Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009
2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff) `http://www.ime.usp.br/~pf/analise_de_algoritmos`

Algoritmos gulosos

Salto do Sapo

Salto do Sapo

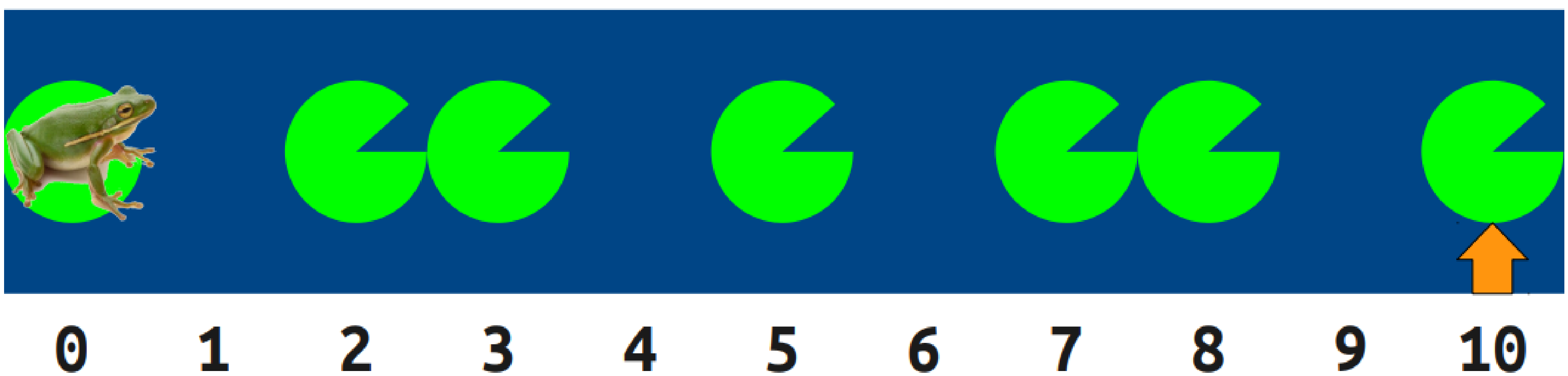
Existem n pedras numa reta numérica, em posições distintas $p_1, p_2, \cdots, p_n$. Dizemos que um sapo pode saltar de uma pedra p_i para outra pedra p_j desde que a distância entre elas seja menor ou igual a $delta$.

Salto do Sapo

Considere um sapo localizado inicialmente na pedra p_1. Qual é o menor número de saltos que ele precisa dar para chegar na pedra p_n?

Ou seja, é dado um vetor de n números distintos ordenados $p = \{p_1, p_2, \cdots, p_n\}$ e um número $delta$.

Salto do Sapo



Salto do Sapo

Uma sequência $u = u_1, u_2, \cdots, u_k$ é solução se:

- $\triangleright$ $u_1 = p_1$
- $\triangleright$ $u_k = p_n$
- $\triangleright$ $u_i = p_j$ para todo $i \in [1, k]$ e algum $j \in [1, n]$
- $\triangleright$ $|u_i - u_{i+1}| \leq delta$ para $i \in [1, k-1]$

Queremos uma sequência u que satisfaça as propriedades acima e que o tamanho k de u seja o mínimo possível. Vamos assumir que sempre existe pelo menos uma solução.

Salto do Sapo

Exemplo 1: entrada $n = 4$, $p = \{1, 2, 3, 4\}$, $delta = 1$.

$\triangleright$ existem diversas soluções possíveis entre elas $\{1, 2, 3, 4\}$, $\{1, 2, 1, 2, 3, 4\}, \{1, 2, 1, 2, 3, 2, 3, 4\}$

A sequência de menor $k = 4$ é $u = \{1, 2, 3, 4\}$

Salto do Sapo

Exemplo 2: entrada $n = 6$, $p = \{1, 2, 3, 5, 6, 7\}$, $delta = 2$.

tem como solução ótima $u = \{1, 3, 5, 7\}$

Salto do Sapo

Exemplo 3: entrada $n = 3$, $p = \{1, 3, 4\}$, $delta = 1$.

não admite solução, já que a partir de 1 não é possível alcançar 3 ou 4.

Vamos desconsiderar este caso.

Salto do Sapo

Observe que:

- ▷ nunca vale "voltar", pois isso aumentaria a sequência de números
- ▷ O sapo deve pular o mais longe possível.

Salto do Sapo

salto_sapo($p, n, delta$)

$u \leftarrow \{p[1]\}$
$ultima\_pos \leftarrow p[1]$
para $i = 2$ **até** n **faça**
    se $p[i] - ultima\_pos > delta$ **então**
        $ultima\_pos \leftarrow p[i-1]$
        $u \leftarrow u \cup p[i-1]$
$u \leftarrow u \cup p[n]$
retorne u

complexidade: $O(n)$

Seleção de paradas

Seleção de paradas

Problema:

- ▷ Viagem da cidade A para a cidade B ao longo de uma rodovia.
- ▷ Tanque de combustível tem capacidade suficiente para cobrir n quilômetros.
- ▷ Mapa indica a localização dos postos de combustível ao longo do caminho.

Objetivo: minimizar a quantidade de paradas ao longo da viagem.

Seleção de paradas

solução gulosa: Avançar a viagem o máximo que puder antes de reabastecer

Seleção de paradas

algoritmo do caminhoneiro(p, n, C)

$S \leftarrow \{\}$ $\triangleright$ paradas selecionadas
$ultima\_parada = 0$
para $i = 2$ **até** n **faça**
    se $p_i - ultima\_parada > C$ **então**
        $ultima\_parada = p_{i-1}$
        $S \leftarrow S \cup (i-1)$
retorne S

complexidade: $O(n)$

Problema da seleção de atividades

Problema da seleção de atividades

Considere um conjunto $\{1, 2, \cdots, n\}$ de n atividades que competem por um recurso, por exemplo uma sala de aula.

Cada atividade tem um início s_i e um término t_i, com $s_i \leq t_i$.

O intervalo requerido pela atividade é $[s_i, t_i)$.

Problema da seleção de atividades

Duas atividades i e j são compatíveis se os intervalos $[s_i, t_i)$ $[s_j, t_j)$ não se interceptam ($s_i \geq t_j$ ou $s_j \geq t_i$).

Problema: encontrar o conjunto de atividades mutuamente compatíveis de tamanho máximo.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

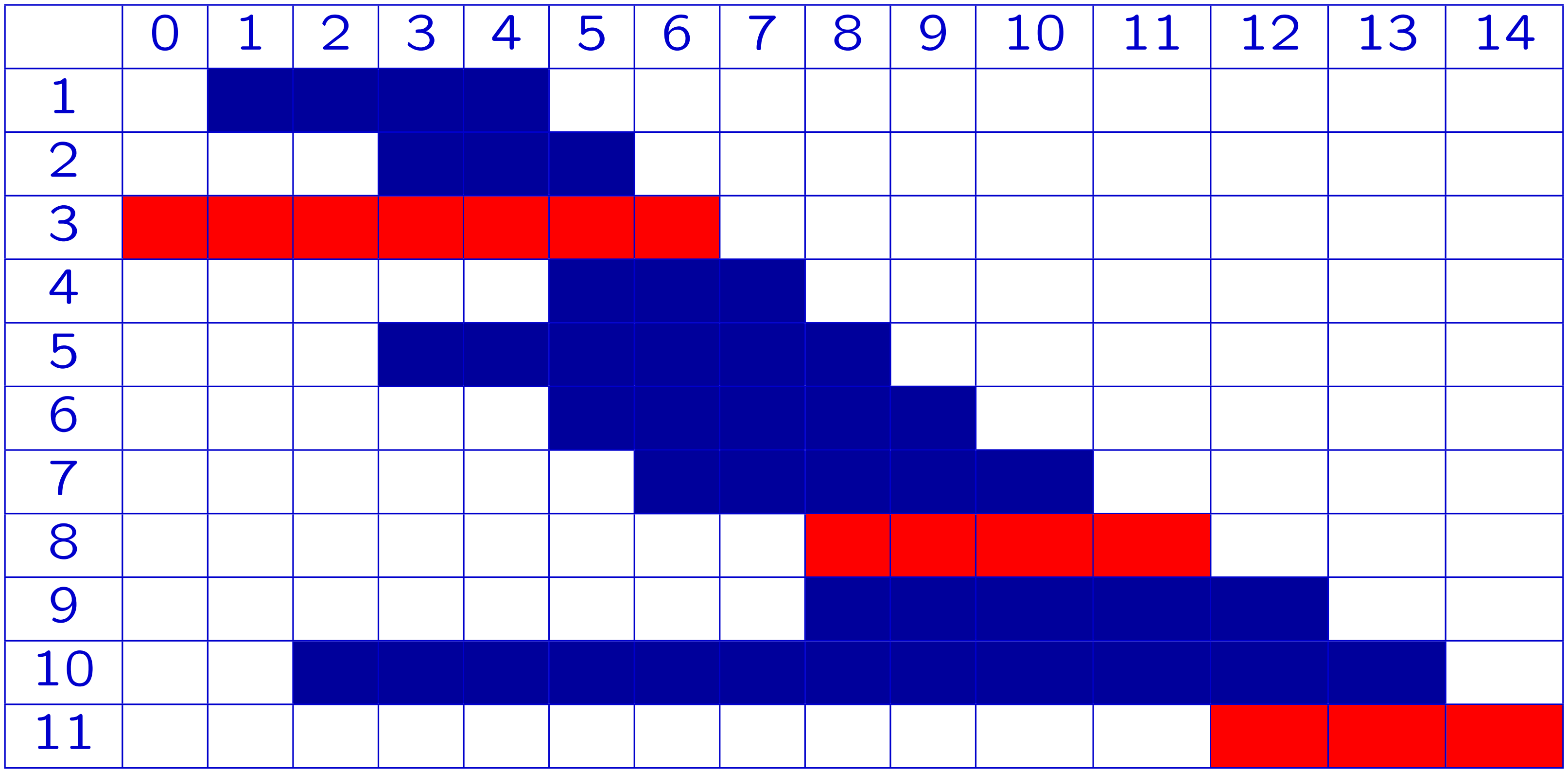
Podemos verificar três estratégias:

$\triangleright$ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.
$\triangleright$ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.
$\triangleright$ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Problema da seleção de atividades

1^a tentativa: Escolher primeiro as atividades que começam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

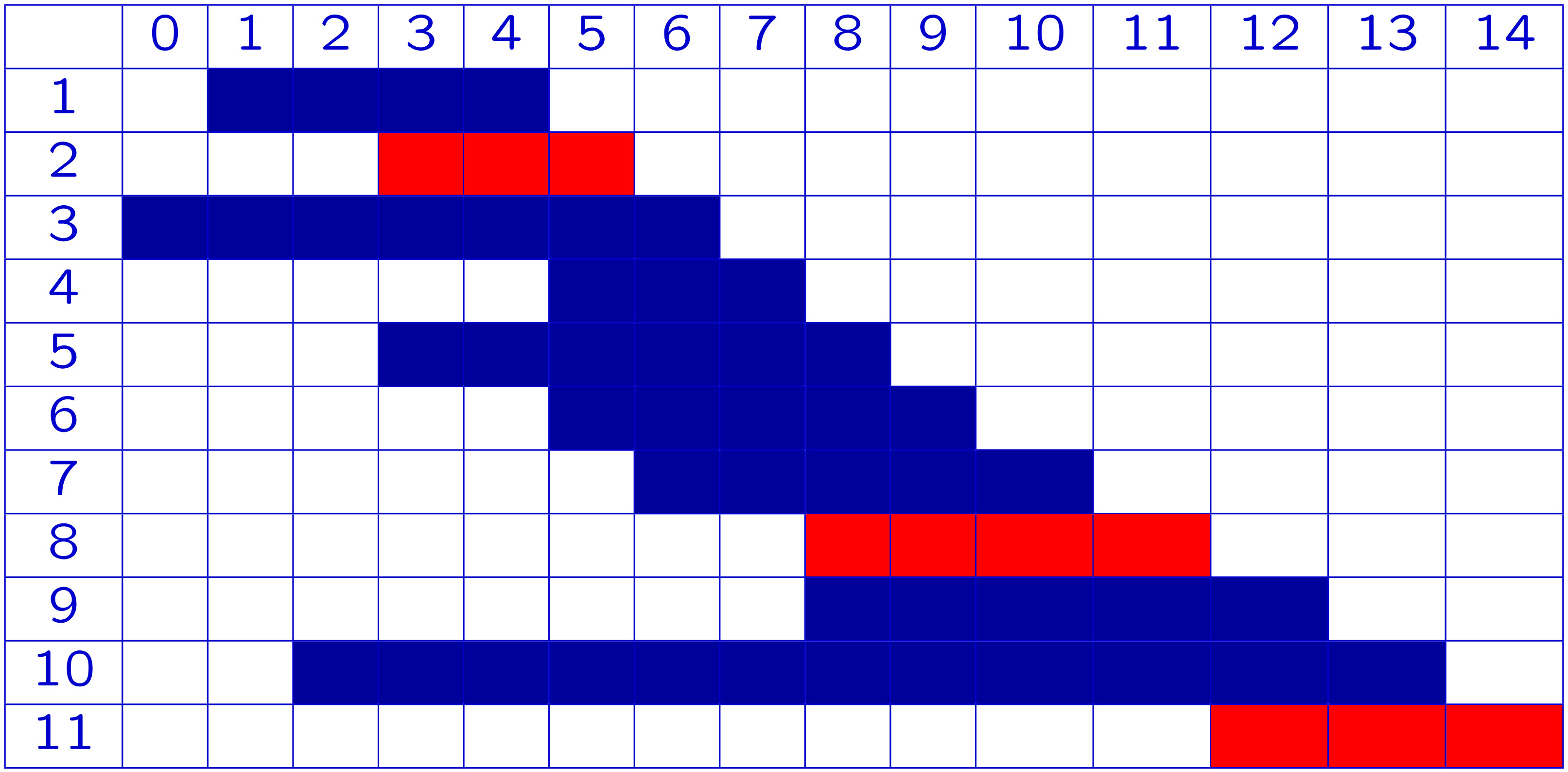


1^a tentativa: Escolher primeiro as atividades que começam primeiro. Escolhemos as atividades 3, 8 e 11.

Problema da seleção de atividades

2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.

Exemplo: Considerando 11 atividades em 14 unidades de tempo

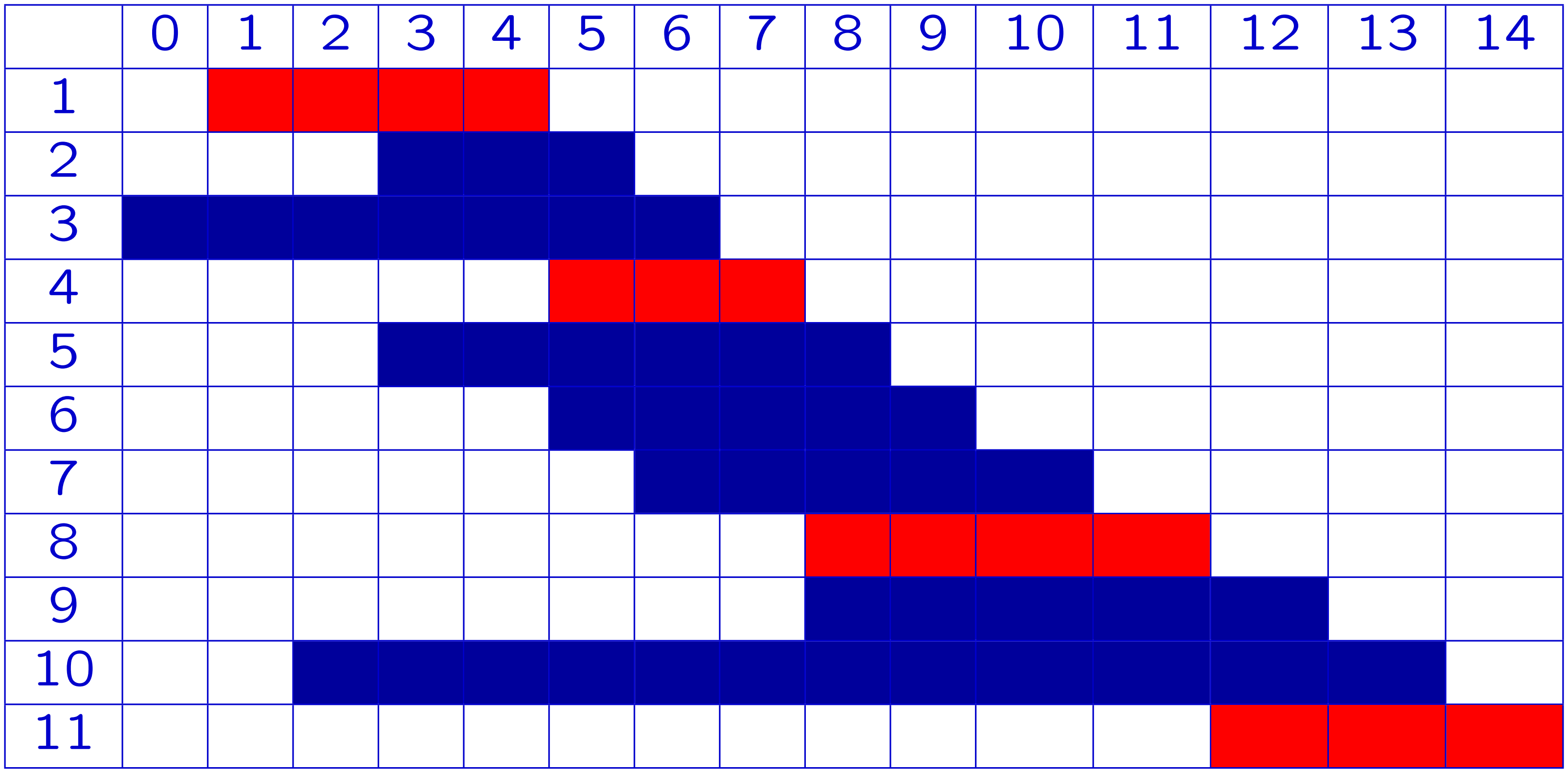


2^a tentativa: Escolher primeiro as atividades que demoram menos tempo. Escolhemos as atividades 2, 8 e 11.

Problema da seleção de atividades

3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

Exemplo: Considerando 11 atividades em 14 unidades de tempo



3^a tentativa: Escolher primeiro as atividades que terminam primeiro. Escolhemos as atividades 1, 4, 8 e 11.

Problema da seleção de atividades

Exemplo: Considerando 11 atividades em 14 unidades de tempo

$\triangleright$ 1^a tentativa: Escolher primeiro as atividades que começam primeiro.
$\triangleright$ 2^a tentativa: Escolher primeiro as atividades que demoram menos tempo.
$\triangleright$ 3^a tentativa: Escolher primeiro as atividades que terminam primeiro.

A 3^a tentativa apresentou a melhor estratégia de seleção de atividade. Dizemos que a solução é **ótima** para esta instância!

Problema da seleção de atividades

seleciona_atividades_guloso(s, t, n)

ordene s e t de tal forma que
$t[1] \leq t[2] \leq \cdots \leq t[n]$
$A \leftarrow \{1\}$ $\triangleright$ $t_1 \leq t_2 \leq \cdots \leq t_n$
$j \leftarrow 1$
para $i = 2$ **até** n
  faça se $s_i \geq t_j$ $\triangleright$ i e j são compatíveis
    então $A \leftarrow A \cup \{i\}$
      $j \leftarrow i$
retorne A

Complexidade do algoritmo: $O(n \lg n)$

Estratégia gulosa

Ingredientes chaves de um algoritmo guloso:

- ▷ subestrutura ótima
- ▷ característica gulosa: Solução ótima global pode ser produzida a partir de uma escolha ótima local.

Código de Huffman

Código de Huffman

▷ Técnica de compressão de dados (economia de 20% a 90%, dependendo do arquivo a ser comprimido).

▷ O algoritmo guloso de Huffman usa uma **tabela de frequência de caracteres** para obter um código binário único (**código prefixo**) de cada caracter encontrado no arquivo de entrada.

Código de Huffman

Suponha um arquivo texto contendo 100000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As frequências de cada caracter no arquivo são indicadas na tabela abaixo.

	a	b	c	d	e	f
(1) Frequência (em milhares)	45	13	12	16	9	5
(2) Código de tamanho fixo	000	001	010	011	100	101
(3) Código de tamanho variável	0	101	100	111	1101	1100

Código de Huffman

Suponha um arquivo texto contendo 100000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As frequências de cada caracter no arquivo são indicadas na tabela abaixo.

	a	b	c	d	e	f
(1) Frequência (em milhares)	45	13	12	16	9	5
(2) Código de tamanho fixo	000	001	010	011	100	101
(3) Código de tamanho variável	0	101	100	111	1101	1100

Tamanho em bits do arquivo:

(1) codifica 800000 bits

(2) codifica 300000 bits

(3) codifica 224000 bits (melhor)

Código de Huffman

Problema da codificação: Dadas as frequências de ocorrências dos caracteres de um arquivo, encontrar a sequência de bits (códigos) para representá-los de modo que o arquivo comprimido tenha tamanho mínimo.

Código de Huffman

Códigos livres de prefixos: o código de um símbolo não é prefixo do código de nenhum outro símbolo.

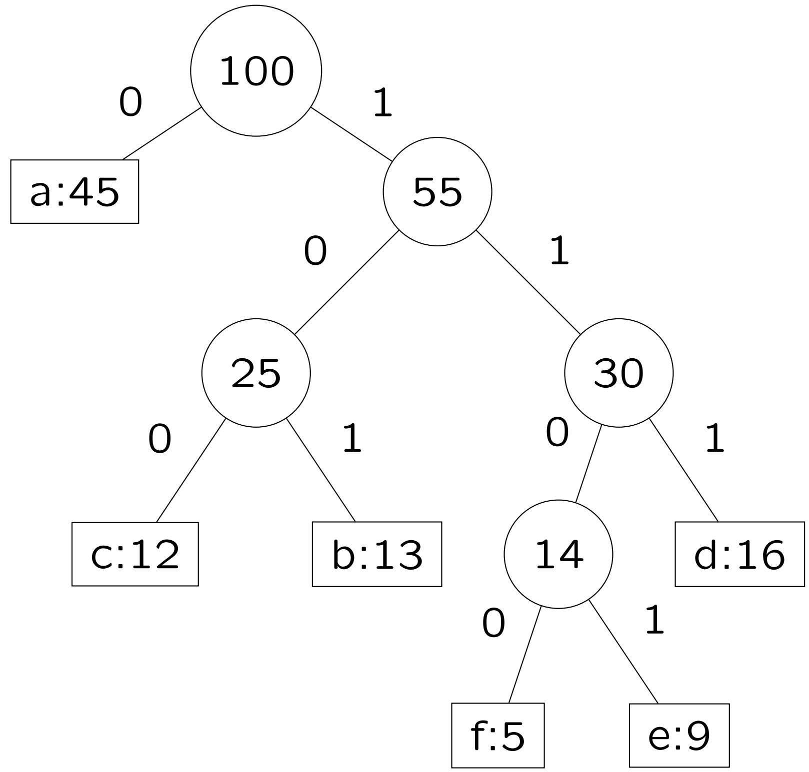
Código de Huffman

Códigos livres de prefixos: o código de um símbolo não é prefixo do código de nenhum outro símbolo.

Códigos livres de prefixos são fáceis de decodificar.

Código de Huffman

letra	freq	código
a	45	0
b	13	101
c	12	100
d	16	111
e	9	1101
f	5	1100



Código de Huffman

Uma codificação ótima sempre pode ser representada por uma árvore binária **cheia**, na qual cada vértice interno tem exatamente dois filhos.

Assim, podemos dizer que, seja C o alfabeto em questão, então a árvore para um código livre de prefixos ótimo terá $|C|$ folhas e $|C| - 1$ nós internos.

Código de Huffman

O número de bits requeridos para codificar um arquivo é

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c) \Rightarrow \text{ custo da árvore } T$$

onde T é uma árvore binária, $f(c)$ é frequência do caracter c no arquivo e $d_T(c)$ é a profundidade da folha na árvore T.

Código de Huffman

Observando a árvore anterior, vamos calcular o número de bits requeridos para codificar em arquivo contituído pelos caracteres da árvore.

$$
\begin{aligned}
a &= 45 \cdot 1 &&= 45 \\
b &= 13 \cdot 3 &&= 39 \\
c &= 12 \cdot 3 &&= 36 \\
d &= 16 \cdot 3 &&= 48 \\
e &= 9 \cdot 4 &&= 36 \\
f &= 5 \cdot 4 &&= 20
\end{aligned}
$$

Custo total: $B(T) = 254$ bits

Construindo o código de Huffman

algoritmo de Huffman

$\triangleright$ **Entrada:** Conjunto de caracteres C com as frequências f dos caracteres em C.

$\triangleright$ **Saída:** raíz da árvore binária representando uma codificação ótima livre de prefixos.

Construindo o código de Huffman

Huffman(C)

$Q \leftarrow C$
para $i = 1$ **até** $|C| - 1$ **faça**
    $x \leftarrow$ Extrai-Min(Q)
    $y \leftarrow$ Extrai-Min(Q)
    $z \leftarrow$ Alocar-No()
    $esq[z] \leftarrow x$
    $dir[z] \leftarrow y$
    $f[z] \leftarrow f[x] + f[y]$
    Insere(Q, z)
retorne Extrai-min(Q)

Desempenho de Huffman

- $\triangleright$ Q é fila de prioridades
- $\triangleright$ Extrai-min retira de Q um nó q com $f[q]$ mínima.
- $\triangleright$ Tamanho da instância: $n = |C|$
- $\triangleright$ $n - 1$ vezes: Extrai-Min, Extrai-Min, Insere
- $\triangleright$ Cada Extrai-Min e cada Insere consome $O(\lg n)$

total: $O(n \lg n)$

Construindo o código de Huffman

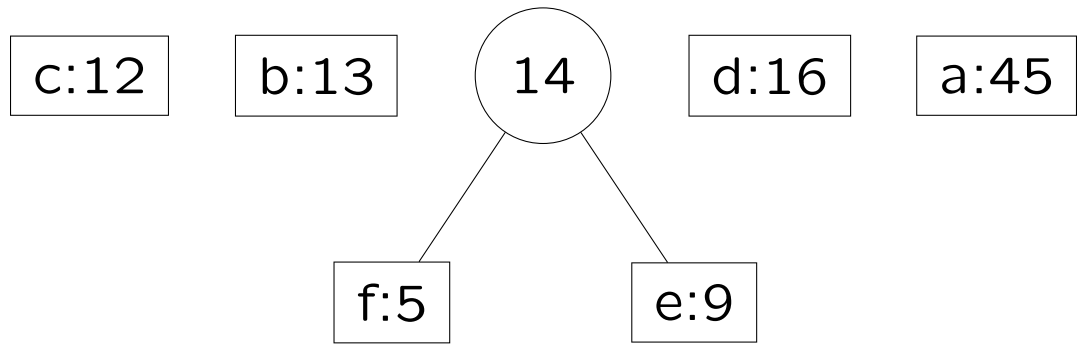
Podemos ilustrar a execução deste algoritmo com o seguinte conjunto de caracteres e suas frequências

letra	freq
a	45
b	13
c	12
d	16
e	9
f	5

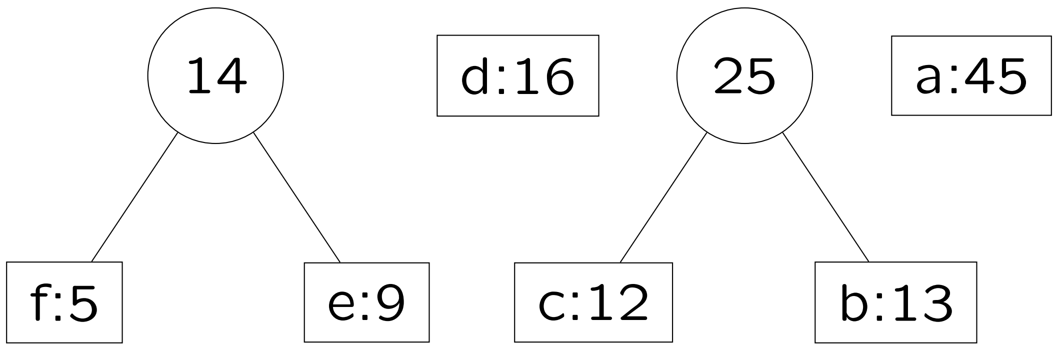
Construindo o código de Huffman (1)

f:5	e:9	c:12	b:13	d:16	a:45

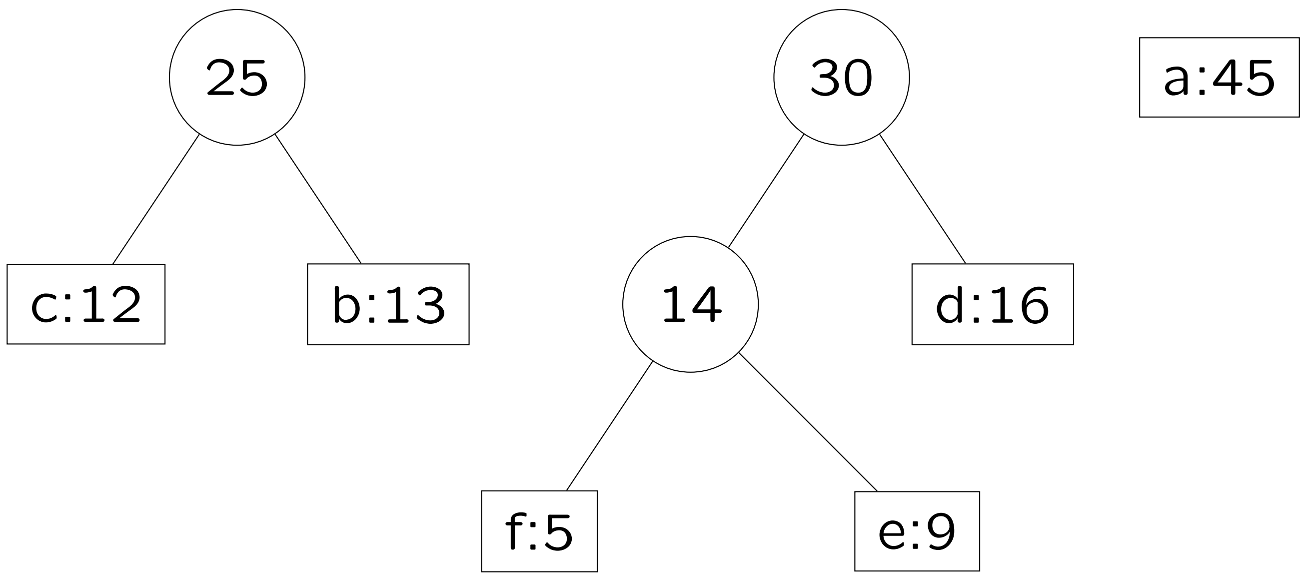
Construindo o código de Huffman (2)



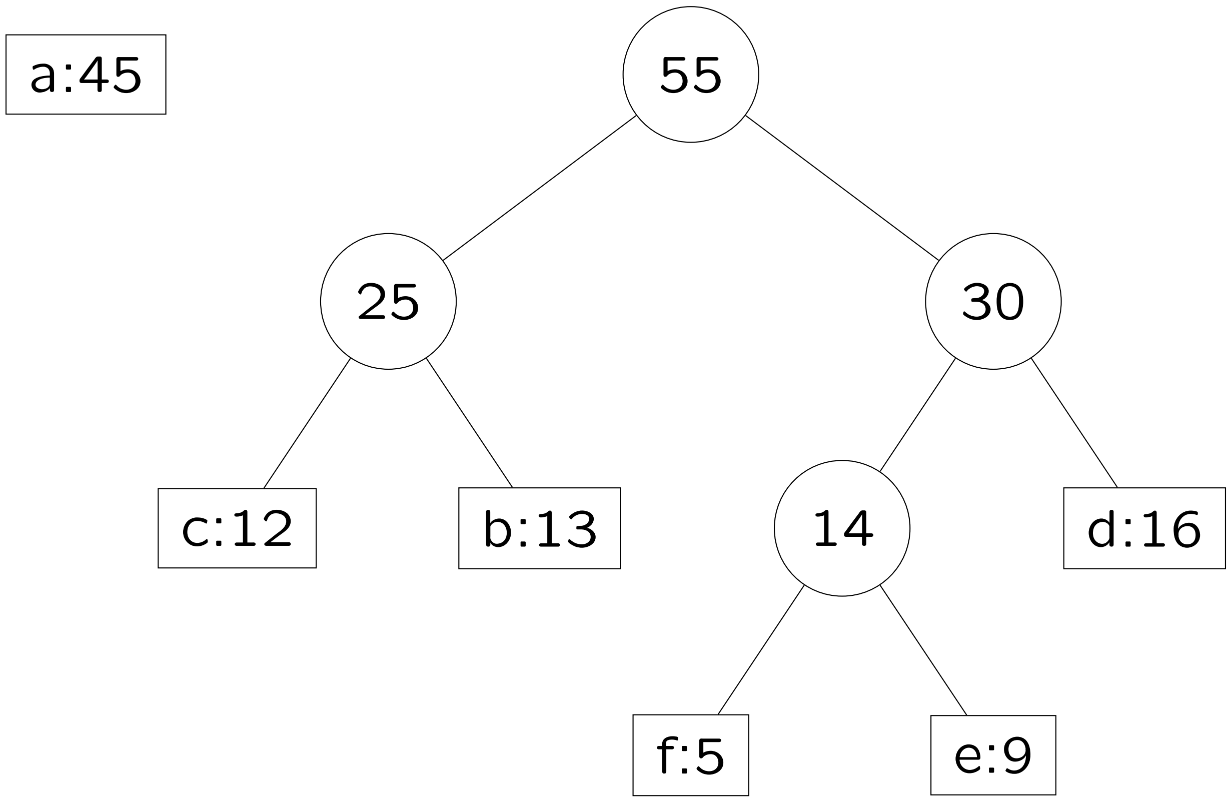
Construindo o código de Huffman (3)



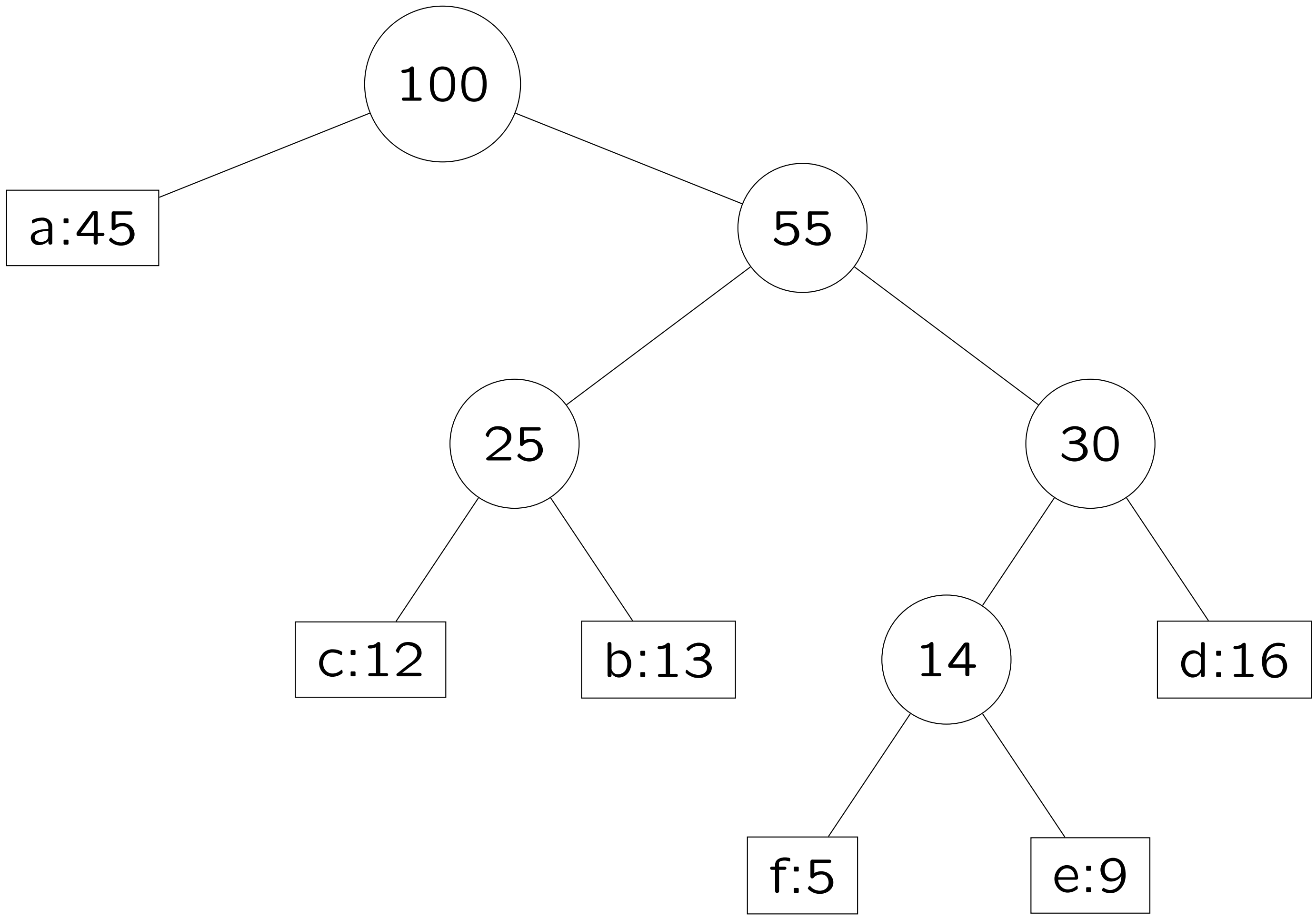
Construindo o código de Huffman (4)



Construindo o código de Huffman (5)



Construindo o código de Huffman (6)



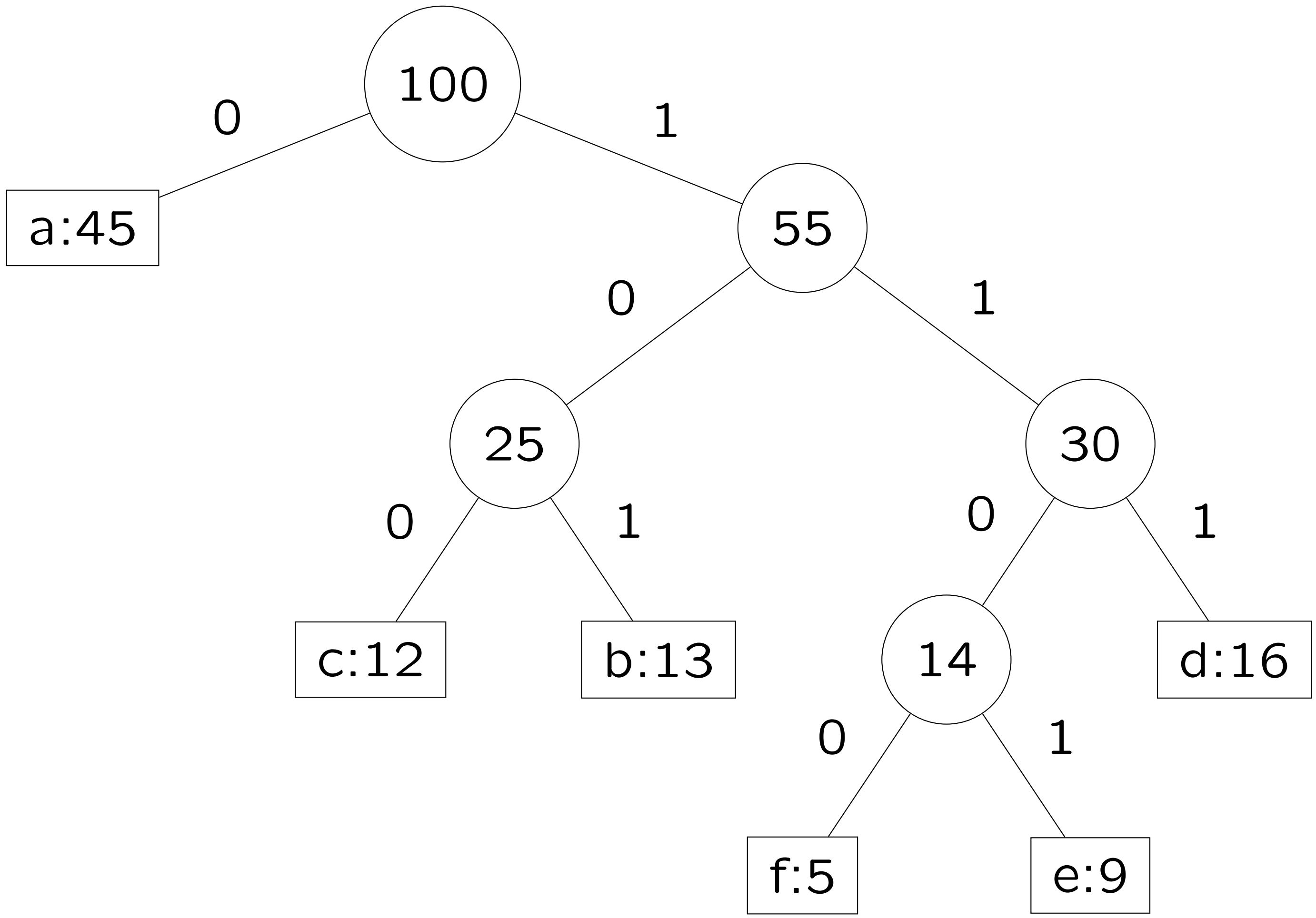
Código de Huffman

Como obter o código a partir da da árvore?

Associe a cada símbolo um código binário assim:

▷ Rotule com 0 as arestas da árvore que ligam um nó com seu filho esquerdo e com 1 as arestas que ligam um nó com seu filho direito.

Código de Huffman



Código de Huffman

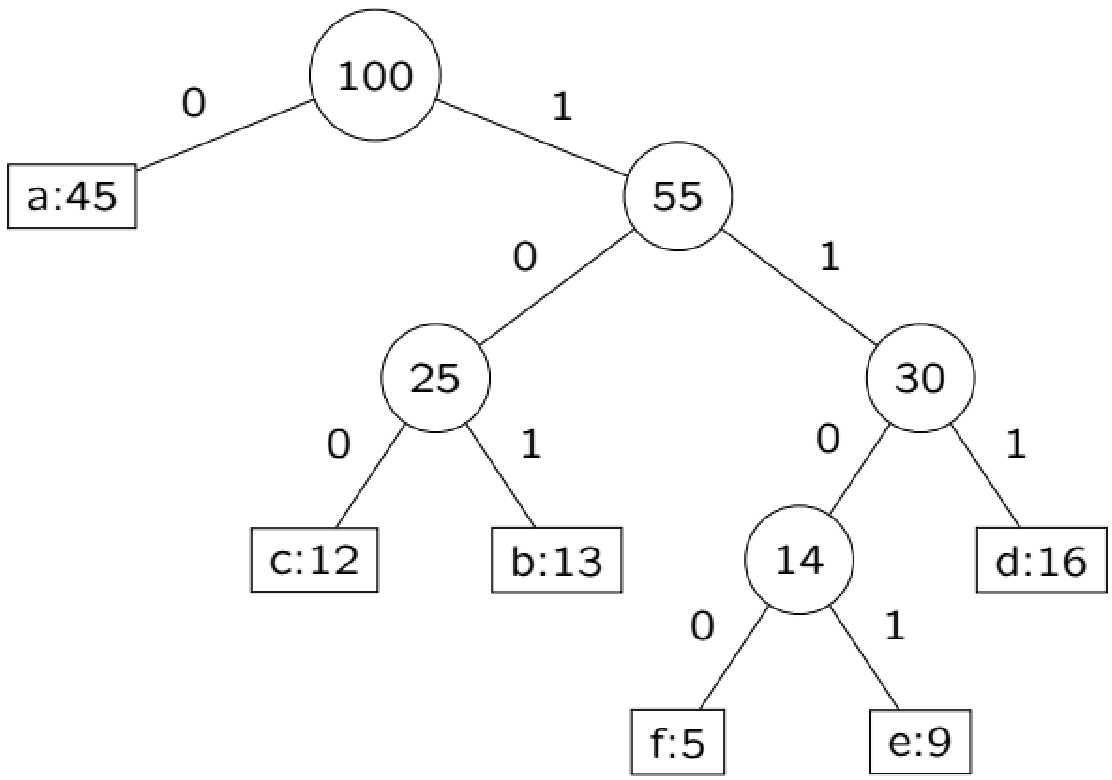
Como obter os códigos a partir da árvore?

Código de Huffman

Como obter os códigos a partir da árvore?

O código correspondente a cada símbolo é a concatenação dos bits associados às arestas do caminho da raiz até a folha correspondente ao símbolo.

Código de Huffman



Exemplo: O código b é 101

Obrigado