

Algoritmos gulosos (*greedy*)

CLRS 16.3-16.5

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \dots, n$

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \dots, n$

t_i : duração da tarefa i

d_i : prazo de entrega da tarefa i

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \dots, n$

t_i : duração da tarefa i

d_i : prazo de entrega da tarefa i

Escalonamento: permutação de 1 a n onde $\pi(i)$ é a tarefa a ser executada na posição i .

Para um escalonamento π , o **tempo de início** da tarefa $\pi(i)$ é

$$s_{\pi(i)} = \sum_{j=1}^{i-1} t_{\pi(j)}$$

(soma da duração das tarefas anteriores a $\pi(i)$).

Problema de escalonamento

Considere n tarefas indicadas pelos números $1, \dots, n$

t_i : duração da tarefa i

d_i : prazo de entrega da tarefa i

Escalonamento: permutação de 1 a n onde $\pi(i)$ é a tarefa a ser executada na posição i .

Para um escalonamento π , o **tempo de início** da tarefa $\pi(i)$ é

$$s_{\pi(i)} = \sum_{j=1}^{i-1} t_{\pi(j)}$$

(soma da duração das tarefas anteriores a $\pi(i)$).

O **tempo de término** da tarefa i é $f_i = s_i + t_i$.

Problema de escalonamento

Para um escalonamento π , o tempo de início da tarefa i é soma da duração das tarefas anteriores a i .

O tempo de término da tarefa i é $f_i = s_i + t_i$.

Problema de escalonamento

Para um escalonamento π , o **tempo de início** da tarefa i é soma da duração das tarefas anteriores a i .

O **tempo de término** da tarefa i é $f_i = s_i + t_i$.

O **atraso** da tarefa i é o número

$$l_i = \begin{cases} 0 & \text{se } f_i \leq d_i \\ f_i - d_i & \text{se } f_i > d_i. \end{cases}$$

Problema de escalonamento

Para um escalonamento π , o tempo de início da tarefa i é soma da duração das tarefas anteriores a i .

O tempo de término da tarefa i é $f_i = s_i + t_i$.

O atraso da tarefa i é o número

$$l_i = \begin{cases} 0 & \text{se } f_i \leq d_i \\ f_i - d_i & \text{se } f_i > d_i. \end{cases}$$

Problema: Dados t_1, \dots, t_n e d_1, \dots, d_n , encontrar um escalonamento com o menor atraso máximo.

Ou seja, que minimize $L = \max_i l_i$.

Exemplo

$$n = 5$$

| | | | | | |
|-------|---|---|---|----|----|
| i | 1 | 2 | 3 | 4 | 5 |
| t_i | 6 | 2 | 1 | 2 | 4 |
| d_i | 8 | 3 | 5 | 12 | 10 |

Exemplo

$$n = 5$$

| | | | | | |
|-------|---|---|---|----|----|
| i | 1 | 2 | 3 | 4 | 5 |
| t_i | 6 | 2 | 1 | 2 | 4 |
| d_i | 8 | 3 | 5 | 12 | 10 |

Para $\pi = (3, 4, 2, 5, 1)$, temos $L = 7$:

| | | | | | |
|----------|----|---|---|---|---|
| i | 1 | 2 | 3 | 4 | 5 |
| f_i | 15 | 5 | 1 | 3 | 9 |
| ℓ_i | 7 | 2 | 0 | 0 | 0 |

Exemplo

$$n = 5$$

| | | | | | |
|-------|---|---|---|----|----|
| i | 1 | 2 | 3 | 4 | 5 |
| t_i | 6 | 2 | 1 | 2 | 4 |
| d_i | 8 | 3 | 5 | 12 | 10 |

Para $\pi = (3, 4, 2, 5, 1)$, temos $L = 7$:

| | | | | | |
|----------|----|---|---|---|---|
| i | 1 | 2 | 3 | 4 | 5 |
| f_i | 15 | 5 | 1 | 3 | 9 |
| ℓ_i | 7 | 2 | 0 | 0 | 0 |

Para $\pi = (2, 3, 1, 5, 4)$, temos $L = 3$:

| | | | | | |
|----------|---|---|---|----|----|
| i | 1 | 2 | 3 | 4 | 5 |
| f_i | 9 | 2 | 3 | 15 | 13 |
| ℓ_i | 1 | 0 | 0 | 3 | 3 |

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \dots \leq d[\pi[n]]$$

2 devolva π

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \dots \leq d[\pi[n]]$$

2 devolva π

Consumo de tempo: $O(n \lg n)$.

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

$$d[\pi[1]] \leq d[\pi[2]] \leq \dots \leq d[\pi[n]]$$

2 devolva π

Consumo de tempo: $O(n \lg n)$.

Na aula, vimos a prova de que este algoritmo está correto (devolve um escalonamento com atraso máximo mínimo).

Ingredientes da prova

Uma **inversão** em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Ingredientes da prova

Uma **inversão** em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Ingredientes da prova

Uma **inversão** em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Mostre que se um escalonamento tem uma inversão, então ele tem uma inversão do tipo $(i, i + 1)$.

Ingredientes da prova

Uma **inversão** em um escalonamento π é um par (i, j) tal que $i < j$ e $d[\pi[i]] > d[\pi[j]]$.

Mostre que dois escalonamentos que não têm inversões têm o mesmo atraso máximo.

Mostre que se um escalonamento tem uma inversão, então ele tem uma inversão do tipo $(i, i + 1)$.

Mostre então que, se trocarmos $\pi[i]$ e $\pi[i + 1]$, obteremos um escalonamento com atraso máximo não superior ao atraso máximo de π .

Códigos de Huffman

Motivação: compressão de textos

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ : alfabeto finito

f_i : frequência de símbolo i em Σ

(coleção de números não-negativos cuja soma é 1)

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ : alfabeto finito

f_i : frequência de símbolo i em Σ

(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo de modo que um texto seja convertido para um arquivo binário o mais compacto possível e seja fácil de decodificar.

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ : alfabeto finito

f_i : frequência de símbolo i em Σ

(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo de modo que um texto seja convertido para um arquivo binário o mais compacto possível e seja fácil de decodificar.

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

Códigos de Huffman

Motivação: compressão de textos

Código ASCII: todo símbolo tem um código de 8 bits.

Σ : alfabeto finito

f_i : frequência de símbolo i em Σ

(coleção de números não-negativos cuja soma é 1)

Objetivo: atribuir um código binário para cada símbolo de modo que um texto seja convertido para um arquivo binário o mais compacto possível e seja fácil de decodificar.

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

Códigos livres de prefixo são fáceis de decodificar.

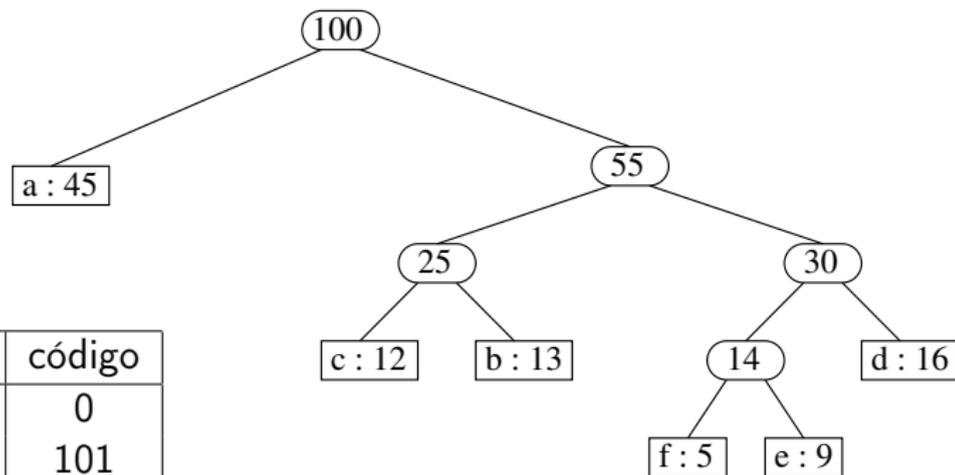
Exemplo

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

Exemplo

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

Exemplo



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

Algoritmo de Huffman

n : número de símbolos em Σ

Algoritmo de Huffman

n : número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

a : 45

d : 16

b : 13

c : 12

e : 9

f : 5

Algoritmo de Huffman

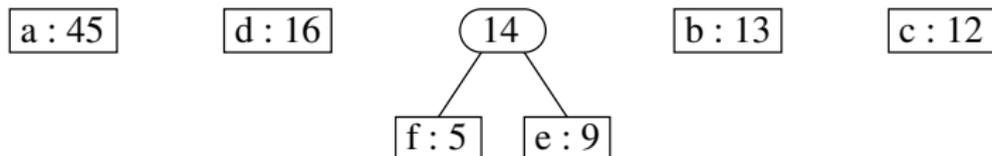
n : número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.



A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.



Algoritmo de Huffman

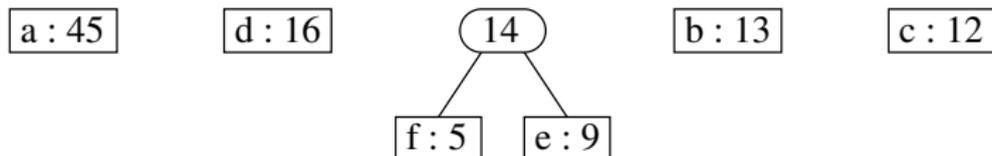
n : número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.



A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.



Pare quando restar uma única árvore.

Algoritmo de Huffman: exemplo

a: 45

d: 16

b: 13

c: 12

e: 9

f: 5

Algoritmo de Huffman: exemplo

a : 45

d : 16

b : 13

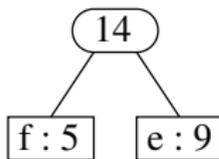
c : 12

e : 9

f : 5

a : 45

d : 16

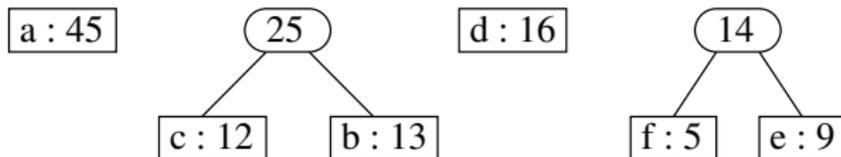
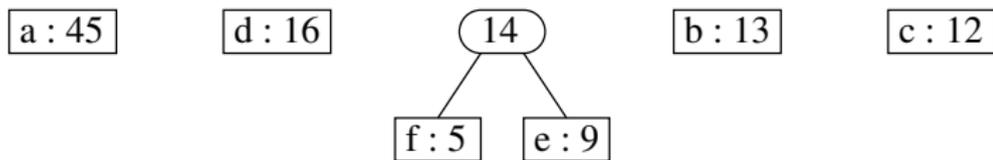


b : 13

c : 12

Algoritmo de Huffman: exemplo

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



Algoritmo de Huffman: exemplo

a : 45

d : 16

b : 13

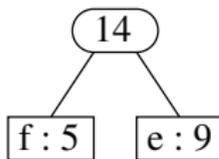
c : 12

e : 9

f : 5

a : 45

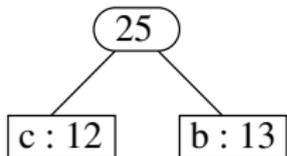
d : 16



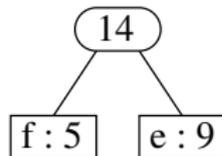
b : 13

c : 12

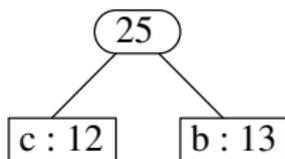
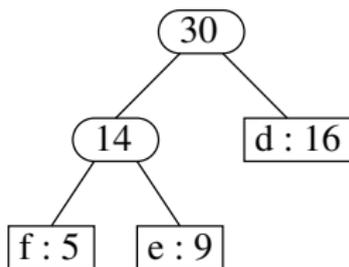
a : 45



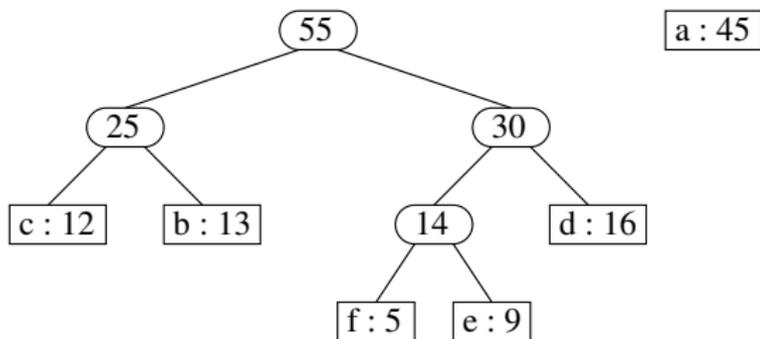
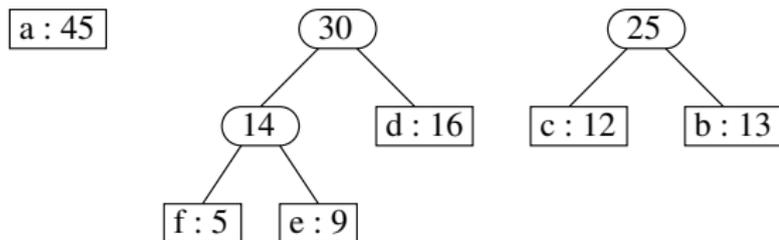
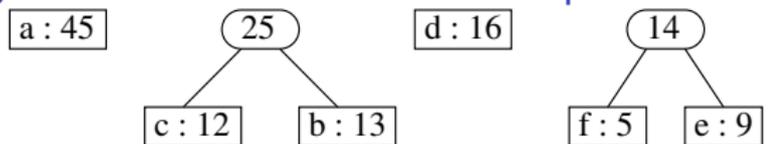
d : 16



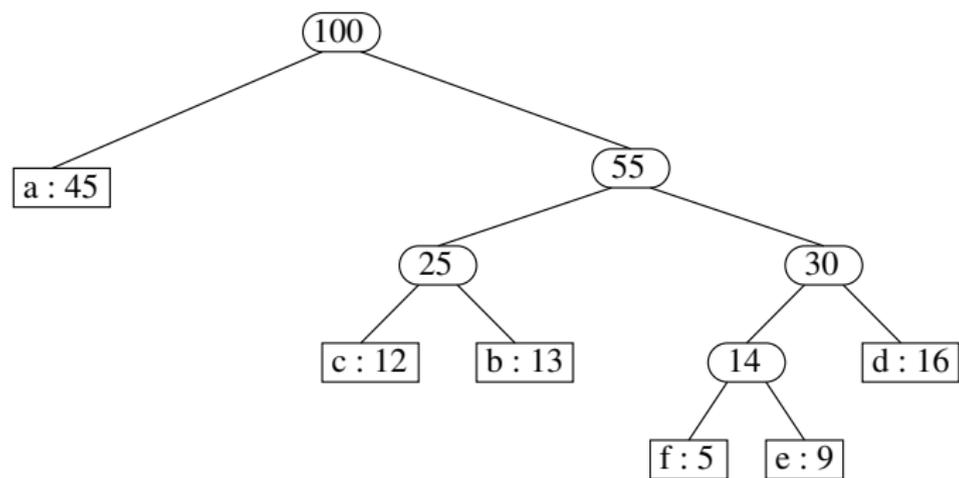
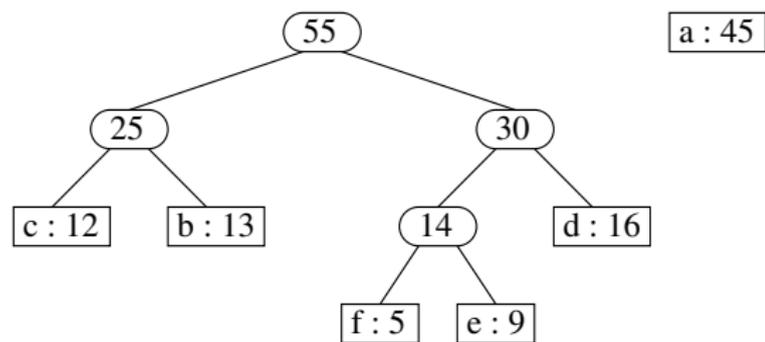
a : 45



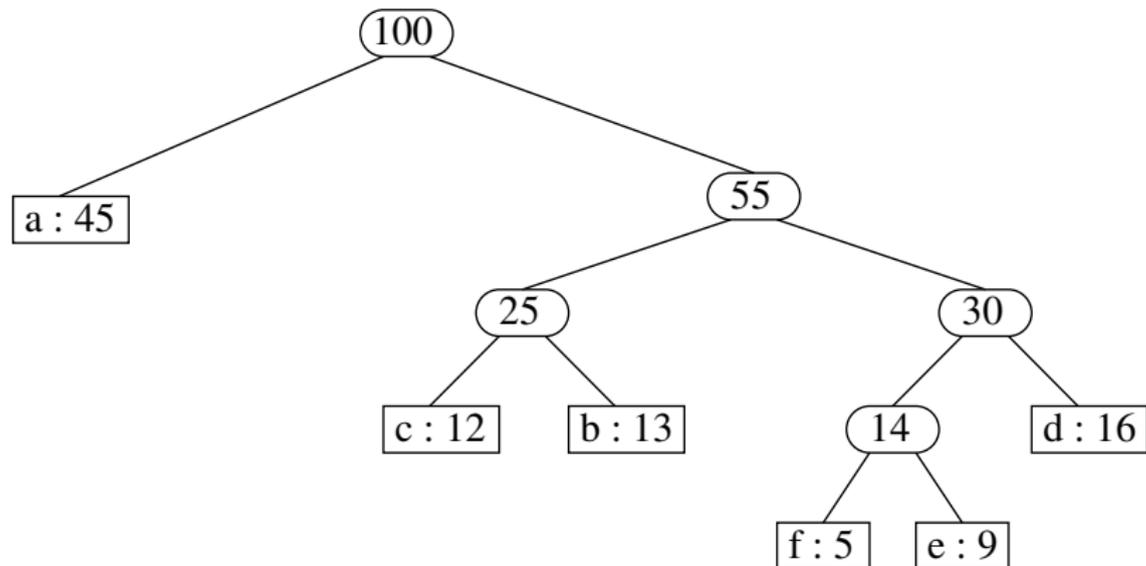
Algoritmo de Huffman: exemplo



Algoritmo de Huffman: exemplo



Árvore de Huffman



Árvore de Huffman

Como obter os códigos a partir da árvore?

Árvore de Huffman

Como obter os códigos a partir da árvore?

Associe a cada símbolo um número binário assim:

Árvore de Huffman

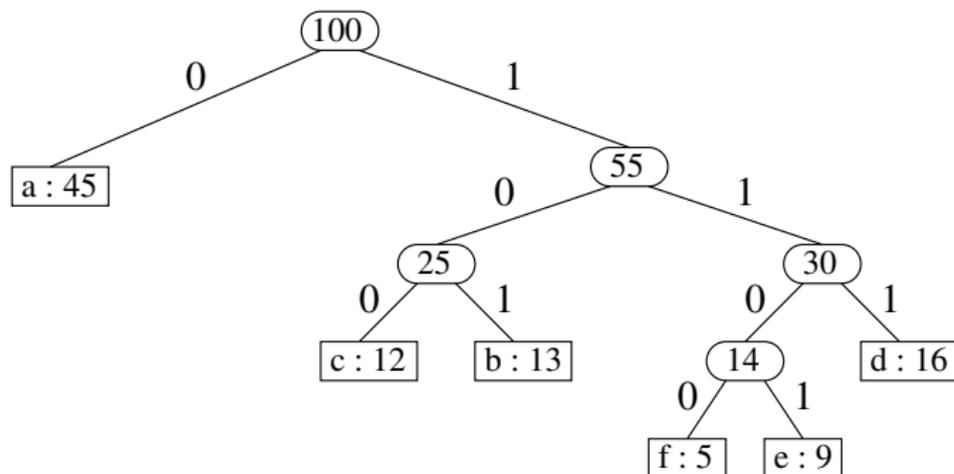
Como obter os códigos a partir da árvore?

Associe a cada símbolo um número binário assim:

Rotule com 0 as arestas da árvore

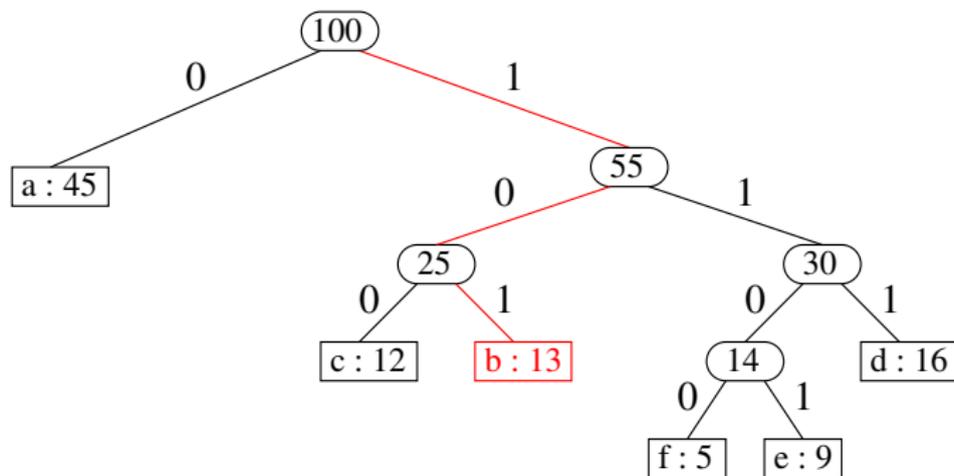
que ligam um nó com seu filho esquerdo e

com 1 as arestas que ligam um nó com seu filho direito.



Árvore 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.

Exemplo: O código de *b* é 101.

Algoritmo de Huffman

n : número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.

Pare quando restar uma única árvore.

Algoritmo de Huffman

n : número de símbolos em Σ

Guloso:

Comece com n árvores disjuntas, cada uma com um único nó, com o símbolo e sua frequência.

A cada iteração, escolha as duas árvores de frequência menor e junte-as, com frequência somada.

Pare quando restar uma única árvore.

Perguntas:

- ▶ Este algoritmo produz um código ótimo?
- ▶ Como implementá-lo do modo mais eficiente possível?

Algoritmo guloso

HUFFMAN (A, f, n)

```
1   $Q \leftarrow \text{BUILD-MIN-HEAP}(A, f, n)$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3       $x \leftarrow \text{EXTRACT-MIN}(Q)$ 
4       $y \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z \leftarrow \text{NOVA-CEL}()$ 
6       $\text{esq}[z] \leftarrow x$ 
7       $\text{esq}[z] \leftarrow y$ 
8       $f[z] \leftarrow f[x] + f[y]$ 
9       $\text{INSEREHEAP}(Q, z, f[z])$ 
10 devolva  $\text{EXTRACT-MIN}(Q)$ 
```

Algoritmo guloso

HUFFMAN (A, f, n)

```
1   $Q \leftarrow \text{BUILD-MIN-HEAP}(A, f, n)$ 
2  para  $i \leftarrow 1$  até  $n - 1$  faça
3       $x \leftarrow \text{EXTRACT-MIN}(Q)$ 
4       $y \leftarrow \text{EXTRACT-MIN}(Q)$ 
5       $z \leftarrow \text{NOVA-CEL}()$ 
6       $\text{esq}[z] \leftarrow x$ 
7       $\text{esq}[z] \leftarrow y$ 
8       $f[z] \leftarrow f[x] + f[y]$ 
9       $\text{INSEREHEAP}(Q, z, f[z])$ 
10 devolva  $\text{EXTRACT-MIN}(Q)$ 
```

Consumo de tempo: $O(n \lg n)$.