Teoria da Informação (#4)

Classes de códigos, desigualdade de Kraft, código óptimo, Shannon, Shannon-Fano, Huffman, Huffman adaptativo

Miguel Barão

ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

O que é um código?

Classes de códigos

- Singular e não singular
- Univocamente descodificável
- Instantaneo

Códigos instantaneos

- Desigualdade de Kraft
- Códigos óptimos
- Código de Shannon
- Código de Shannon-Fano
- Código de Huffman
- Grupos de símbolos
- Penalização no comprimento médio

O que é um código?

Definition (Código)

Um código é uma função $C : \mathcal{X} \to \mathcal{D}^*$, em que

- $\mathcal{X}$ é um alfabeto dos símbolos que se pretende codificar
- $\mathcal{D}$ é o alfabeto dos símbolos usados no output (ex. binário)
- $\mathcal{D}^*$ é o conjunto das strings finitas de símbolos de $\mathcal{D}$

O que é um código?

Definition (Código)

Um código é uma função $C : \mathcal{X} \rightarrow \mathcal{D}^*$, em que

- $\mathcal{X}$ é um alfabeto dos símbolos que se pretende codificar
- $\mathcal{D}$ é o alfabeto dos símbolos usados no output (ex. binário)
- $\mathcal{D}^*$ é o conjunto das strings finitas de símbolos de $\mathcal{D}$

- $C(x)$ é a palavra de código correspondente a x
- $l(x)$ é o comprimento da palavra de código $C(x)$
- $L(C)$ é o comprimento médio do código C

O que é um código?

Definition (Código)

Um código é uma função $C : \mathcal{X} \to \mathcal{D}^*$, em que

- $\mathcal{X}$ é um alfabeto dos símbolos que se pretende codificar
- $\mathcal{D}$ é o alfabeto dos símbolos usados no output (ex. binário)
- $\mathcal{D}^*$ é o conjunto das strings finitas de símbolos de $\mathcal{D}$

- $C(x)$ é a palavra de código correspondente a x
- $l(x)$ é o comprimento da palavra de código $C(x)$
- $L(C)$ é o comprimento médio do código C

Example

x	$p(x)$	$C(x)$	$l(x)$
A	0.4	00	2
B	0.3	101	3
C	0.3	110	3

$$
\begin{aligned}
L(C) &= \sum_{x \in \mathcal{X}} p(x) l(x) \\
&= 0.4 \times 2 + 0.3 \times 3 + 0.3 \times 3 \\
&= 2.6 \text{ bits}
\end{aligned}
$$

Que propriedades deve um código satisfazer?

Definition (Código não singular)

Um código é não singular se símbolos diferentes têm palavras de código diferentes, *i.e.*

$$x_i \neq x_j \quad \Rightarrow \quad C(x_i) \neq C(x_j)$$

Que propriedades deve um código satisfazer?

Definition (Código não singular)

Um código é não singular se símbolos diferentes têm palavras de código diferentes, *i.e.*

$$x_i \neq x_j \quad \Rightarrow \quad C(x_i) \neq C(x_j)$$

Só garante a descodificação de símbolos isolados.

Example

x	$C(x)$
A	0
B	00

Este código é não singular.
No entanto a string "ABBA", codificada como "000000", não é descodificável univocamente.

Definition (Extensão de um código)

A extensão C^* de um código C é um novo código que codifica uma sequência de símbolos usando a sequência das respectivas palavras de código:

$$C^*(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n)$$

Definition (Extensão de um código)

A extensão C^* de um código C é um novo código que codifica uma sequência de símbolos usando a sequência das respectivas palavras de código:

$$C^*(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n)$$

Definition (Código univocamente descodificável)

Um código é univocamente descodificável se a sua extensão é não singular.

I.e., cada string pode apenas ter sido gerada por uma única mensagem.

Código univocamente descodificável

Example

x	$C(x)$
A	001
B	00
C	11
D	110

- É univocamente descodificável.
- Pode ser necessário analisar toda a sequência para descodificar o primeiro símbolo.
- Não é praticável quando a string é grande ou quando não termina (ex: streaming de radio pela internet).

Example

Descodifique a string:

$$0011111111111100000001$$

Será

ACCCC...

ou

BCCCC...?

Código instantaneo (ou de prefixo)

Definition (Código instantaneo)

Diz-se que um código C é um código instantaneo ou código de prefixo se nenhuma palavra de código é prefixo de outra.

Um código instantaneo pode ser descodificado sem referência às palavras de código futuras.

Definition (Código instantaneo)

Diz-se que um código C é um código instantaneo ou código de prefixo se nenhuma palavra de código é prefixo de outra.

Um código instantaneo pode ser descodificado sem referência às palavras de código futuras.

Example

x	$C(x)$
A	01
B	001
C	111
D	10

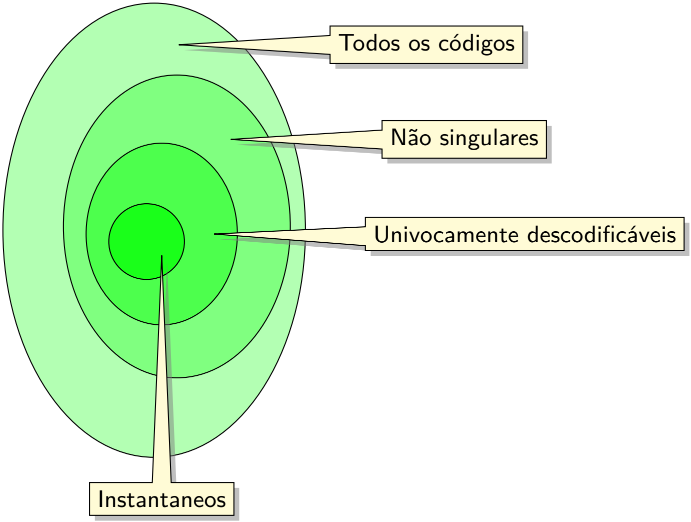
A string

$$0100110111001$$

é imediatamente reconhecida como

$$01,001,10,111,001$$

ou seja ABDCB


Todos os códigos
Não singulares
Univocamente descodificáveis
Instantaneos

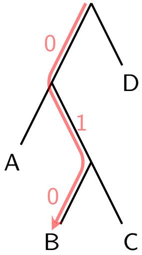
Exemplos de códigos

Example

x	Singular	Não singular	Univ. descod.	Instantaneo
A	0	0	10	00
B	0	00	00	10
C	1	000	11	111
D	1	0000	110	110

Instantaneo $\Rightarrow$ Univocamente descodificável $\Rightarrow$ Não singular

Um código instantaneo pode ser construído desenhando uma árvore e seleccionando como palavras de código o caminho desde a raiz até às folhas.



x	$C(x)$
A	00
B	010
C	011
D	1

Os símbolos estão nas folhas da árvore

Repare que a restrição "nenhuma palavra de código é prefixo de outra" obriga a que nenhum símbolo possa ser definido num nó interno da árvore.

Problema: Dados os comprimentos $l(x)$ das palavras de código, será que existe um código instantaneo que satisfaz esses comprimentos?

Theorem (Desigualdade de Kraft)

É possível construir um código instantaneo com palavras de código de comprimento $l(x)$ se e só se

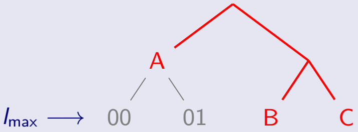
$$\sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1.$$

Demonstração.

- Seja $l_{\max}$ o tamanho da maior palavra de código (altura da árvore).

Demonstração.

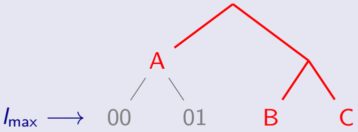
- Seja $l_{\max}$ o tamanho da maior palavra de código (altura da árvore).
- Qualquer palavra de código de comprimento $l(x) < l_{\max}$ é prefixo de $2^{l_{\max} - l(x)}$ palavras (não usadas) no nível $l_{\max}$.



Desigualdade de Kraft

Demonstração.

- Seja $l_{\max}$ o tamanho da maior palavra de código (altura da árvore).
- Qualquer palavra de código de comprimento $l(x) < l_{\max}$ é prefixo de $2^{l_{\max}-l(x)}$ palavras (não usadas) no nível $l_{\max}$.



- Podem existir até $2^{l_{\max}}$ palavras de código no nível $l_{\max}$. Então

$$\sum_{x \in \mathcal{X}} 2^{l_{\max}-l(x)} \leq 2^{l_{\max}} \quad \Leftrightarrow \quad \sum_{x \in \mathcal{X}} 2^{-l(x)} \leq 1.$$

$\square$

- Considere-se o conjunto de todos os códigos instantaneos (*i.e.*, satisfazem a desigualdade de Kraft).
- Códigos diferentes têm comprimentos médios diferentes.

Problem

Qual o código C que tem o menor comprimento médio $L(C)$?

Códigos óptimos

- Considere-se o conjunto de todos os códigos instantaneos (*i.e.*, satisfazem a desigualdade de Kraft).
- Códigos diferentes têm comprimentos médios diferentes.

Problem

Qual o código C que tem o menor comprimento médio $L(C)$?

- Sabemos construir um código instantaneo desenhando uma árvore.
- Falta saber que posição cada símbolo ocupa na árvore.
- Um código óptimo pode ser obtido calculando os comprimentos óptimos $l(x)$ das palavras de código.
  - Os símbolos mais frequentes devem ter comprimentos menores.
- O código obtém-se pendurando os símbolos na árvore à altura $l(x)$.
  - Existem vários códigos possíveis para os mesmos comprimentos $l(x)$, mas todos terão o mesmo comprimento médio.

Problem

Minimizar $L(C)$ com a restrição do código resultante ser um código instantaneo.

- *Funcional a optimizar:*
$$L(C) = \sum_x p(x) l(x).$$
- *Restrição:*
$$\sum_x 2^{-l(x)} \leq 1.$$

Usando o método dos multiplicadores de Lagrange obtém-se o funcional modificado:

$$J = \sum_x p(x) l(x) + \lambda \underbrace{\left(\sum_x 2^{-l(x)} - 1 \right)}_{\text{restrição}}.$$

Derivando J em ordem aos comprimentos $l(x)$ e ao multiplicador de Lagrange λ, e igualando a zero, obtêm-se os pontos de estacionariedade

$$\frac{\partial J}{\partial l(x_j)} = p(x_j) - \lambda 2^{-l(x_j)} \log_e 2 = 0 \qquad (1)$$

$$\frac{\partial J}{\partial \lambda} = \sum_x 2^{-l(x)} - 1 = 0 \qquad (2)$$

Da equação (1) resulta que

$$l(x_j) = -\log_2 p(x_j) + \log_2(\lambda \log_e 2) \qquad (3)$$

Substituindo (3) em (2), e resolvendo em ordem a λ, obtém-se $\lambda = 1/\log 2$. Substituindo este λ em (3) obtém-se finalmente

$$l(x_j) = -\log_2 p(x_j). \qquad (4)$$

Conclui-se que a probabilidade de um símbolo determina directamente o comprimento da sua palavra de código.
Podemos construir um código instantaneo que satisfaça estes comprimentos.

Código óptimo

Um código C cujos comprimentos são

$$l(x_j) = -\log_2 p(x_j)$$

tem comprimento médio

$$\begin{aligned} L(C) &= \sum_x p(x) l(x) \\ &= \sum_x p(x) \left(-\log_2 p(x)\right) \\ &= -\sum_x p(x) \log_2 p(x). \end{aligned}$$

Ou seja

$$L(C) = H(X)$$

A entropia dá o comprimento médio de um código óptimo!

Código de Shannon

- Em geral, os comprimentos $l(x_j)$ não são números inteiros. Shannon propôs fazer o arredondamento para cima

$$l(x) = \lceil -\log_2 p(x) \rceil$$

satisfazendo ainda assim a desigualdade de Kraft.

- Construção das palavras de código:
  1. Ordenar as probabilidades por ordem decrescente.
  2. Calcular a probabilidade acumulada $F(x) = \sum_{i<x} p(i)$.
  3. A palavra de código $C(x)$ é formada pelos primeiros $l(x)$ bits a seguir à vírgula na expansão binária de $F(x)$.

Example

x	$p(x)$	$F(x)$	$F(x)_{\text{bin}}$	$l(x)$	$C(x)$
B	0.7	0.0	0.0000...	1	0
C	0.2	0.7	0.1011...	3	101
A	0.1	0.9	0.1110...	4	1110

O comprimento médio é $L(C) = 1.7$ bits.
A entropia é $H(X) \approx 1.157$ bits.

- Se as probabilidades $p(x)$ forem potências negativas de 2,

$$2^{-1} = \frac{1}{2}, \quad 2^{-2} = \frac{1}{4}, \quad 2^{-3} = \frac{1}{8}, \ldots$$

então os logaritmos dão números inteiros e $L(C) = H(X)$.

- Geralmente isso não acontece e os arredondamentos levam a que $H(X) \leq L(C)$. Portanto, a entropia dá o valor mínimo atingível para o comprimento médio de códigos instantaneos.
- O arredondamento introduz alguma ineficiência. Como os erros de arredondamento são sempre inferiores à unidade, o comprimento médio satisfaz

$$L(C) < H(X) + 1$$

A ineficiência medida por $L(C) - H(X)$ é sempre inferior a um bit.

Comprimento médio do código de Shannon

$$H(X) \leq L(C) < H(X) + 1$$

Example

Uma fonte sem memória gera símbolos do alfabeto $\mathcal{X} = \{A, B, C, D\}$ com probabilidadades $\{0.1, 0.2, 0.3, 0.4\}$, respectivamente.
O código de Shannon é:

x	$p(x)$	$F(x)$	$F(x)_{\text{bin}}$	$-\log p(x)$	$l(x)$	$C(x)$
D	0.4	0.0	0.0000	1.3219	2	00
C	0.3	0.4	0.0110	1.7370	2	01
B	0.2	0.7	0.1011	2.3219	3	101
A	0.1	0.9	0.1110	3.3219	4	1110

$$L(C) = 2.4 \text{ bits}$$
$$H(X) = 1.8464 \text{ bits}$$
$$1.8464 \leq L(C) < 2.8464$$

O código não é óptimo.

Example

Uma fonte sem memória gera símbolos do alfabeto $\mathcal{X} = \{A, B, C, D\}$ com probabilidadades $\{0.1, 0.2, 0.3, 0.4\}$.
É possível construir um código de Shannon com comprimento médio mais perto de $H(X)$ do que o obtido no exemplo anterior.
Para o efeito, agrupam-se vários símbolos e constrói-se um código para esses grupos (que são os "novos" símbolos).

$x_i x_{i+1}$	$p(x_i, x_{i+1})$	$l(x_i x_{i+1})$
AA	0.01	7
AB	0.02	6
AC	0.03	6
AD	0.04	5
BA	0.02	6
BB	0.04	5
$\vdots$	$\vdots$	$\vdots$
DD	0.16	3

$$L(C) = \frac{\sum_y p(x_i, x_{i+1}) l(x_i x_{i+1})}{2} = 2.155 \text{ bits}$$

Agrupando 2 símbolos obtém-se um comprimento médio

$$H(X) \leq L(C) < H(X) + \frac{1}{2}$$

O erro de arredondamento é diluído por dois símbolos e globalmente obtém-se melhor desempenho.

O código de Shannon-Fano funciona do seguinte modo:

1. Ordenam-se as probabilidades por ordem decrescente.
2. Dividem-se dois grupos com probabilidades o mais equilibradas possível (perto de 50%).
3. Atribui-se os bits 0 e 1 a cada um dos grupos anteriores.
4. Subdividem-se sucessivamente cada um dos grupos com a mesma estratégia até não haver mais subdivisões possíveis.

Este algoritmo usa uma estratégia *greedy* e é sub-óptimo. Quase nunca é utilizado. O código de Huffman é igualmente simples e produz melhores resultados.

Exemplos de códigos de Shannon-Fano

Example

Uma fonte sem memória gera símbolos do alfabeto $\mathcal{X} = \{A, B, C\}$ com probabilidadades $\{0.3, 0.1, 0.6\}$, respectivamente.
O código Shannon-Fano é:

x	$p(x)$	$C(x)$
C	0.6	0
A	0.3	10
B	0.1	11

$$L(C) = 1.4 \text{ bits}$$
$$H(X) \approx 1.295 \text{ bits}$$
$$1.295 \leq L(C) < 2.295$$

Código de Huffman

- Requesitos:
  - Conhecer *a priori* as probabilidades $p(x)$

- Requesitos:
  - Conhecer *a priori* as probabilidades $p(x)$
- Algoritmo:
  - Todos os símbolos são folhas de uma árvore a construir.
  - Em cada passo seleccionam-se os dois nós de menor probabilidade para formar o novo nó pai.
  - As palavras de código correspondem ao caminho da raiz até às folhas.

Código de Huffman

- Requesitos:
  - Conhecer *a priori* as probabilidades $p(x)$
- Algoritmo:
  - Todos os símbolos são folhas de uma árvore a construir.
  - Em cada passo seleccionam-se os dois nós de menor probabilidade para formar o novo nó pai.
  - As palavras de código correspondem ao caminho da raiz até às folhas.
- Optimalidade:
  - O código de Huffman satisfaz

$$H(X) \leq L(C) < H(X) + 1$$

  - É o melhor código instantaneo que se pode construir para um dado alfabeto.

- Requesitos:
  - Conhecer *a priori* as probabilidades $p(x)$
- Algoritmo:
  - Todos os símbolos são folhas de uma árvore a construir.
  - Em cada passo seleccionam-se os dois nós de menor probabilidade para formar o novo nó pai.
  - As palavras de código correspondem ao caminho da raiz até às folhas.
- Optimalidade:
  - O código de Huffman satisfaz

$$H(X) \leq L(C) < H(X) + 1$$

  - É o melhor código instantaneo que se pode construir para um dado alfabeto.
- Desvantagens:
  - Assume que os símbolos são v.a. independentes.
  - Assume que as probabilidades não variam no tempo.
  - O compressor e o descompressor têm de conhecer a mesma árvore (necessário enviar a árvore para o descompressor descodificar a mensagem).

Código de Huffman

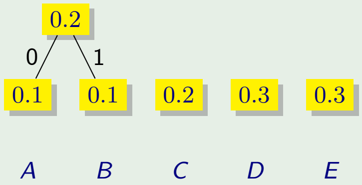
Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:

0.1	0.1	0.2	0.3	0.3
A	B	C	D	E

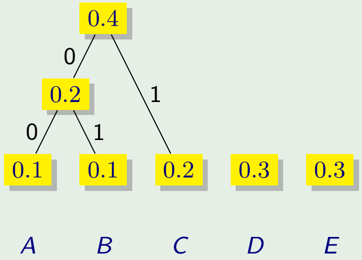
Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



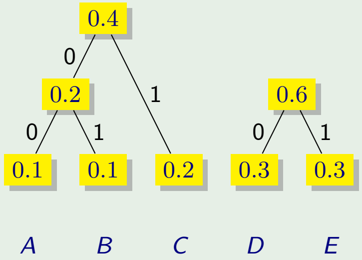
Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



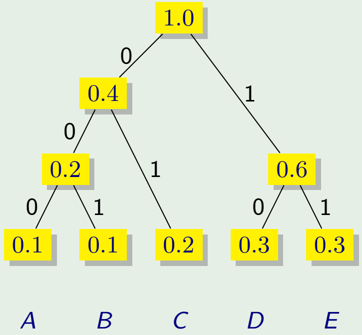
Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



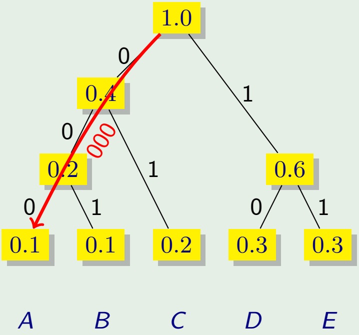
Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



Example

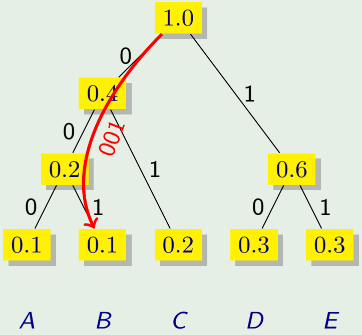
Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



As palavras de código lêem-se da raíz até às folhas.

Example

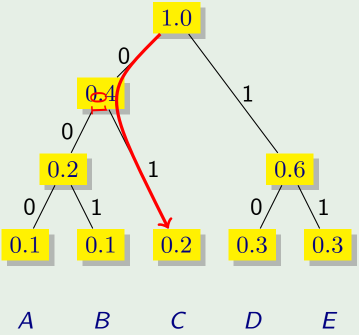
Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



As palavras de código lêem-se da raíz até às folhas.

Example

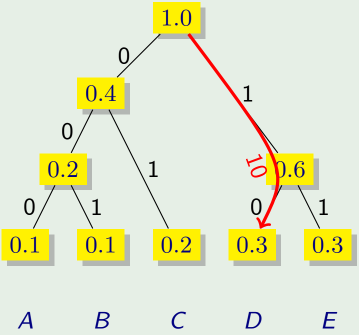
Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



As palavras de código lêem-se da raíz até às folhas.

Example

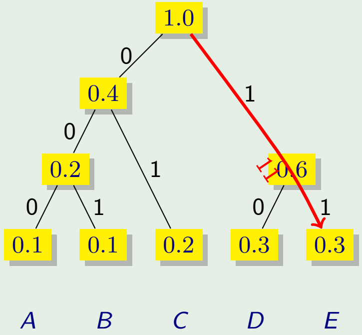
Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



As palavras de código lêem-se da raíz até às folhas.

Example

Considere um alfabeto $\{A, B, C, D, E\}$ com probabilidades $\{0.1, 0.1, 0.2, 0.3, 0.3\}$. O código de Huffman constrói-se do seguinte modo:



As palavras de código lêem-se da raíz até às folhas.

- O código de Huffman anterior só pode ser construído depois de conhecer as probabilidades da fonte.

- O código de Huffman anterior só pode ser construído depois de conhecer as probabilidades da fonte.
- No caso de se pretender comprimir uma string gerada por uma fonte desconhecida (*e.g.* um ficheiro), é necessário determinar a frequência com que os símbolos ocorrem.

- O código de Huffman anterior só pode ser construído depois de conhecer as probabilidades da fonte.
- No caso de se pretender comprimir uma string gerada por uma fonte desconhecida (*e.g.* um ficheiro), é necessário determinar a frequência com que os símbolos ocorrem.
- São necessárias duas passagens sobre o ficheiro:
  1. Contar o número de ocorrências de cada símbolo;
  2. Desenhar código e comprimir ficheiro.

Código de Huffman

- O código de Huffman anterior só pode ser construído depois de conhecer as probabilidades da fonte.
- No caso de se pretender comprimir uma string gerada por uma fonte desconhecida (*e.g.* um ficheiro), é necessário determinar a frequência com que os símbolos ocorrem.
- São necessárias duas passagens sobre o ficheiro:
  1. Contar o número de ocorrências de cada símbolo;
  2. Desenhar código e comprimir ficheiro.

Agrupando n símbolos para formar "novos" símbolos, obtém-se um comprimento médio no intervalo

$$H(X_1, \ldots, X_n) \leq \underbrace{E[l(X_1, \ldots, X_n)]}_{\text{comprimento médio}} < H(X_1, \ldots, X_n) + 1$$

Código sobre grupos de símbolos

Agrupando n símbolos para formar "novos" símbolos, obtém-se um comprimento médio no intervalo

$$H(X_1, \ldots, X_n) \leq \underbrace{E[l(X_1, \ldots, X_n)]}_{\text{comprimento médio}} < H(X_1, \ldots, X_n) + 1$$

Como os símbolos são i.i.d. (fonte sem memória):

$$H(X_1, \ldots, X_n) = nH(X).$$

Código sobre grupos de símbolos

Agrupando n símbolos para formar "novos" símbolos, obtém-se um comprimento médio no intervalo

$$H(X_1, \ldots, X_n) \leq \underbrace{E[l(X_1, \ldots, X_n)]}_{\text{comprimento médio}} < H(X_1, \ldots, X_n) + 1$$

Como os símbolos são i.i.d. (fonte sem memória):

$$H(X_1, \ldots, X_n) = nH(X).$$

Substituindo em cima obtém-se

$$nH(X) \leq E[l(X_1, \ldots, X_n)] < nH(X) + 1.$$

Agrupando n símbolos para formar "novos" símbolos, obtém-se um comprimento médio no intervalo

$$H(X_1, \ldots, X_n) \leq \underbrace{E[l(X_1, \ldots, X_n)]}_{\text{comprimento médio}} < H(X_1, \ldots, X_n) + 1$$

Como os símbolos são i.i.d. (fonte sem memória):

$$H(X_1, \ldots, X_n) = nH(X).$$

Substituindo em cima obtém-se

$$nH(X) \leq E[l(X_1, \ldots, X_n)] < nH(X) + 1.$$

Dividindo por n obtém-se

o comprimento médio por símbolo individual

$$H(X) \leq \frac{E[l(X_1, \ldots, X_n)]}{n} < H(X) + \frac{1}{n}.$$

Aplicação do código numa fonte diferente

Problem

- *Uma fonte sem memória gera símbolos com probabilidades* $p(x)$*.*
- *Constrói-se um código usando probabilidades* $q(x)$ *de uma fonte diferente.*
- *Ao aplicar o código à fonte* $p(x)$*, qual vai ser a penalização no comprimento médio* $L(C)$*?*

Problem

- *Uma fonte sem memória gera símbolos com probabilidades $p(x)$.*
- *Constrói-se um código usando probabilidades $q(x)$ de uma fonte diferente.*
- *Ao aplicar o código à fonte $p(x)$, qual vai ser a penalização no comprimento médio $L(C)$?*

A penalização será a diferença entre o comprimento médio e a entropia da fonte. Isto é,

$$\begin{aligned} L(C) - H(X) &= \sum_x p(x) \lceil -\log_2 q(x) \rceil + \sum_x p(x) \log_2 p(x) \\ &\geq \sum_x p(x) \big(-\log_2 q(x) \big) + \sum_x p(x) \log_2 p(x) \\ &= \sum_x p(x) \log \frac{p(x)}{q(x)} \qquad \textbf{(Kullback-Leibler)}. \end{aligned}$$

Código de Huffman adaptativo

O código de Huffman adaptativo permite fazer compressão nas seguintes condições:

- não necessita de conhecer as probabilidades dos símbolos.
- faz apenas uma passagem sobre o ficheiro.
- não é necessário transmitir o código. O receptor consegue reconstruir a árvore à medida que faz a descodificação.

O algoritmo foi desenvolvido independentemente por Faller (1973) e Galager (1978) e melhorado por Knuth (1985), ficando conhecido como algoritmo FGK. Existe ainda outro melhoramento devido a Vitter (1987), conhecido como algoritmo V, que não é apresentado aqui.
O código de Huffman adaptativo é usado no comando `compact` do UNIX.

Código de Huffman adaptativo: ideias chave

- Construir uma árvore binária à medida que são lidos novos símbolos.

Código de Huffman adaptativo: ideias chave

- Construir uma árvore binária à medida que são lidos novos símbolos.
- Reserva um símbolo de `ESCAPE` para representar todos os símbolos que ainda não ocorreram até ao momento actual.

Código de Huffman adaptativo: ideias chave

- Construir uma árvore binária à medida que são lidos novos símbolos.
- Reserva um símbolo de `ESCAPE` para representar todos os símbolos que ainda não ocorreram até ao momento actual.
- Manipular a árvore de modo a aproximar-se de uma árvore de Huffman:
  1. Cada nível da árvore não pode conter nós com número de ocorrências superior aos níveis de cima: ordenar nós de baixo para cima de modo a que os símbolos mais frequentes estejam mais acima.
  2. As ocorrências devem estar ordenadas da esquerda para a direita em cada nível da árvore.

Código de Huffman adaptativo: ideias chave

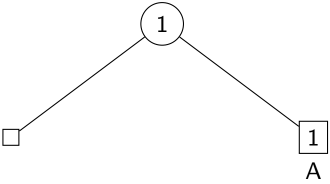
- Construir uma árvore binária à medida que são lidos novos símbolos.
- Reserva um símbolo de `ESCAPE` para representar todos os símbolos que ainda não ocorreram até ao momento actual.
- Manipular a árvore de modo a aproximar-se de uma árvore de Huffman:
  1. Cada nível da árvore não pode conter nós com número de ocorrências superior aos níveis de cima: ordenar nós de baixo para cima de modo a que os símbolos mais frequentes estejam mais acima.
  2. As ocorrências devem estar ordenadas da esquerda para a direita em cada nível da árvore.
- Para codificar um símbolo, verifica-se se este já existe na árvore e:
  - Se já existe, usa-se o código correspondente e incrementam-se as ocorrências desse símbolo e dos seus ascendentes.
  - Se não existe, usa-se o código de "escape" seguido do símbolo sem ser codificado, acrescenta-se o símbolo à árvore por baixo de onde antes estava o "escape", e actualizam-se as ocorrências dos nós ascendentes.

  Em qualquer um dos casos anteriores, modifica-se a árvore de modo a que as ocorrências estejam ordenadas de baixo para cima e depois da esquerda para a direita.

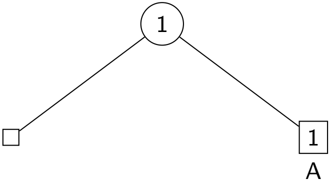
Código de Huffman adaptativo

"ABRACADABRA" $\longrightarrow$ A

"ABRACADABRA" $\longrightarrow$ A

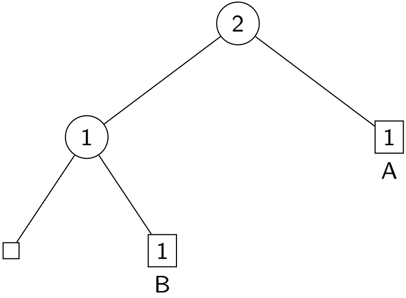


"ABRACADABRA" $\longrightarrow$ A 0B

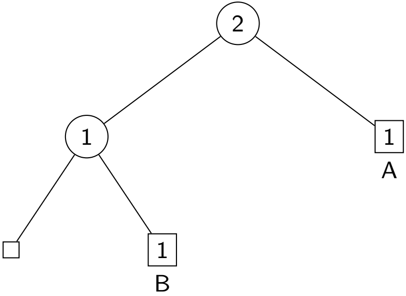


Código de Huffman adaptativo

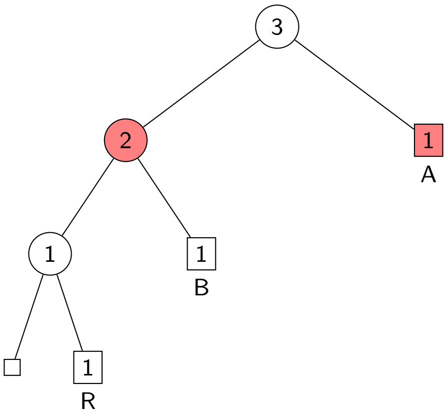
"ABRACADABRA" $\longrightarrow$ A 0B



"ABRACADABRA" $\longrightarrow$ A 0B 00R

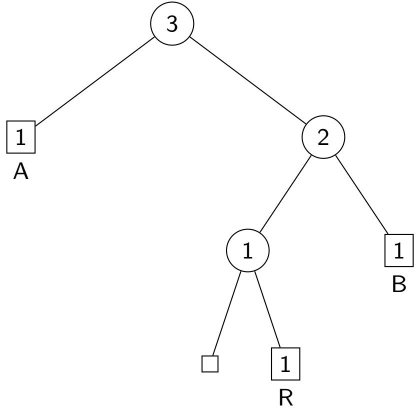


"ABRACADABRA" $\longrightarrow$ A 0B 00R

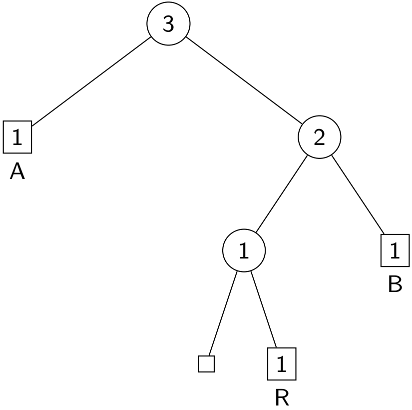


Código de Huffman adaptativo

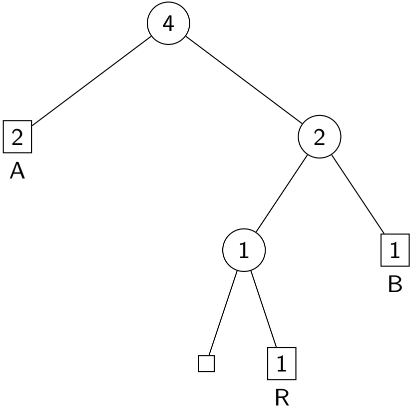
“ABRACADABRA” $\longrightarrow$ A 0B 00R



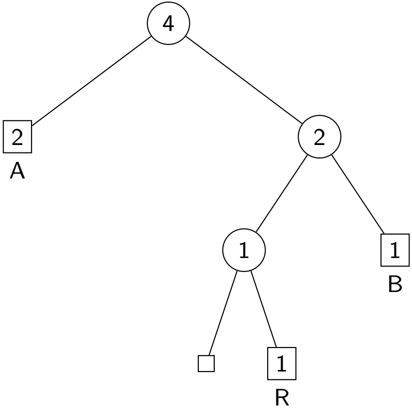
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0



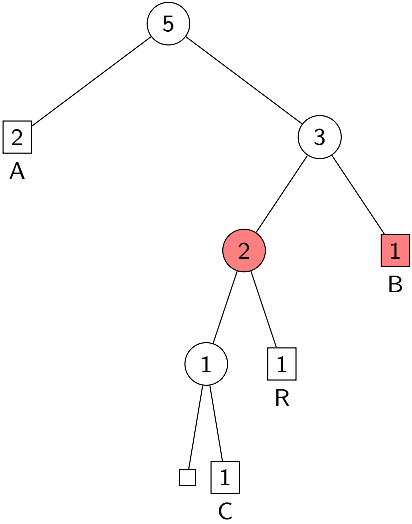
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0



"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C

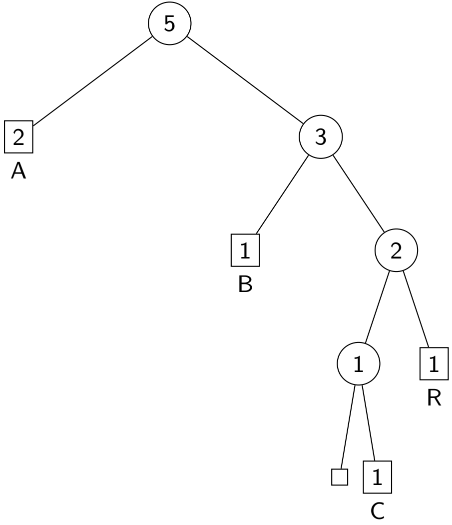


"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C

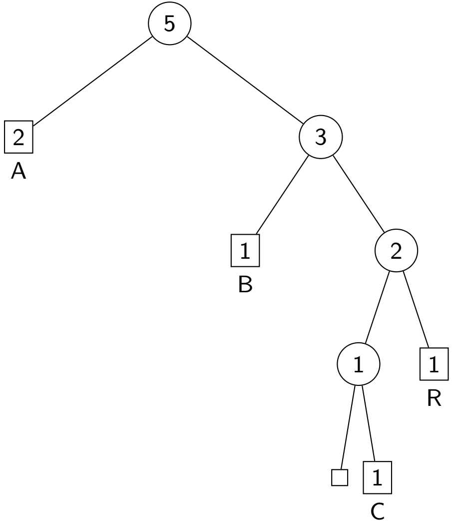


Código de Huffman adaptativo

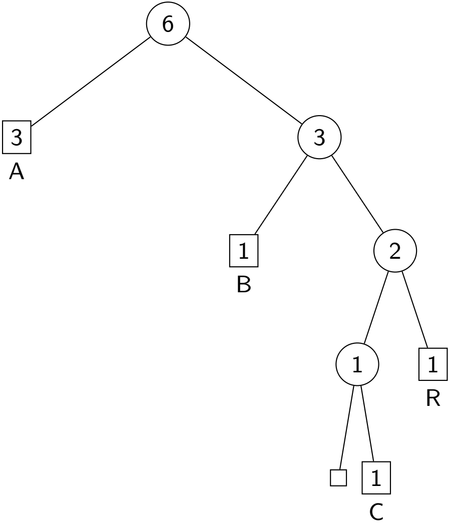
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C



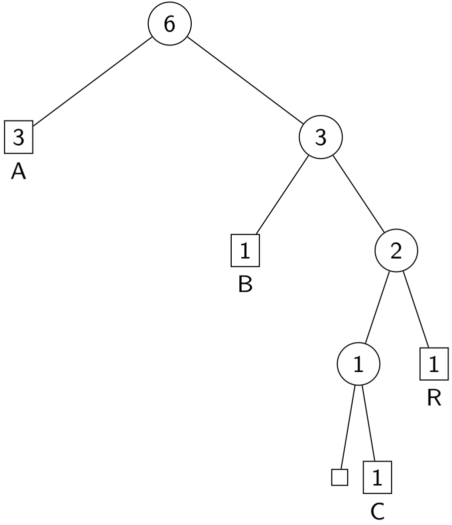
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0



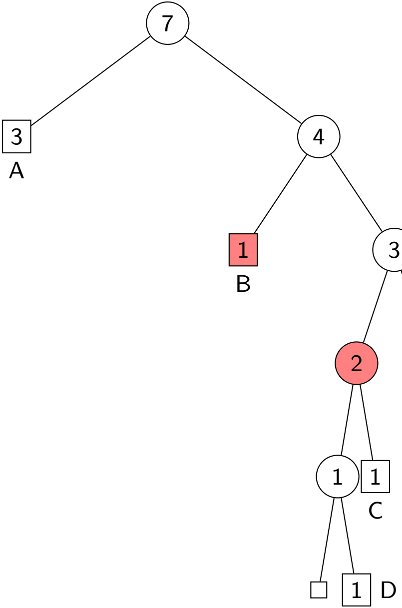
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0



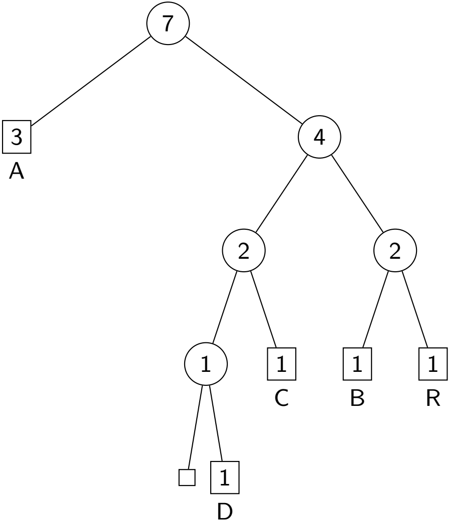
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D



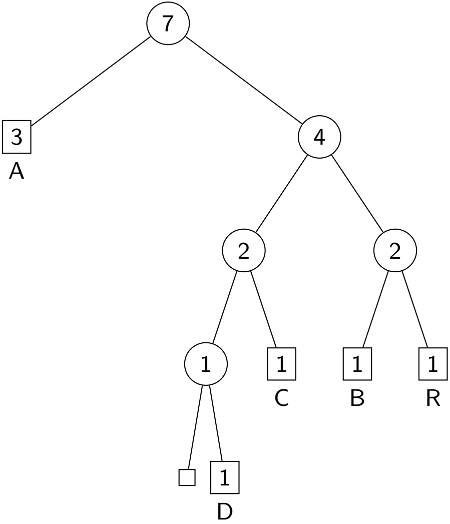
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D



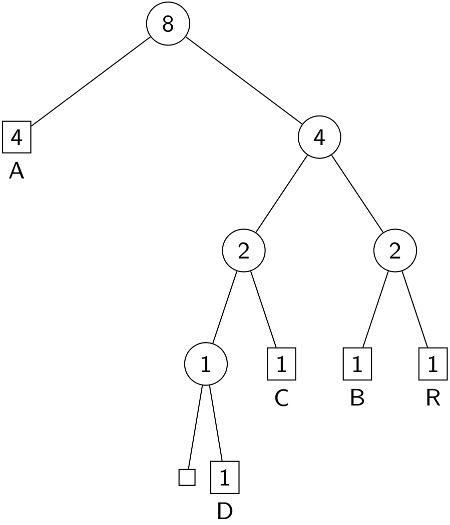
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D



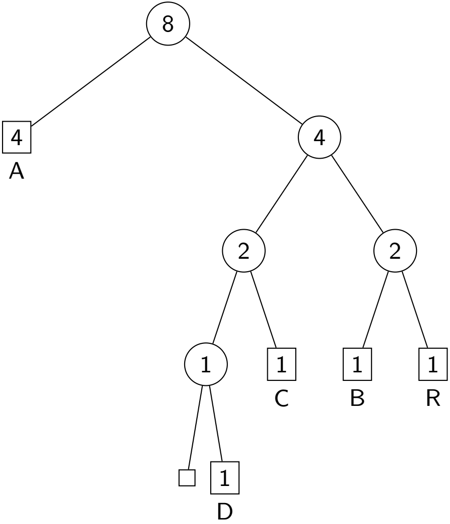
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0



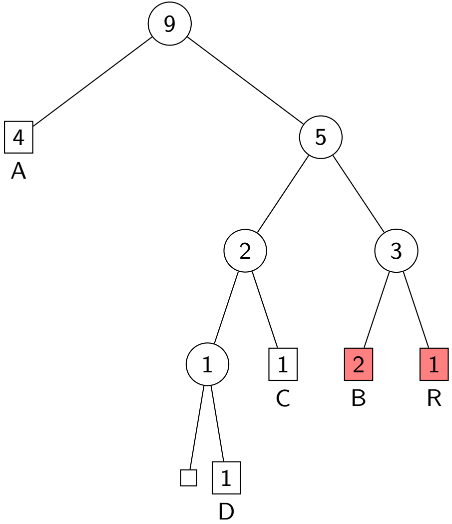
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0



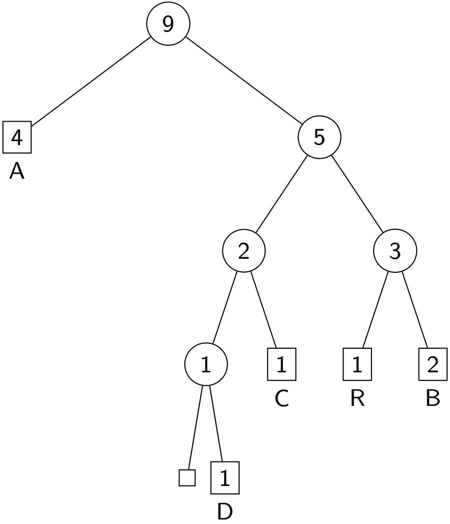
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110



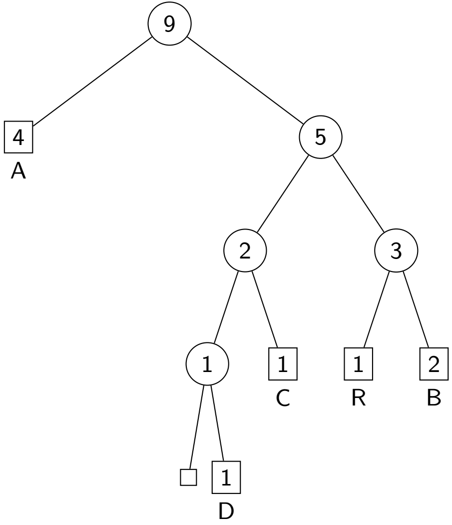
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110



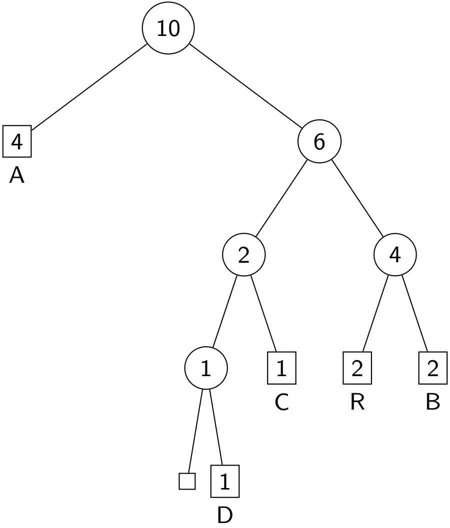
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110



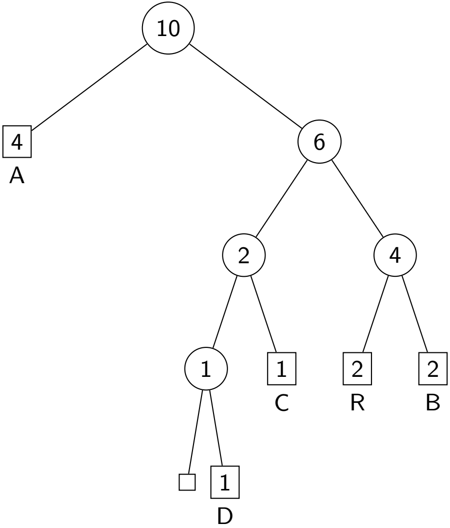
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110 110



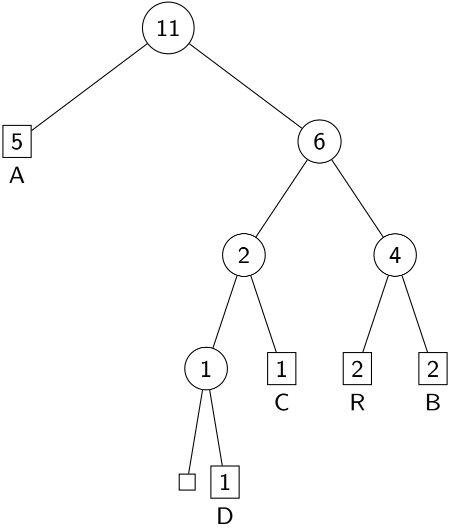
"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110 110



"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110 110 0



"ABRACADABRA" $\longrightarrow$ A 0B 00R 0 100C 0 1100D 0 110 110 0



"A B R A C A D A B R A"

- Em ASCII (88 bits):
  01000001 01000010 01010010 01000001 01000011 01000001
  01000100 01000001 01000010 01010010 01000001
- Com Huffman adaptativo + ASCII (60 bits):
  01000001 0 01000010 00 01010010 0 100 01000011 0 1100
  01000100 0 110 110 0

Cada símbolo só é representado uma única vez em ASCII: na sua primeira ocorrência.
Nas ocorrências seguintes é substituído por palavras de código tipicamente mais curtas, cujo comprimento pode variar dependendo da frequência com que ocorreu até ao momento.