Introdução a Teoria da Informação
TI0056 2014.1
Codificação de Fonte

Prof. Walter C. Freitas Jr

walter@gtel.ufc.br

Grupo de Pesquisa em Telecomunicações Sem Fio (GTEL)
http://www.gtel.ufc.br/~walter
http://sites.google.com/site/walterjr/
http://sites.google.com/site/profwalterfreitas/

Curso de Graduação em Engenharia de Teleinformática (CGETI)

Codificação de Fonte

Introdução

Idéia: transformar as mensagens de uma fonte em um conjunto de símbolos de modo que seja "ocupado menos espaço" ou que a informação da fonte "demore menos tempo a ser transmitida". **Queremos fazê-lo sem perdas: a operação inversa de decodificação ou descompressão deve dar origem exatamente às mesmas mensagens originais**.

1. A caracterização de uma fonte discreta é fácil se os símbolos forem independentes e equiprováveis (a entropia é máxima).
2. Essa caracterização já é mais difícil se os símbolos não forem independentes (a probabilidade de um símbolo depende dos símbolos passados e a entropia é menor).
3. Shannon provou que, se dividirmos a mensagem em blocos de letras ou palavras (no caso de texto), podemos calcular a entropia de cada bloco (= símbolo) pela mesma fórmula usada com símbolos independentes e assim aproximar-nos da entropia da fonte, considerando os blocos muito compridos.

Codificação de Fonte

Introdução

Existe uma relação entre a entropia e o número médio de dígitos binários por símbolo necessários para codificar uma mensagem

1. A questão está em descobrir como codificar eficientemente os símbolos da fonte, tendo cada um uma certa probabilidade de ocorrer. No caso dos símbolos da fonte não serem equiprováveis o código ótimo tem em conta as diferentes probabilidades de ocorrência usando palavras com comprimentos variáveis.
2. Convém agrupar os símbolos da fonte antes de codificar ou comprimir.

Representação de símbolos por dígitos binários

Exemplos

1. 26 letras + SPACE + 10 dígitos (0,1, ..., 9) = 37 símbolos
   i. Precisaríamos de $\log_2 37 = 5,21$ bits/símbolo.
   ii. A codificação faz-se atribuindo um número binário com 6 dígitos ($2^5 < 37 < 2^6$) a cada símbolo (desperdiçando $2^6 - 37 = 27$ números binários - palavras código).
   iii. Como se vê, a informação média por símbolo é inferior ao comprimento da palavra de código indicando que este código corresponde a uma representação redundante do alfabeto considerado.
   iv. Consideremos agora um novo alfabeto (extensão de 2da ordem) em que cada símbolo estendido corresponde a um dos 37^2 pares de símbolos do alfabeto original.
   v. Número de símbolos - $37^2 = 1369$, $2^{10} = 1024 < 1369$, $\mathbf{2^{11} = 2048 > 1369}$.
   vi. Para codificar cada um dos símbolos da extensão são necessários 11 bits, enquanto que a respectiva entropia é agora $H(S^2) = \log_2 37^2 = 10.42$ bit/símbolo estendido. Isto corresponde de fato a usar, em média, 5.5 (11/2) bits por cada símbolo original, resultando numa estratégia de codificação mais eficiente.
   Conclusão: agrupando os símbolos a compressão ou codificação é mais eficiente.
2. Inglês (26 letras + espaço)
   i. Codificação letra a letra, consideradas equiprováveis: são precisos 4,76 dígitos binários/carácter
   ii. Codificação letra a letra tendo em conta as probabilidades relativas de ocorrência: são precisos 4,03 dígitos binários/carácter
   iii. Codificação palavra a palavra tendo em conta a freqüência relativa das palavras: São precisos 1,66 dígitos binários/carácter
   Shannon calculou a entropia do inglês entre 0,6 e 0,13 bits/carácter.

Eficiência da codificação de alfabetos estendidos

ordem da extensão	entropia	comprimento da palavra de código	comprimento médio por símbolo
1	5.21	6	6.00
2	10.42	11	5.50
3	15.63	16	5.33
4	20.84	21	5.25
5	26.05	27	5.40
6	31.26	32	5.33
7	36.47	37	5.29
8	41.68	42	5.25
9	46.89	47	5.22
10	52.09	53	5.30

- A medida que aumenta a ordem da extensão, vai diminuindo o número médio de bits necessários para codificar cada símbolo do alfabeto original;
- Esta diminuição não é uniforme, embora pareça convergir para a entropia do alfabeto original.

Redundância e Entropia

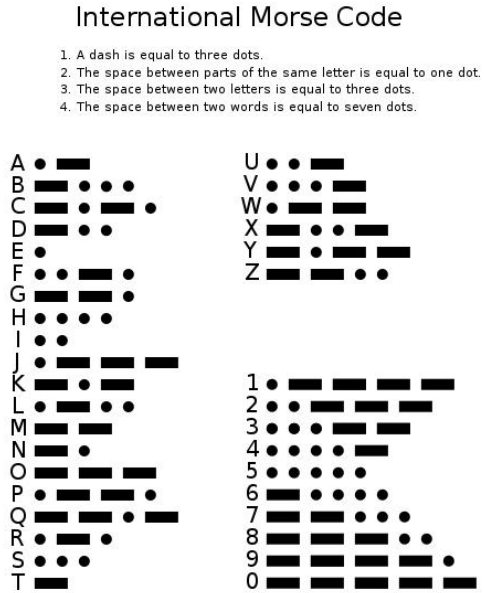
1. A influência entre símbolos reduz a entropia.
   Exemplo: linguagem, textos escritos: cada letra depende da(s) precedente(s) ou até das palavras anteriores, devido as regras impostas pela linguagem. A seguir à letra Q vem sempre U.
2. A influência mútua reduz a incerteza, logo, reduz a quantidade de informação. Trata-se de uma fonte redundante: significa "qe há símbolos produzidos pla font qe não são absolutament" essenciais para a transmissão da informação.
3. A redundância de uma seqüência de símbolos mede-se em função da redução de entropia ocorrida:

$$\boxed{\text{Redundância: } E = 1 - \frac{\mathcal{H}(Y|X)}{\mathcal{H}(X)}}$$

4. Em uma comunicação eficiente a redundância é indesejável (a mesma informação poderá ser transmitida usando menos símbolos).
5. A codificação para transmissão ótima pretende:
   i. reduzir a redundância "ineficiente" $\rightarrow$ códigos de fonte (compressão)
   ii. acrescentar redundância "eficiente" $\rightarrow$ códigos de canal

Considerações

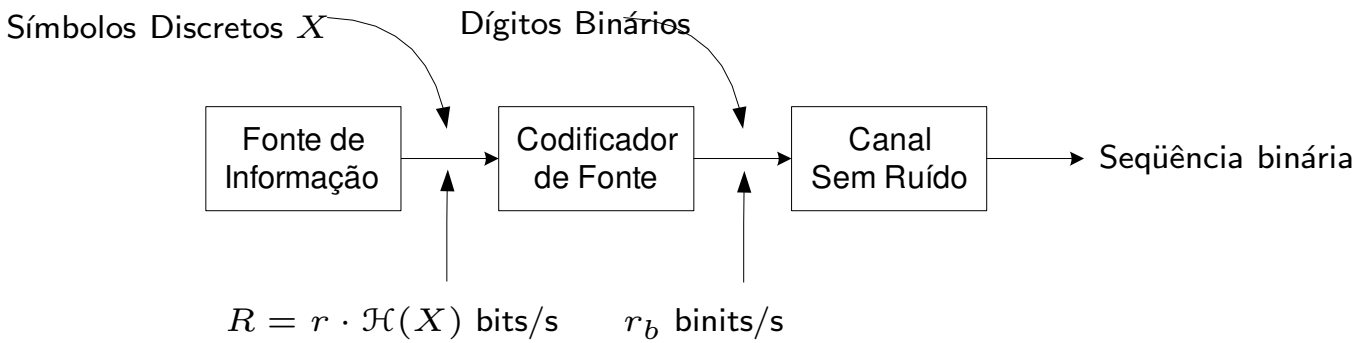
- Codificação de fonte é a representação dos dados da fonte de forma **eficiente**
- O dispositivo responsável por esta tarefa é chamado **codificador de fonte**
- Código Eficientes
  - Símbolos mais freqüentes com *palavras* menores
  - Símbolos menos freqüentes (raros) com palavras maiores
- **Códigos de tamanho variável**
- Exemplo: código Morse (língua inglesa)



  - Letra mais freqüente: E (‘.’)
  - Letra menos freqüente: Q (‘- - . -’)

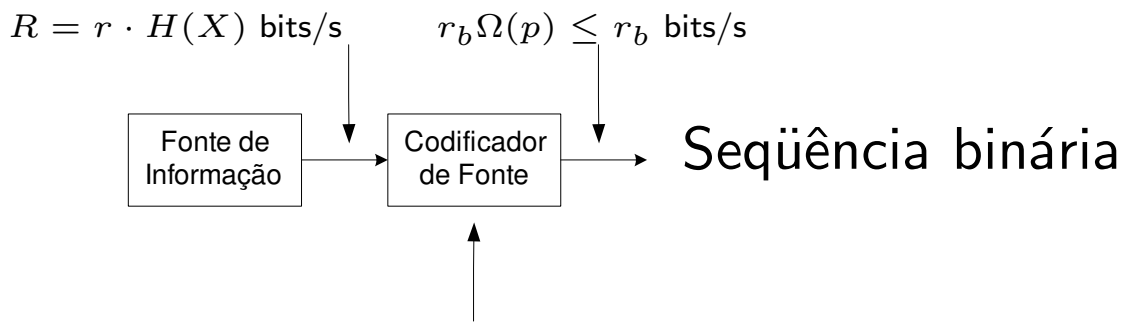
Codificador de fonte

- Palavras código são dadas na forma binária
- Código da fonte é *unicamente decodificável* tal que a seqüência de símbolos original pode ser reconstruída **perfeitamente** da seqüência binária codificada
- Subsistema:



Codificador de fonte

- Consideremos uma fonte discreta de entropia $\mathcal{H}(X)$ a gerar símbolos à cadência de r símbolos/s. O ritmo de informação é, como se sabe, $R = r\mathcal{H}(X)$ bits/s.
- E se quisermos codificar estes símbolos através de dígitos binários?
- Shannon mostrou que a informação proveniente de qualquer fonte discreta sem memória pode ser codificada como dígitos binários e transmitida através de um canal sem ruído à taxa binária de $r_b \geq R$ dígitos binários/s. Não é possível fazê-lo a uma taxa inferior ($r_b \leq R$).
- Esta é uma manifestação do famoso "teorema da codificação de fonte", de Shannon, demonstrado mais adiante.



Codificador de fonte - Exemplo

Uma fonte emite $r = 2000$ símbolos/s selecionados de um alfabeto de $M = 4$ elementos, com as probabilidades de ocorrência indicadas. Se quisermos codificar estes símbolos através de dígitos binários qual é o número mínimo de binits que podemos transmitir por unidade de tempo?

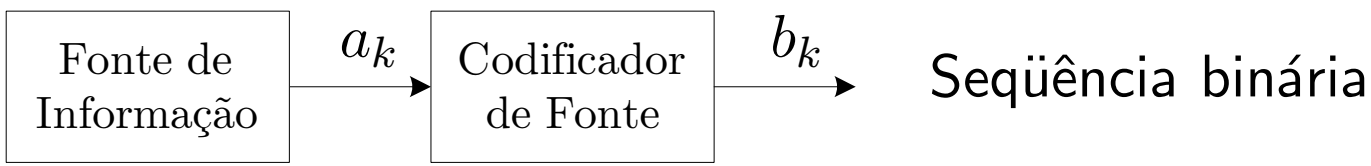
x_i	P_i	I_i
A	1/2	1
B	1/4	2
C	1/8	3
D	1/8	3

$$\mathcal{H}(X) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1,75 \text{bits/símbolo}$$
$$\Rightarrow R = 2000 \cdot 1,75 = 3500 \text{bits/s}$$

Uma codificação de fonte apropriada permitirá transmitir a informação da fonte a uma taxa binária rb $= 3500$ binits/s. Não é possível transmitir com menos dígitos binários por unidade de tempo. Repare-se nas dimensões que aparecem neste exemplo:
[r] - símbolos/s, [R] - bits/s, [rb] - binits/s

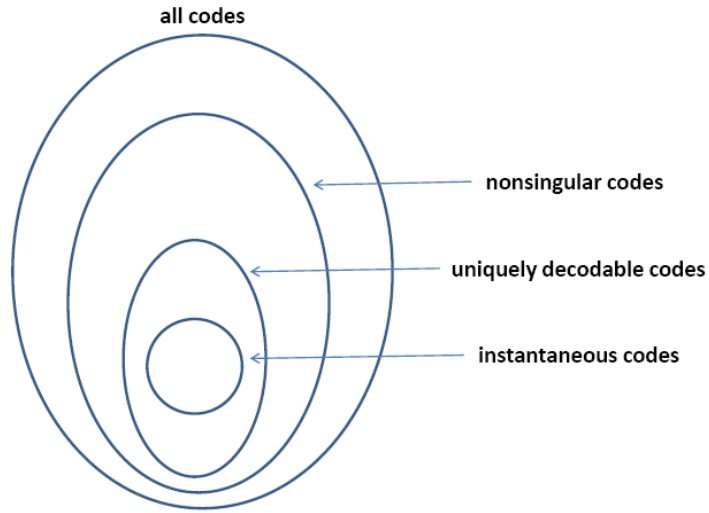
Teorema da Codificação de Fonte

- Seja uma fonte de K símbolos equiprováveis produzidos ao ritmo $r \rightarrow R = r \log_2 K$
- Fonte de símbolos com diferentes probabilidades $p_k \rightarrow R = r\mathcal{H}(X) < r \log_2 K$



Codificação de Fonte - classificação

- Singularidade - cada elemento do alfabeto da fonte é mapeado em uma palavra código diferente, i.e., $s_i \neq s_j \Rightarrow C(s_i) \neq C(s_j), i \neq j$
- Unicidade na decodificação - a extensão de um código mantém a característica da singularidade
- Prefixação - nenhuma palavra código se apresenta como prefixo de qualquer outra (também conhecido como códigos instantâneos)



Codificador de fonte - hipóteses (I)

- Seqüência da fonte de informação possui K diferentes símbolos
- Probabilidade de ocorrência do k-ésimo símbolo (s_k) é denominada p_k
- A palavra código (binária) associada ao símbolo s_k tem tamanho l_k
- Comprimento médio da palavra código: número médio de bits por símbolo da fonte usado na codificação

$$L = \sum_{k=0}^{K-1} p_k \cdot l_k \tag{1}$$

Codificador de fonte - hipóteses (II)

- Valor mínimo possível de L: L_{min}
- **Eficiência de codificação** do codificador de fonte

$$\eta = \frac{L_{\text{min}}}{L} \tag{2}$$

- Codificador é dito eficiente quando $\eta \to 1$
- **Como determinar o valor L_{min}?**

Primeiro Teorema de Shannon: Teorema da codificação de fonte

- Também chamado de *Teorema da codificação sem ruído* - trata da condição de codificação sem erros
- Responde a questão fundamental da codificação de fonte

Teorema:

Dada uma fonte de informação discreta com entropia $\mathcal{H}(\mathcal{S})$, o tamanho médio da palavra código L para qualquer codificação de fonte sem distorção é limitado por

$$L \geq \mathcal{H}(\mathcal{S}) \tag{3}$$

Codificação de fonte - limites

Entropia da fonte

Limite fundamental no número médio de bits para representar cada símbolo da fonte

Limite mínimo

$$L_{\text{min}} = \mathcal{H}(\mathcal{S}) \tag{4}$$

Eficiência de codificação

$$\eta = \frac{\mathcal{H}(\mathcal{S})}{L} \tag{5}$$

O que fazer para achar o código?

- Remoção da redundância de informação do sinal a ser transmitido
- Processo geralmente chamado de *compactação de dados* ou *compressão sem perdas*
- Como gerar códigos com comprimento médio próximo da entropia?

Codificação prefixo

- Restrição de codificação de fonte: ser unicamente decodificável
- Nenhuma seqüência de palavras código correspondente a uma dada seqüência da fonte pode ser associada a outra seqüência qualquer

Código prefixo

É um código no qual nenhuma palavra código é **prefixo** de qualquer outra palavra código

Seja $x_i = (x_{i,1}, ..., x_{i,n_i})$ a palavra de código de ordem i. Qualquer seqüência constituída por uma parte inicial de x_i (isto é, $(x_{i,1}, ..., x_{i,k}), k \leq n_i)$ chama-se um prefixo de x_i.
Para decodificar uma seqüência de palavras de código gerada por um código sem prefixos começa-se no princípio de cada palavra e decodifica-se uma a uma. Sabemos que chega ao fim de cada palavra porque essa palavra de código não é prefixo de qualquer outra (exemplo).

Codificação prefixo - exemplos

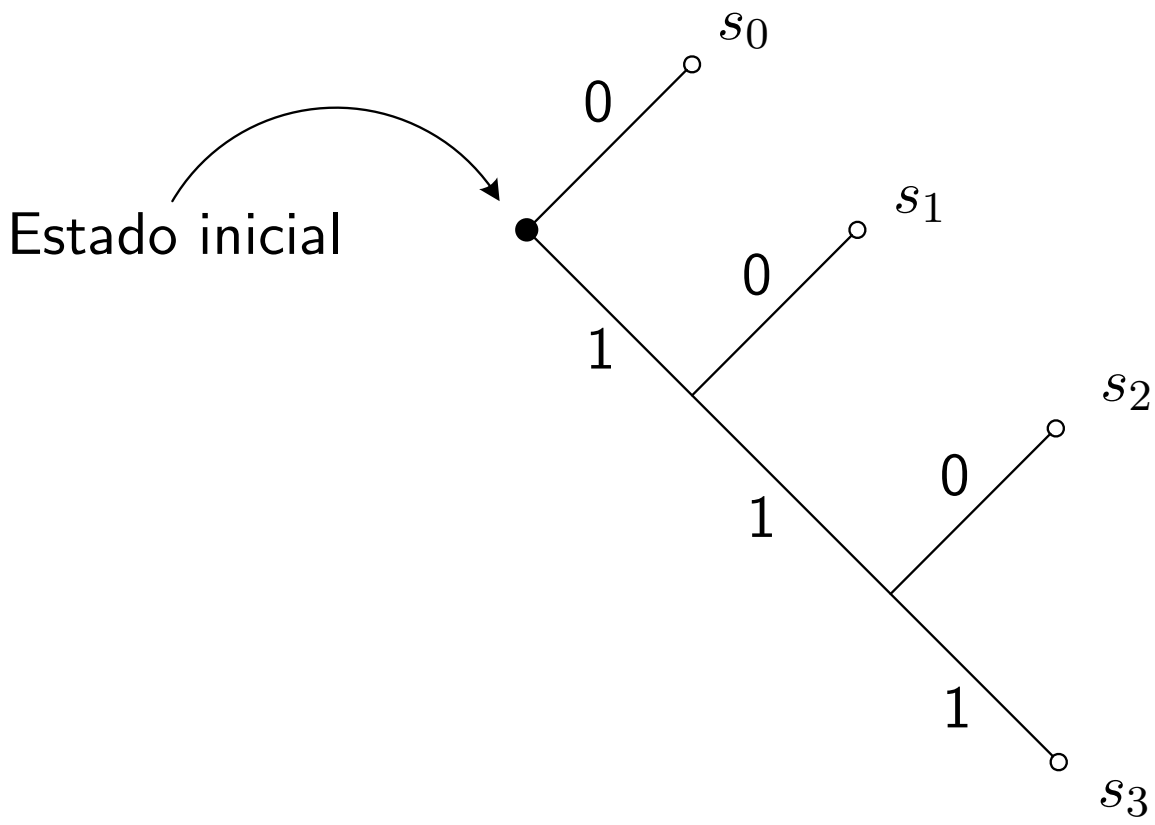
Símbolo da fonte	Probabilidade de ocorrência	**Código I**	**Código II**	**Código III**
s_0	0.5	0	0	0
s_1	0.25	1	10	01
s_2	0.125	00	110	011
s_3	0.125	11	111	0111

- **Código II é um código prefixo!**

Decodificação - código prefixo

- Busca numa *árvore de decisão*
- Facilidade de decodificação
- Cada bit que é recebido move o detector para algum ponto da árvore e um respectivo símbolo da fonte
- São **sempre** unicamente decodificáveis

Árvore de decisão - exemplo



- Decodificar 1011111000

Códigos prefixo

- Fonte discreta sem memória: $\{ s_0, s_1, \cdots, s_{K-1} \}$
- Probabilidades dos eventos: $\{ p_0, p_1, \cdots, p_{K-1} \}$
- Tamanho das palavras código: $\{ l_0, l_1, \cdots, l_{K-1} \}$

Inequação Kraft- McMillan

O tamanho das palavras códigos de um código prefixo devem satisfazer a seguinte inequação:

$$\sum_{k=0}^{K-1} 2^{-l_k} \leq 1 \tag{6}$$

Tal condição é necessária, mas não suficiente!

Dica para a prova: usar a representação em árvore.

Códigos prefixo - continuação

- Usando o inverso da definição da inequação de Kraft-McMillan pode-se dizer que

  > Se os tamanhos das palavras código de um determinado código respeitam a inequação de Kraft-McMillan, então um código prefixo com aquelas palavras código pode ser construído

- Embora códigos prefixos sejam unicamente decodificáveis, o inverso não é verdade
- Outros códigos unicamente decodificáveis não permitem conhecer sempre o fim de uma palavra código
- Códigos prefixo são também chamados de **códigos instantâneos**

Codificação binária de uma fonte discreta sem memória - Exemplo

Seja uma fonte discreta sem memória produzindo 4 símbolos A, B, C e D com probabilidades $1/2, 1/4, 1/8$ e $1/8$, respectivamente. Tem-se muitas possibilidades de codificação. Eis quatro possíveis:

x_i	P_i	Código I	Código II	Código III	Código IV
A	1/2	00	0	0	0
B	1/4	01	1	01	10
C	1/8	10	10	011	110
D	1/8	11	11	0111	111
	$\overline{N}$	2,0	1,25	1,875	1,75
	K	1,0	1,5	0,9375	1,0
		(comprimento fixo)		("comma code")	("tree code")

- $\mathcal{H}(X) = 1{,}75$ bits/símbolo (exemplo anterior)
- Código I: Eficiência $= \frac{\mathcal{H}(X)}{L}$=88%
- Código II: $\bar{N} = 1,25 < \mathcal{H}(X), K = 2^{-1} + 2^{-1} + 2^{-2} + 2^{-2} = 1,5 > 1 \Rightarrow$ não é unicamente decifrável (ex.:10011 pode ser BAABB ou CABB ou CAD, etc.)
- Código III: É unicamente decifrável (cada palavra começa por 0). Eficiência $= \frac{\mathcal{H}(X)}{L}$=93%
- Código IV: Nenhuma palavra de código aparece como o prefixo de outra palavra de código. É um código ótimo para esta fonte porque $\bar{N} = \mathcal{H}(X)$ e $K = 1$. Eficiência $= \frac{\mathcal{H}(X)}{L}$=100% (Exemplo: 110010111 $\leftrightarrow$ CABD)

Obs. Neste caso $\bar{N} = L$

Códigos prefixo - equacionamento

- Seja uma fonte discreta sem memória com entropia $\mathcal{H}(\mathcal{S})$, o tamanho médio da palavra código de um código prefixo é limitado por

$$\mathcal{H}(\mathcal{S}) \leq L < \mathcal{H}(\mathcal{S}) + 1 \tag{7}$$

- O lado esquerdo de (7) é satisfeito com a igualdade sob a condição de

$$p_k = 2^{-l_k} \tag{8}$$

- Logo, pode-se escrever

$$\sum_{k=0}^{K-1} 2^{-l_k} \leq \sum_{k=0}^{K-1} p_k = 1 \tag{9}$$

Códigos prefixo - equacionamento (cont.)

- Usando a inequação de Kraft-McMillan (construção do código), o tamanho médio da palavra código é

$$L = \sum_{k=0}^{K-1} \frac{l_k}{2^{l_k}} \tag{10}$$

- Entropia da fonte

$$\begin{aligned} \mathcal{H}(\mathcal{S}) &= -\sum_{k=0}^{K-1} \left(\frac{1}{2^{l_k}} \right) \log \left(2^{-l_k} \right) \\ &= \sum_{k=0}^{K-1} \frac{l_k}{2^{l_k}} \end{aligned} \tag{11}$$

- **Código prefixo apresenta a codificação mais eficiente neste caso**!

Códigos prefixo - equacionamento (cont.)

- Como encontrar o código prefixo para uma fonte arbitrária (probabilidades de ocorrência dos símbolos quaisquer valores)?
- Resposta: uso de **código estendido**.
- Seja L_n o tamanho médio da palavra código de um código prefixo estendido. Para um código unicamente decodificável L_n é o menor possível.
- Logo

$$\begin{aligned} \mathcal{H}(\mathcal{S}^n) \leq L_n < \mathcal{H}(\mathcal{S}^n) + 1 \\ n \cdot \mathcal{H}(\mathcal{S}) \leq L_n < n \cdot \mathcal{H}(\mathcal{S}) + 1 \\ \mathcal{H}(\mathcal{S}) \leq \frac{L_n}{n} < \mathcal{H}(\mathcal{S}) + \frac{1}{n} \end{aligned} \tag{12}$$

- No limite, os limites inferior e superior convergem

$$\lim_{n \to \infty} \frac{L_n}{n} = \mathcal{H}(\mathcal{S}) \tag{13}$$

Códigos prefixo - equacionamento (cont.)

- Então, fazendo a ordem n de um código prefixo estendido grande o suficiente, o código pode representar, fidedignamente, uma fonte $\mathcal{S}$ tão próximo quanto se deseje
- Ou seja, o tamanho médio de uma palavra código de um código estendido pode ser tão pequeno quanto a entropia da fonte, dado que o código estendido tem um tamanho *suficiente*
- There is no free lunch!!!
- Complexidade de decodificação: aumenta na ordem do tamanho do código estendido

Codificação de Fonte - Considerações

- Embora não possamos controlar a estatística da fonte de informação, a codificação de fonte deve obedecer ao seguinte (como nos códigos de Morse, Huffman, etc.):
  Aos símbolos mais frequentes devem corresponder palavras de código curtas; aos símbolos menos frequentes devem corresponder palavras compridas
- Entropia de fonte binária $= 1$ bit/símbolo se os símbolos forem equiprováveis
- Se os símbolos *não forem equiprováveis* a informação média por símbolo é inferior a um e a fonte apresenta redundância.
- A codificação da fonte reduz essa *redundância* por forma a aumentar a eficiência da comunicação
- A codificação faz-se agrupando a seqüência de dígitos binários da fonte em blocos de n símbolos, passando a formar um novo conjunto de 2^n símbolos
- A probabilidade de cada um dos 2^n novos símbolos é calculada e atribui-se a palavra de código mais curta ao novo símbolo mais provável, e assim por diante: palavras mais compridas correspondem a símbolos menos freqüentes
  Ao conjunto de 2^n novos símbolos chama-se extensão de ordem n da fonte binária

Codificação de Huffman

- Idéia básica: associar a cada símbolo, uma seqüência de bits, aproximadamente igual a quantidade de informação existente no símbolo
- Tamanho médio da palavra código aproxima o limite (entropia)
- Substituição do conjunto de estatísticas da fonte por um conjunto **mais simples**
- Processo de redução interativo até que o restem apenas dois símbolos para os quais '0' e '1' é um código ótimo
- Construção do código é feita através da retropropagação

Codificação de Huffman - algoritmo

1. Os símbolos da fonte são listados em ordem decrescente das probabilidades. Os dois símbolos com as menores probabilidades são assinalados como 0 e 1
2. Os dois símbolos anteriormente assinalados são *combinados* em um novo símbolo com a soma das probabilidades dos dois primeiros. A lista decrescente de probabilidades é novamente feita.
3. Procedimento se repete até sobrarem somente dois símbolos
4. Do final para o início da tabela, a palavra código de cada símbolo é encontrada

Codificação de Huffman - exemplo

- Seja a fonte discreta com as seguintes probabilidades para seus símbolos

Símbolo	Probabilidade
s_0	0.4
s_1	0.2
s_2	0.2
s_3	0.1
s_4	0.1

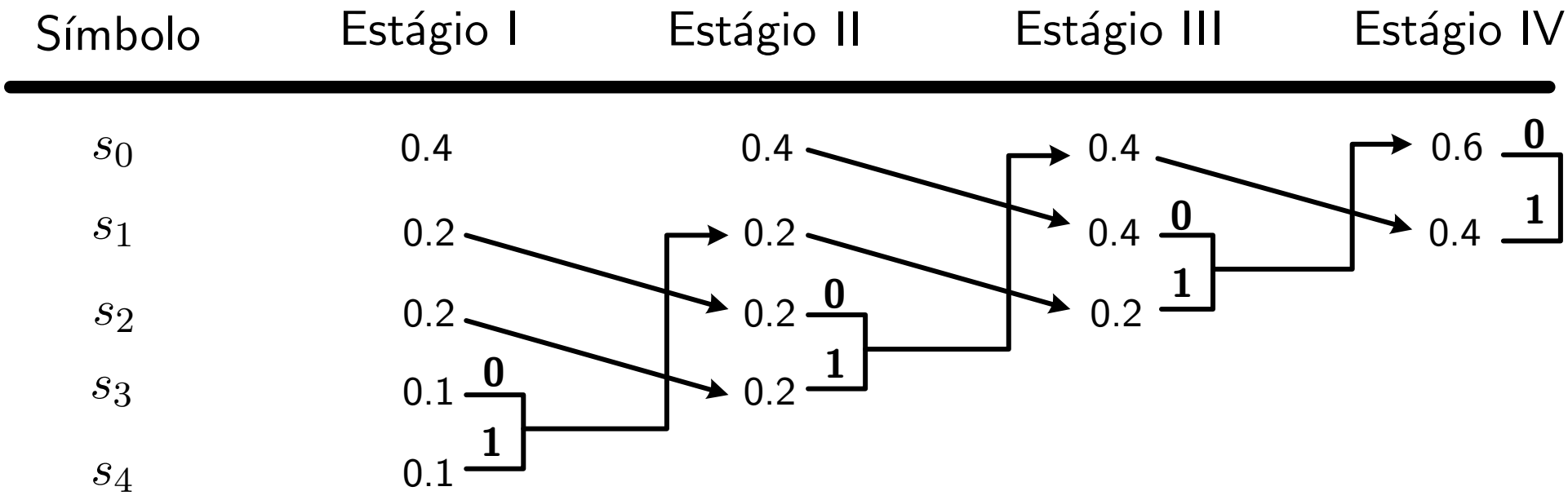
- Entropia da fonte

$$\mathcal{H}(\mathcal{S}) = 2.12193$$

Pergunta

O código de Huffman vai fornecer um tamanho médio da palavra código próximo da entropia da fonte?

Codificação de Huffman - exemplo



Codificação de Huffman - exemplo

Símbolo	Probabilidade	Palavra código
s_0	0.4	00
s_1	0.2	10
s_2	0.2	11
s_3	0.1	010
s_4	0.1	011

- Tamanho médio palavra código

$$L = 2.2$$

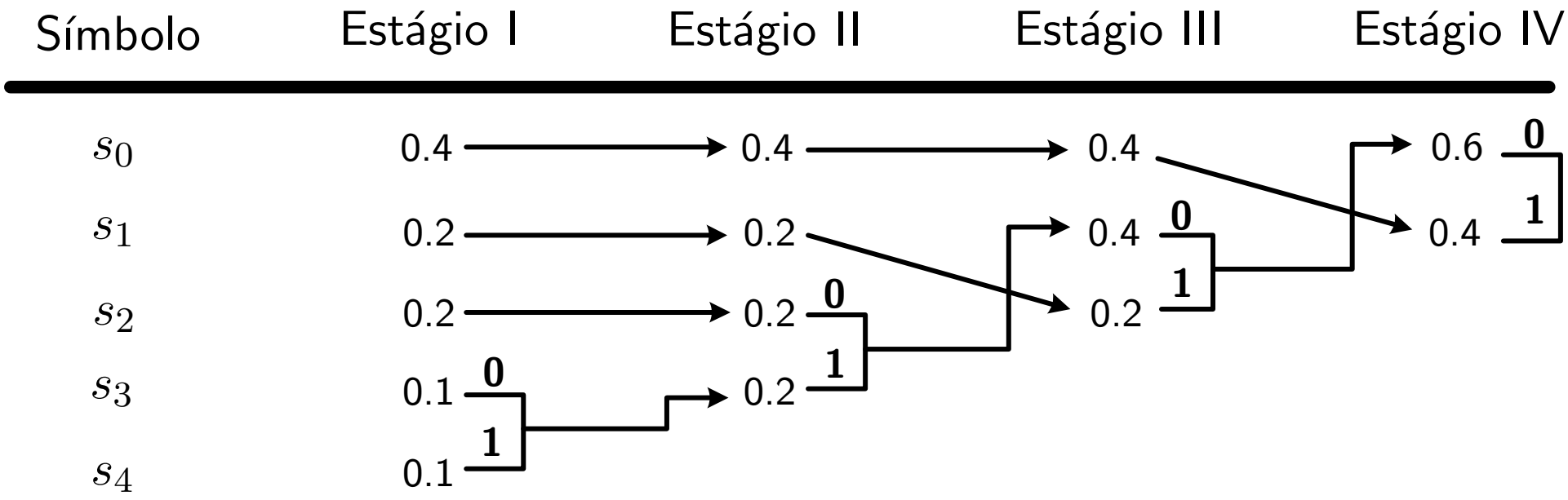
- L excede $\mathcal{H}(\mathcal{S})$ de apenas 3.67%
- L satisfaz $\mathcal{H}(\mathcal{S}) \leq L < \mathcal{H}(A) + 1$

Codificação de Huffman - comentários

- A codificação de Huffman não é única!
- Maneiras
  1. Arbitrariedade da associação de 0 e 1
  2. Valor da probabilidade igual: colocação dos valores
  3. Diferentes tamanhos médios da palavra código
- Medida da variabilidade (variância) do tamanho médio das palavras código

$$\sigma^2 = \sum_{k=0}^{K-1} p_k \left(l_k - L\right)^2 \tag{14}$$

Codificação de Huffman - não unicidade



Codificação de Huffman - não unicidade

Símbolo	Probabilidade	Palavra código
s_0	0.4	1
s_1	0.2	01
s_2	0.2	000
s_3	0.1	0010
s_4	0.1	0011

$$\sigma_1^2 = 0.16$$

$$\sigma_2^2 = 1.36$$

Codificação de Huffman - problemas

1. Requer conhecimento *a priori* do modelo probabilístico da fonte
2. Na prática, são poucas as situações que se conhece *a priori* as estatísticas da fonte
3. Além disso, em algumas aplicações, como modelos de texto escrito, a armazenagem impede o código de Huffman de capturar as relações entre palavras e frase - comprometendo a eficiência do código
4. **Alternativa??**

Codificação de Shannon-Fano

- Semelhante ao código de Huffman
- Utiliza listagem das probabilidades
- Algoritmo:
  - Dividir o conjunto de probabilidades em subconjuntos de igual probabilidade
  - Atribuir a cada subconjunto os dígitos 0 e 1
  - Seguir o procedimento sucessivamente para cada subconjunto até restar apenas um símbolo em cada subconjunto

x_i	$p(x_i)$	$-log2\ p(x_i)$	$C(x_i)$
a	1/2	1	0
b	1/4	2	10
c	1/8	3	110
d	1/8	3	111

Codificação de Lempel-Ziv

- Codificação adaptativa por natureza
- Implementação mais simples que a de Huffman

Idéia básica

Organização da seqüência de dados da fonte em segmentos que são subseqüências menores não encontradas anteriormente

Codificação de Lempel-Ziv - exemplo

- Seqüência de dados: **000101110010100101**
- Assume-se que os símbolos '0' e '1' já estão armazenados
- Logo, o início do algoritmo:
  Subseqüências armazenadas: 0, 1
  Dados para organização: 000101110010100101
- Menor seqüência **não** armazenada?
  Subseqüências armazenadas: 0, 1, 00
  Dados para organização: **0101110010100101**
- Outra seqüência?
  Subseqüências armazenadas: 0, 1, 00, 01
  Dados para organização: **01110010100101**
- Mais uma?
  Subseqüências armazenadas: 0, 1, 00, 01, 011
  Dados para organização: **10010100101**

Codificação de Lempel-Ziv - exemplo

Tabela de codificação

Pos.:	1	2	3	4	5	6	7	8	9
Subseq.:	**0**	**1**	**00**	**01**	**011**	**10**	**010**	**100**	**101**
Rep. num.:			11	12	42	21	41	61	62
Blocos cod.:			0010	0011	1001	0100	1000	1100	1101

Codificação de Lempel-Ziv - palavra código

1. Contagem do número de seqüências diferentes
2. Cálculo do número de bits n_1 para representar as seqüências
3. Número de bits da palavra código: $n_1 + 1$
4. Identificação da inovação e do prefixo
5. Identificação da **posição** do prefixo na tabela
6. Palavra código: **prefixo + inovação**

Codificação de Lempel-Ziv - características

- O último símbolo de cada subseqüência é chamado de **símbolo de inovação** - acrescenta à informação anterior e torna uma seqüência distinta
- O último bit de cada seqüência codificada representa o símbolo de inovação da subseqüência considerada
- Os bits restantes fornecem a representação equivalente do **ponteiro** para as *subseqüências originais*
- Decodificação: utiliza o ponteiro para identificar a subseqüência e adiciona o símbolo de inovação

Codificação de Lempel-Ziv - observações

- Código utiliza um número fixo de bits para representar um número variável de símbolos da fonte
- Na prática, 12 bits são utilizados, implicando num código de 4096 entradas
- Lempel-Ziv é o padrão de compactação de dados
- Para textos, Lempel-Ziv atinge uma compactação de aproximadamente 55% contra 43% de Huffman