22$
$x_5 = - q_4 . x_4 + x_3 = - 3 . - 22 + 5 = 71$

Logo $x = 71$.

Fazer o mesmo para o y (y_5 vai dar -29), logo $y = - 29$

Congruências

$a \equiv b \bmod n$ significa que:

$$\boxed{\begin{array}{c} a \bmod n = b \bmod n \\ a = b + nk \end{array}}$$

Exemplo de congruência:

$5x + 6 \equiv 13 \bmod 11$
$5x \equiv 7 \bmod 11$
$x \equiv \frac{7}{5} \bmod 11$ => mdc(5,11) = 1, logo podemos dividir

Como calcular isto?

1ºProcesso) 7 mod 11 ≡ 18 ≡ 29 ≡ 40 ≡ …
Pegamos no 40 porque sabemos que $\frac{40}{5}$ dá um número inteiro (8). Logo $x \equiv 8 \bmod 11$

Problema: Pode não ser assim tão fácil encontrar um número que dê inteiro na divisão
2ºprocesso) Inverso Multiplicativo

mdc(5,11) = 1, logo podemos dividir
$5x \equiv 1 \, mod \, 11 \; => \; 5x = 1 + 11k \; => \; 5x + 11k = 1$
Aplicar Algoritmo de Euclides Estendido:

11/5 = 1(resto), 2(quociente)
5/1 = 0(resto),5(quociente)

Temos 2 quocientes portanto a solução de x estará em x_2:

$x_0 = 0$

$x_1 = 1$

$y_0 = 1$

$y_1 = 0$

$x_2 = -2 . 1 + 0 = -2$

Logo o inverso multiplicativo de 5 = -2 mod 11 ≡ 9 mod 11

$x \equiv \frac{7}{5} mod \, 11 \equiv 7 * 9 \, mod \, 11 \equiv 63 \, mod \, 11 \equiv 8 \, mod \, 11$

Teorema do Resto Chinês (中國剩餘定理)

$x \equiv a \, mod \, m$
$x \equiv b \, mod \, n$
Então $x \equiv y \, mod \, (m * n)$

Exemplo:

$x \equiv 3 \, mod \, 7$
$x \equiv 5 \, mod \, 15$

Isto significa que $3 + 7k \equiv 5 \, mod \, 15 \; => \; 7k \equiv 2 \, mod \, 15 \; => \; k \equiv \frac{2}{7} mod \, 15$
mdc(7,15) = 1, podemos dividir

Descobrir o inverso multiplicativo:

$7z \equiv 1 \, mod \, 15 \equiv 1 + 15k \; => \; 7z + 15k = 1$
Aplicar algoritmo de euclides estendido

(...)

Z = -2 mod 15 ≡ 13 mod 15

$\frac{1}{7} \equiv 13 \; mod \; 15$ logo $k \equiv 2 * 13 \; mod \; 15 \equiv 11 \; mod \; 15$

Assim, $y = 3 + 7k = 3 + 7 * 11 = 80$
$x \equiv 80 \; mod \; 105$

Função de Euler

$\phi(n) = \#números \; inteiros \; 1 \leq a < n \; que \; são \; primos \; com \; n$

Esta função diz-nos duas coisas importantes:

- $n = p \, . \, q$
- $\phi(n) = (p - 1)(q - 1)$

Teorema de Euler

Este teorema diz-nos que se mdc(a,n) = 1, então:

- $a^{\phi(n)} \equiv 1 \; mod \; n$

Este teorema é útil para reduzirmos potências muito grandes.

Corolário - Pequeno Teorema de Fermat

Este corolário diz-nos que se mdc(a,n) = 1, então:

- $a^{n-1} \equiv 1 \; mod \; n$

Teste de Primalidade de Fermat

Se $a^{n-1} \equiv 1 \; mod \; n$, então n é provavelmente primo
Caso contrário, n é composto.

Teorema dos Números Primos

- $\pi(x)$ = nº de primos inferiores a x $\simeq \frac{x}{ln(x)}$

Para o RSA vamos precisar do:

- Teorema dos Números Primos
- Algoritmo de Euclides
- Algoritmo de Euclides Estendido
- Inverso Multiplicativo
- Função de Euler
- Teorema de Euler
- Teste de Primalidade de Fermat

RSA

1. Bob escolhe dois números primos secretos (p e q) e multiplica-os: n = p . q
2. Bob escolhe um expoente de encriptação e, que tem de ser menor que n, e que satisfaça a seguinte condição: $mdc(e,\ \phi(n))\ =\ 1$. Além disso, e não tem fatores em comum com p-1 e q-1 (do ϕ(N)).
3. Bob calcula o expoente de desencriptação d, com base na seguinte expressão:
   $d.e\ \equiv\ 1\ mod\ \phi(n)$ => $d.e\ mod\ \phi(n)\ \equiv\ 1$
4. Bob torna n e e públicos (chave pública do Bob) e mantém p,q e d secretos (chave privada do Bob)
5. Alice encripta a mensagem m com a chave pública e do Bob e envia-lhe a cifra c:
   $c\ =\ E(m)\ =\ m^e\ mod\ n$
6. Bob desencripta a mensagem calculando:
   $m\ =\ D(c)\ =\ c^d\ mod\ n$

Nota: Se se encriptar duas vezes a mensagem com a mesma chave de encriptação, obtém-se a mensagem original.