Problema de escalonamento

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

Problema de escalonamento

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

t_i: duração da tarefa i

d_i: prazo de entrega da tarefa i

Problema de escalonamento

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

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 posição em que a tarefa i é executada.

Para um escalonamento π, o tempo de início da tarefa i é

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

(soma da duração das tarefas anteriores a i).

Problema de escalonamento

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

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 posição em que a tarefa i é executada.

Para um escalonamento π, o tempo de início da tarefa i é

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

(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 $f_i = s_i + t_i$.

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

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

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

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

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

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

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

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

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

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

2 **devolva** π

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

Algoritmo guloso

Devolve escalonamento com atraso máximo mínimo.

ESCALONAMETO (t, d, n)

1 seja π a permutação de 1 a n tal que

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

2 **devolva** π

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

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 ótimo 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 ótimo 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 escalomento 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.

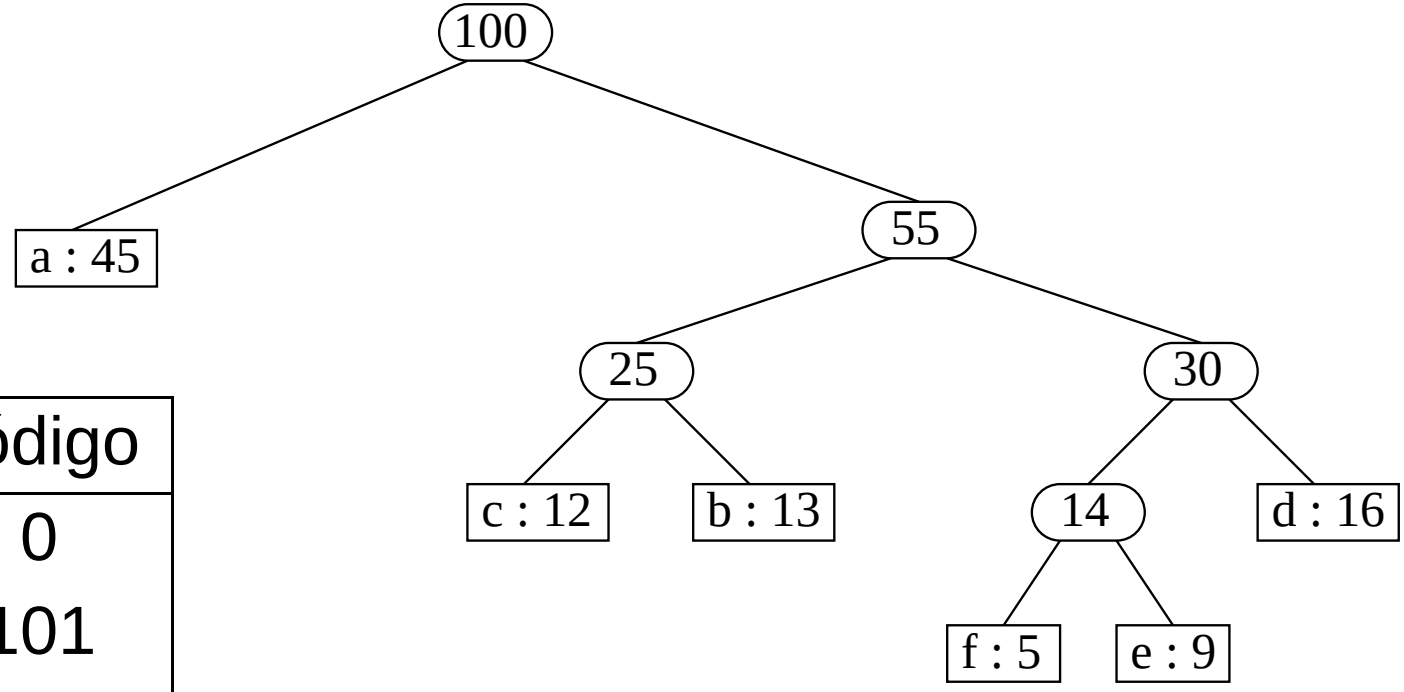
Exemplo

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

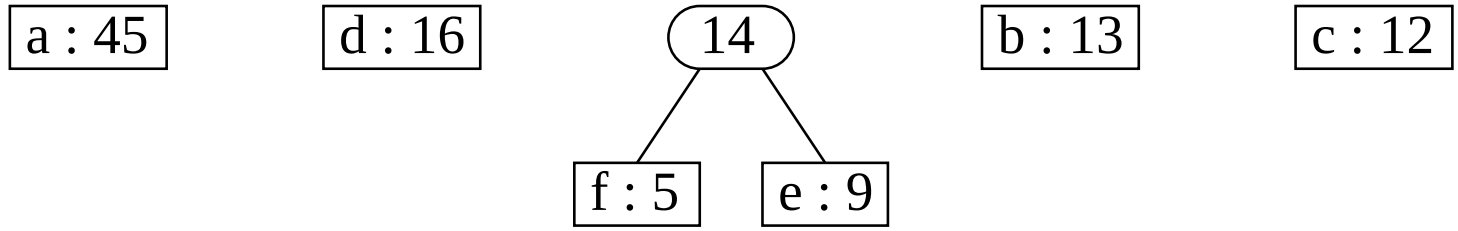
Algoritmo de Huffman

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.



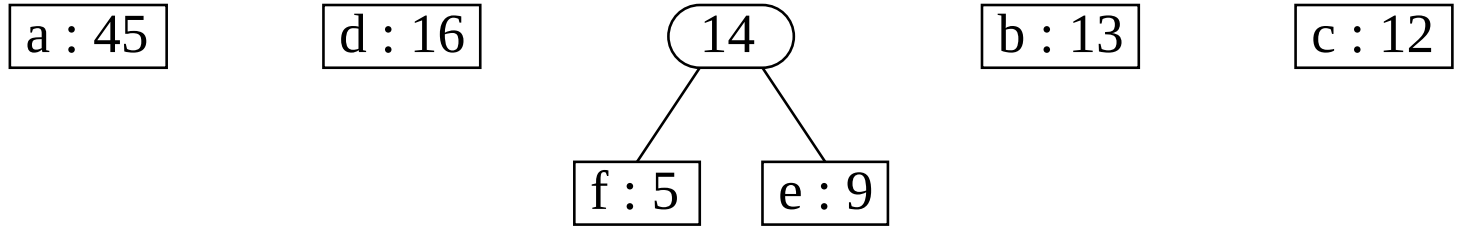
Algoritmo de Huffman

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.

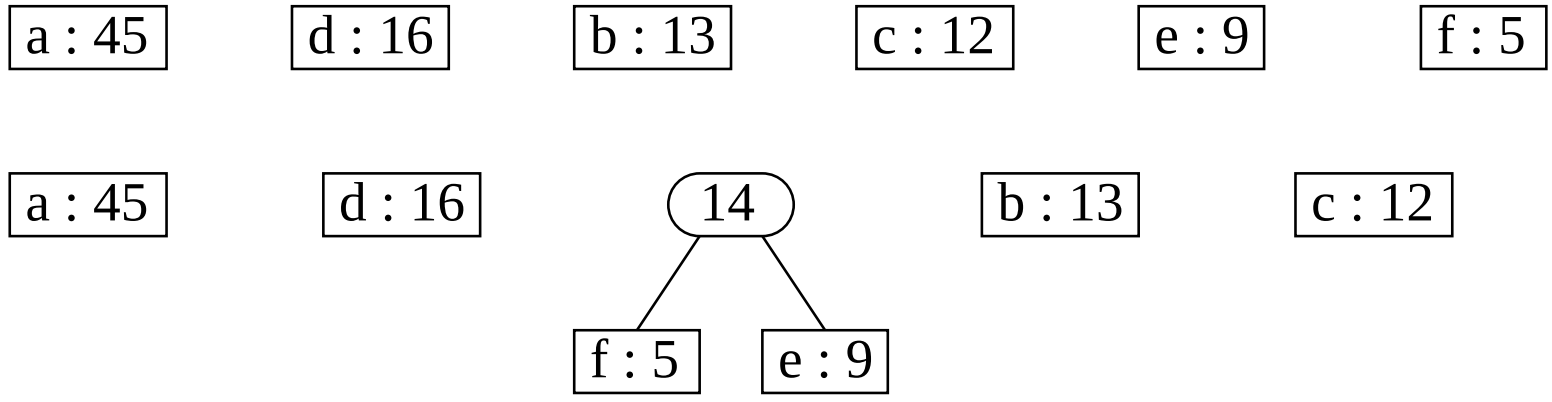


Pare quando restar uma única árvore.

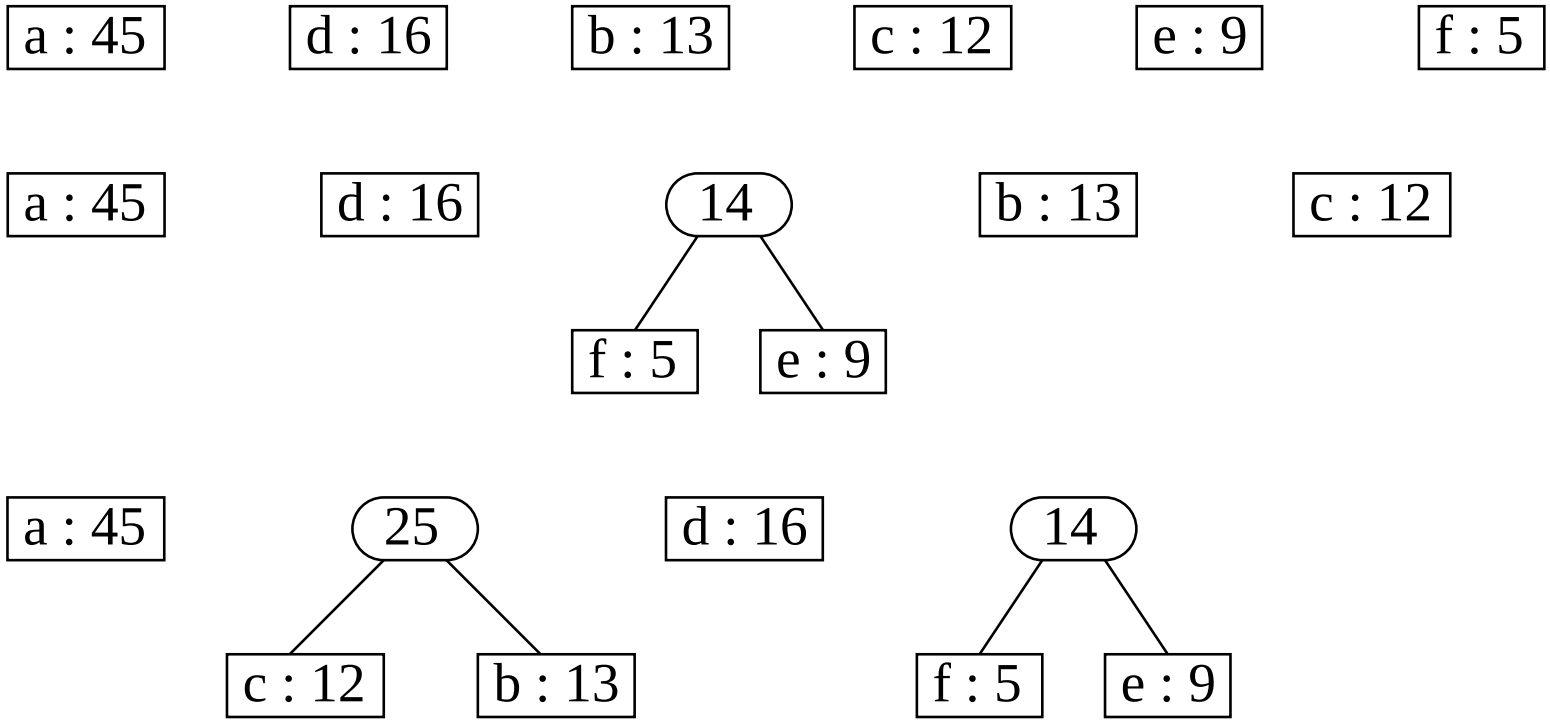
Algoritmo de Huffman: exemplo

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

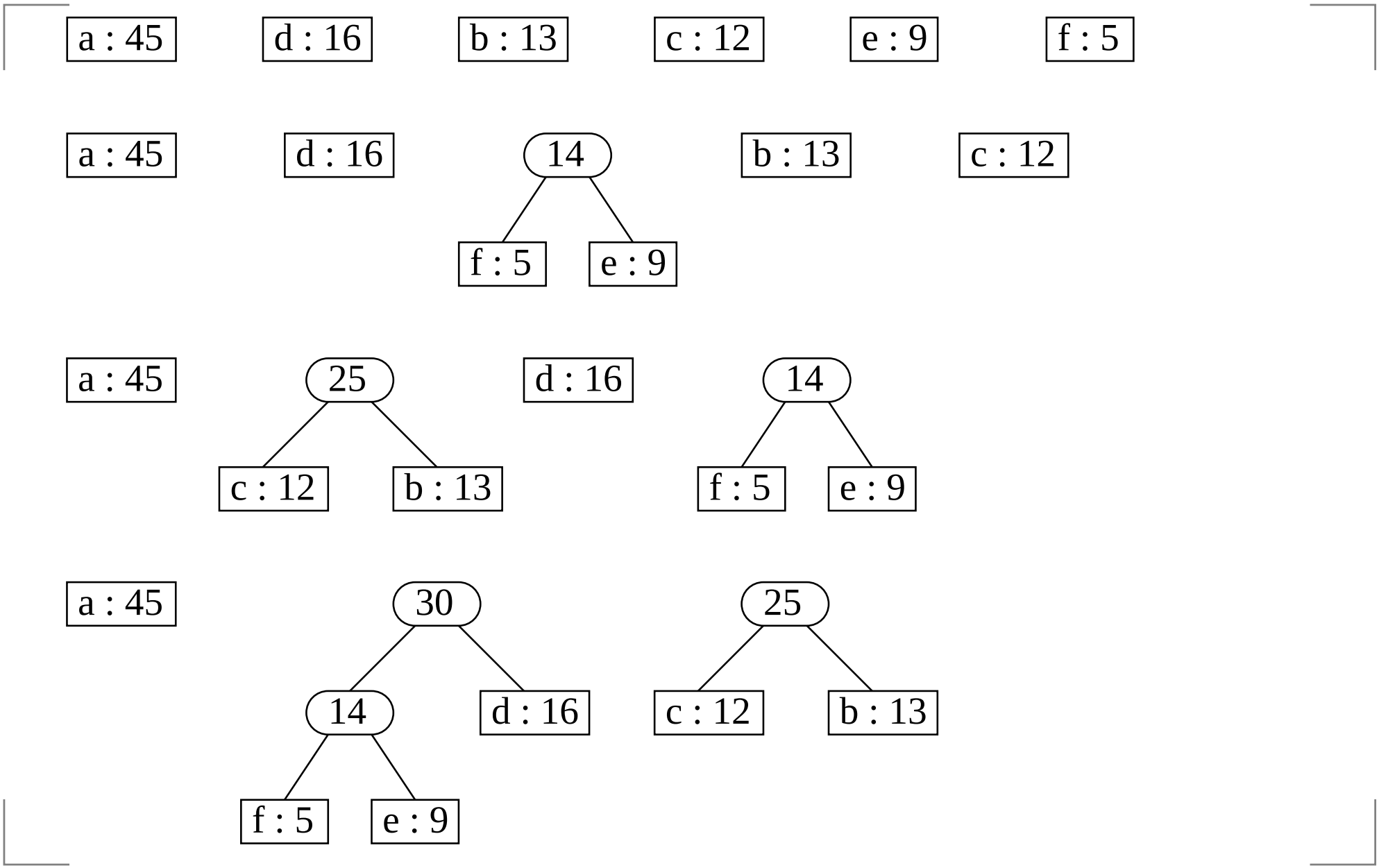
Algoritmo de Huffman: exemplo



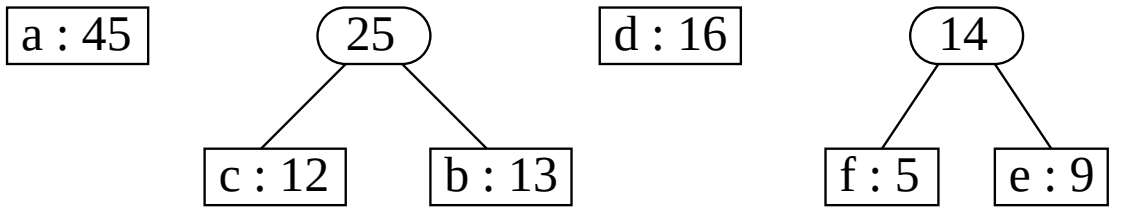
Algoritmo de Huffman: exemplo

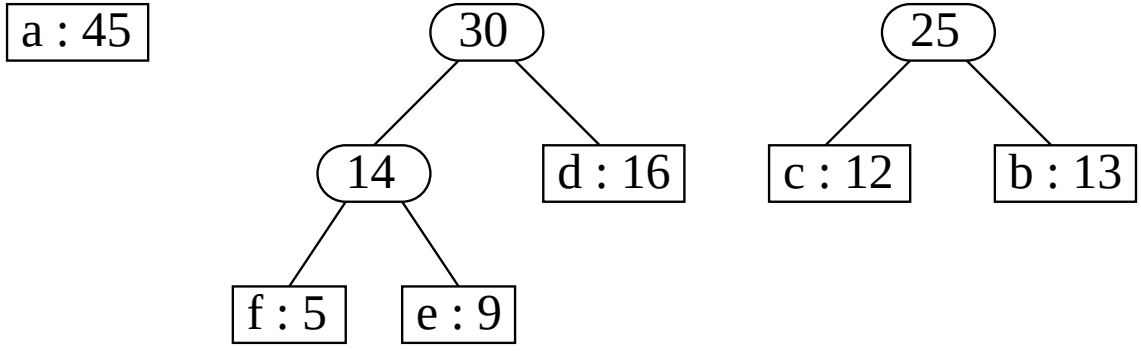


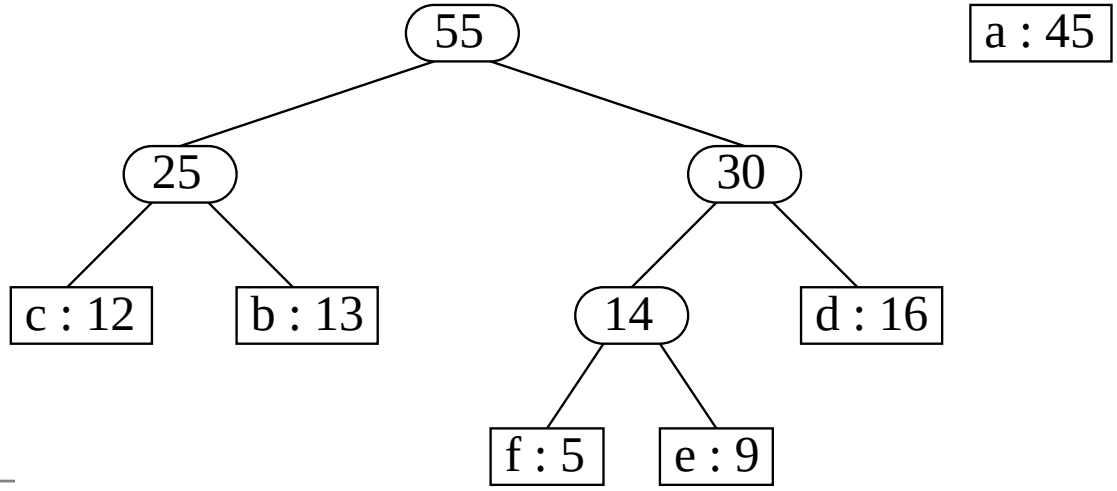
Algoritmo de Huffman: exemplo



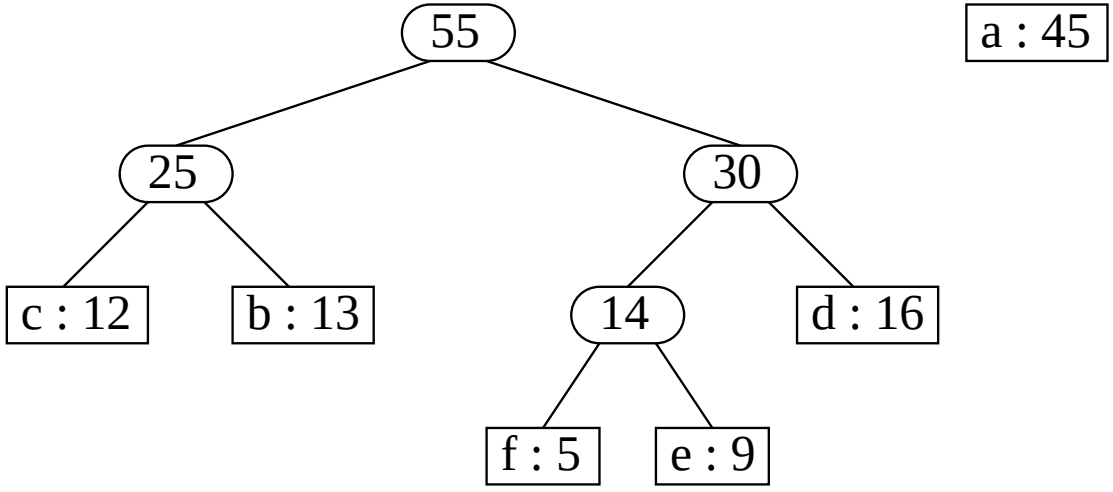
Algoritmo de Huffman: exemplo



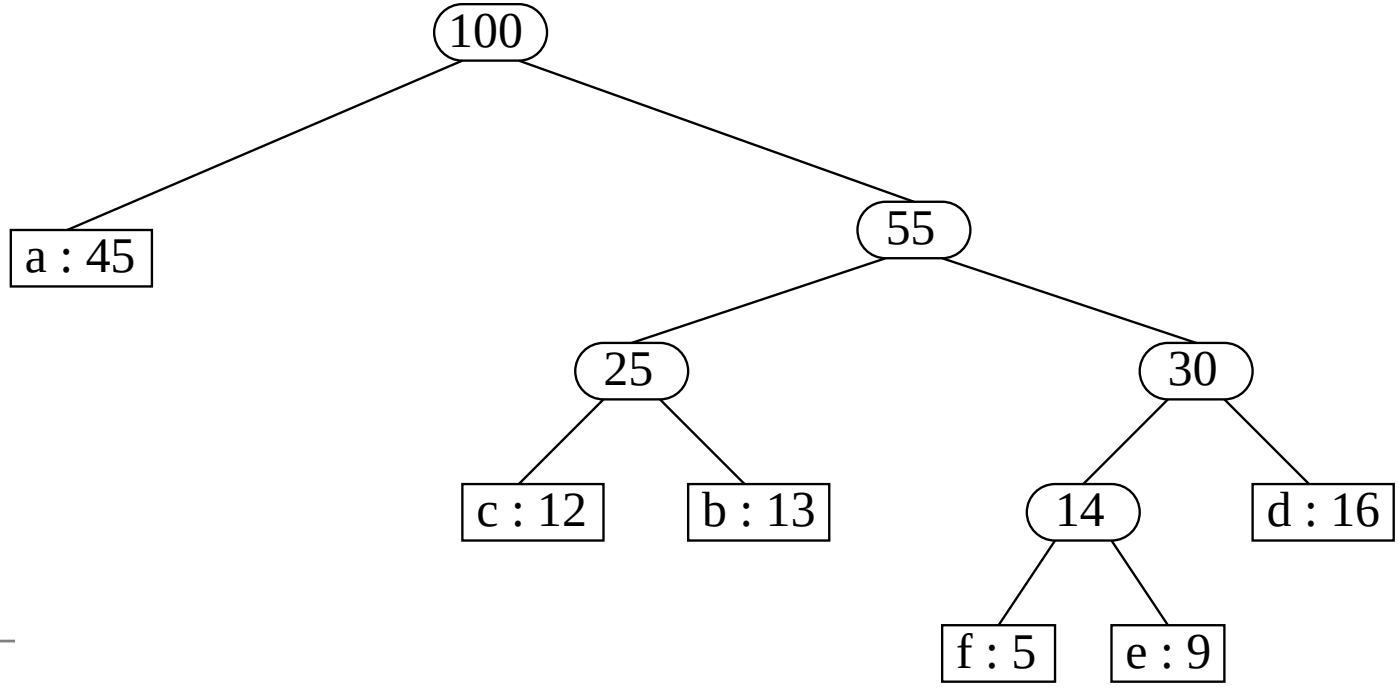




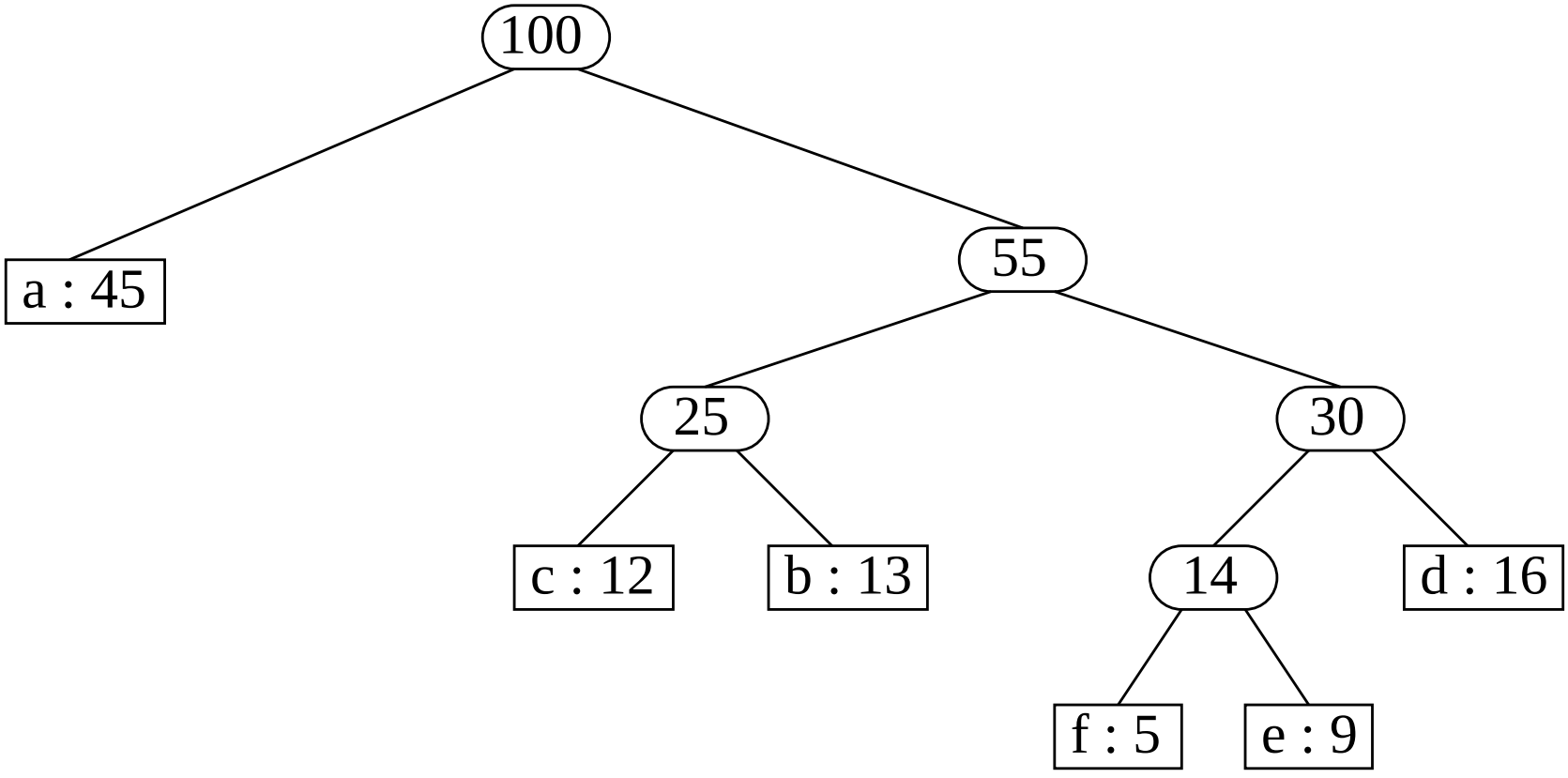
Algoritmo de Huffman: exemplo



a : 45



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?

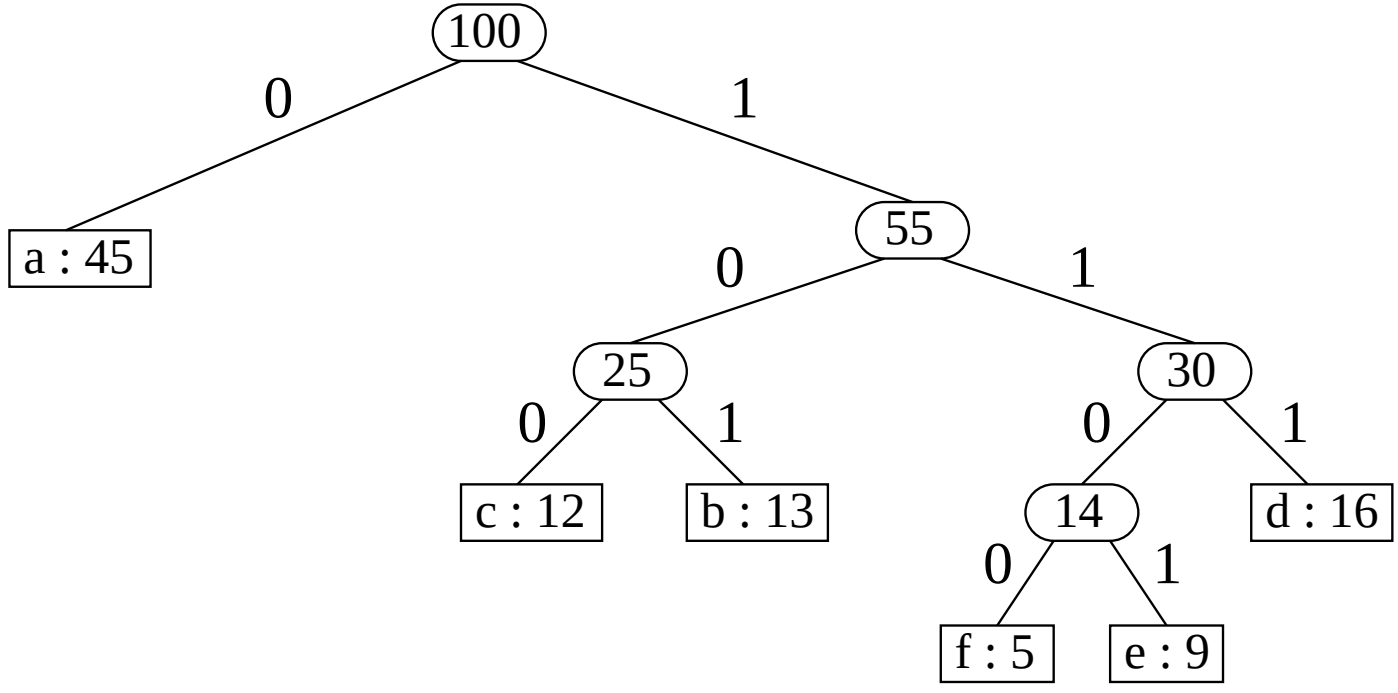
Associe a cada símbolo um número binário assim:

Árvore de Huffman

Como obter os códigos a partir da árvore?

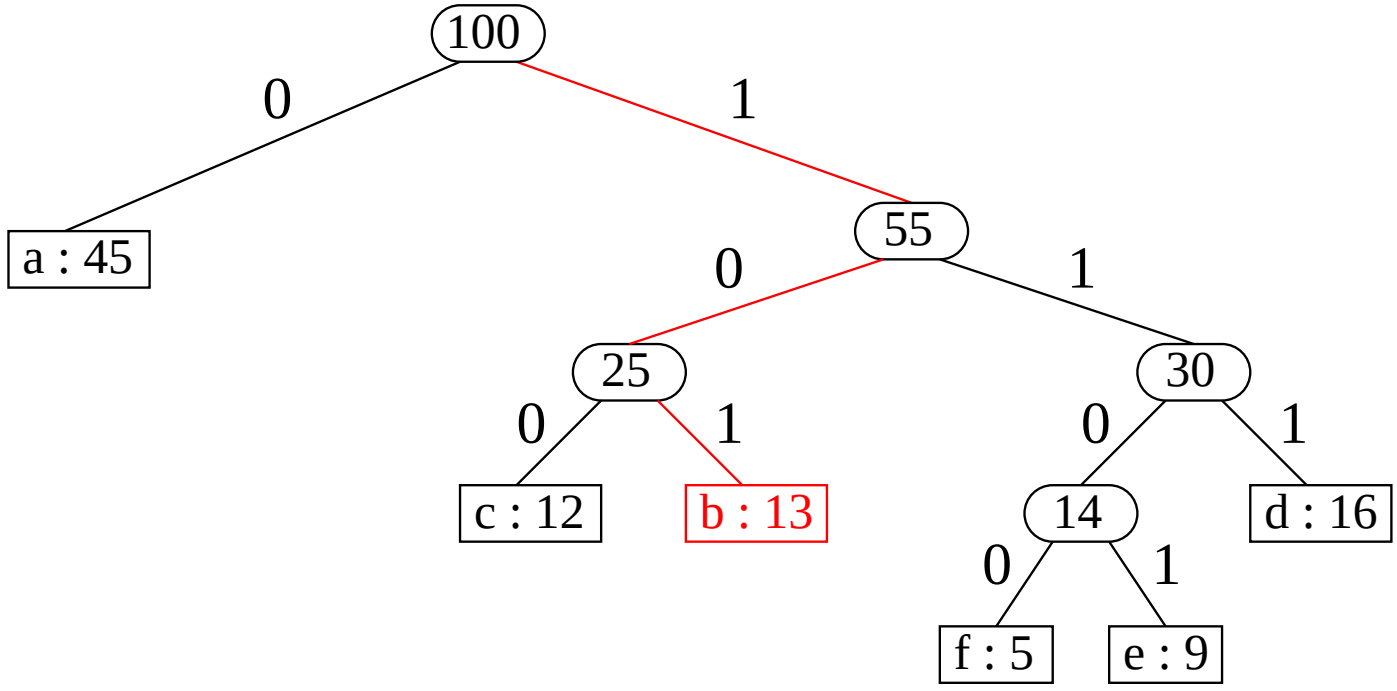
Associe a cada símbolo um número binário assim:

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

Algoritmo guloso

```
HUFFMAN (A, f, n)
 1  Q ← BUILD-MIN-HEAP(A, f, n)
 2  para i ← 1 até n − 1 faça
 3      x ← EXTRACT-MIN(Q)
 4      y ← EXTRACT-MIN(Q)
 5      z ← NOVA-CEL()
 6      esq[z] ← x
 7      dir[z] ← y
 8      f[z] ← f[x] + f[y]
 9      INSEREHEAP(Q, z, f[z])
10  devolva EXTRACT-MIN(Q)
```

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