

**Università di Pisa**

**Pietro Ducange**

**Algoritmi e strutture dati**  
**Programmazione dinamica e algoritmi greedy**

**a.a. 2020/2021**

**Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione  
la maggior parte delle slide utilizzate nella presente lezione**

## Programmazione dinamica

**Si può usare quando non è possibile applicare il metodo del divide et impera (non si sa con esattezza quali sottoproblemi risolvere e non è possibile partizionare l'insieme in sottoinsiemi disgiunti)**

**Metodo: si risolvono **tutti** i sottoproblemi a partire dal basso e si conservano i risultati ottenuti per poterli usare successivamente. (strategia bottom-up)**

**La complessità del metodo dipende dal numero dei sottoproblemi**

## Quando si può applicare

**sottostruttura ottima:** una soluzione ottima del problema contiene la soluzione ottima dei sottoproblemi

**sottoproblemi comuni :** un algoritmo ricorsivo richiederebbe di risolvere lo stesso sottoproblema più volte

## Più lunga sottosequenza comune (PLSC)

$\alpha = \text{a b c a b b a}$      $\beta = \text{c b a b a c}$

$\alpha = \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7$

$\beta = \beta_1 \beta_2 \beta_3 \beta_4 \beta_5 \beta_6$

**3 PLSC:** **baba, cbba, caba**

**Lunghezza delle PLSC = 4**

## PLSC

$$\alpha = \alpha_1 \dots \alpha_i \dots \alpha_m \quad \beta = \beta_1 \dots \beta_j \dots \beta_n$$

**$L(i,j)$  = lunghezza delle PLSC di  $\alpha_1 \dots \alpha_i$  e  $\beta_1 \dots \beta_j$**

$$L(0,0)=L(i,0)=L(0,j)=0$$

$$L(i,j)=L(i-1,j-1)+1 \quad \text{se } \alpha_i = \beta_j$$

$$L(i,j)=\max(L(i,j-1),L(i-1,j)) \quad \text{se } \alpha_i \neq \beta_j$$

```
int length(char* a, char* b, int i, int j) {  
    if (i==0 || j==0) return 0;  
    if (a[i]==b[j]) return length(a, b, i-1, j-1)+1;  
    else  
        return max(length(a,b,i,j-1),length(a,b,i-1,j));  
};
```

La funzione ha un tempo esponenziale nel più piccolo fra  $n$  e  $m$ .

**Costruisce tutti gli  $L(i,j)$  a partire dagli  
indici più piccoli (bottom-up):**

**$L(0,0), L(0,1) \dots L(0,n),$**

**$L(1,0), L(1,1) \dots L(1,n),$**

**$\dots$**

**$L(m,0), L(m,1) \dots L(m,n)$**



## Algoritmo di programmazione dinamica

```
const int m=7; const int n=6;
int L [m+1][n+1];
int quickLength(char *a, char *b) {
    for (int j=0; j<=n; j++) L[ 0 ][ j ]=0; // prima riga

    for (int i=1; i<=m; i++) {           // i-esima riga
        L[ i ][ 0 ]=0;
        for (j=1; j<=n; j++)
            if (a[ i] != b[ j])
                L[ i ][ j ] = max(L[ i ][ j-1 ],L[ i-1 ][ j ]);
            else L[ i ][ j ]=L[ i-1 ][ j-1 ]+1;
    }
    return L[ m ][ n ];
}
```

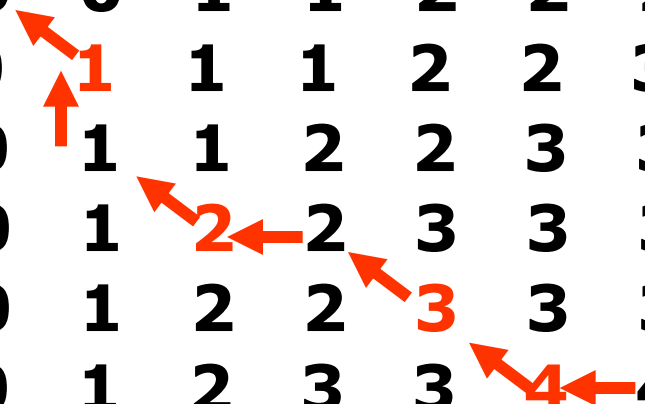
**Complessità?**

# PLSC

		<b>c</b>	<b>b</b>	<b>a</b>	<b>b</b>	<b>a</b>	<b>c</b>
	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>a</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>b</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>2</b>
<b>c</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>3</b>
<b>a</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>
<b>b</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>3</b>
<b>b</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>3</b>
<b>a</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>3</b>	<b>4</b>	<b>4</b>

# PLSC

		c	b	a	b	a	c
	0	0	0	0	0	0	0
a	0	0	0	1	1	1	1
b	0	0	1	1	2	2	2
c	0	1	1	1	2	2	3
a	0	1	1	2	2	3	3
b	0	1	2	2	3	3	3
b	0	1	2	2	3	3	3
a	0	1	2	3	3	4	4



cbba

```
void print(int ** L, char *a, char *b, int i=m, int j=n){  
    if ((i==0) || (j==0) ) return;  
    if (a[i]==b[j]) {  
        print(a,b, i-1, j-1);  
        cout << a[i];  
    }  
    else if (L[i][j] == L[i-1][j])  
        print(a,b, i-1, j);  
    else print(a,b, i, j-1);  
}
```

## Riferimenti Bibliografici

Demetrescu:

Capitolo 10.2

Cormen:

Capitolo 15

## Algoritmi greedy (golosi)

**la soluzione ottima si ottiene mediante una sequenza di scelte**

**In ogni punto dell'algoritmo, viene scelta la strada che in quel momento sembra la migliore**

**la scelta locale deve essere in accordo con la scelta globale: scegliendo ad ogni passo l'alternativa che sembra la migliore non si dovrebbero perdere alternative che potrebbero rivelarsi migliori nel seguito.**

### Metodo **top-down**

**Non sempre si trova la soluzione **ottima** ma in certi casi si può trovare una soluzione approssimata (esempio del problema dello zaino)**

**Alfabeto** : insieme di caratteri (es: a, b, c, d, e, f)

**Codice binario**: assegna ad ogni carattere una stringa binaria

**Codifica del testo**: sostituisce ad ogni carattere del testo il corrispondente codice binario.

**Decodifica**: ricostruire il testo originario.

**Il codice può essere a lunghezza fissa o a lunghezza variabile**



## codici di compressione

	a	b	c	d	e	f
frequenza	45	13	12	16	9	5
Codice a lunghezza fissa	000	001	010	011	100	101
Codice a lunghezza variabile	0	101	100	111	1101	1100

## codici

Codifica di **abc** con codice a lunghezza fissa: :

**000 001 010 (9 bit)**

Codifica di **abc** con codice a lunghezza variabile :

**0 101 100 (7 bit)**

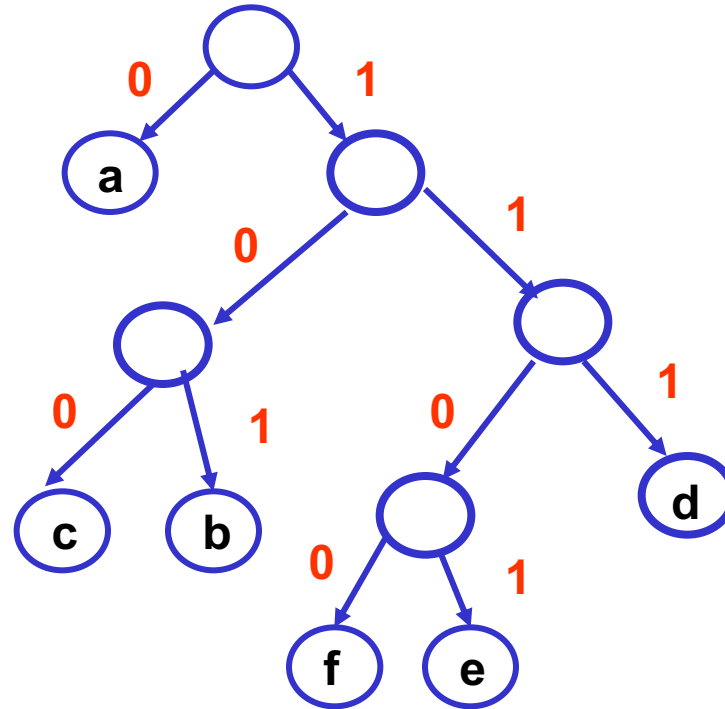
**Problema della decodifica**

**Codice prefisso: nessun codice può essere il prefisso di un altro codice**

## codici prefissi

**I codici prefissi  
possono essere  
rappresentati con  
alberi binari**

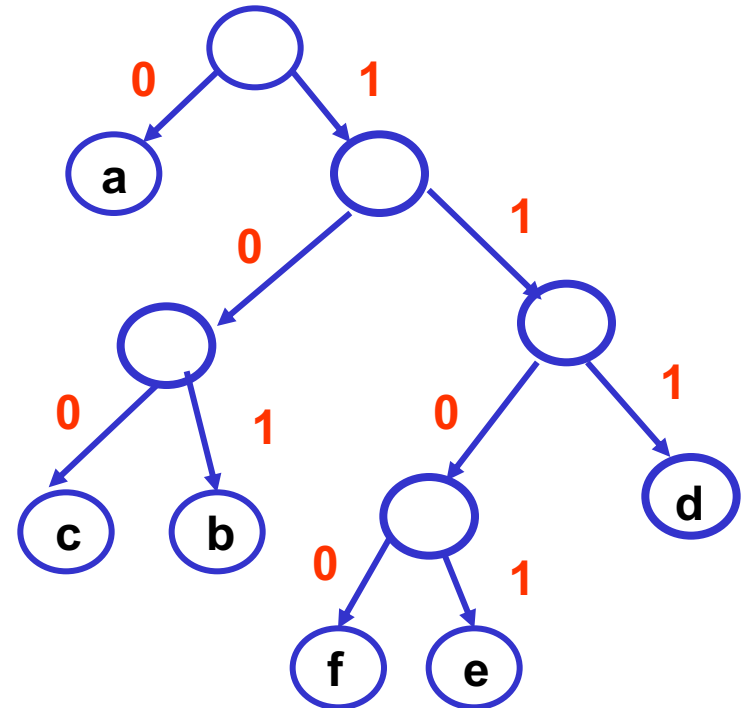
**Rappresentazione  
ottima: albero  
pienamente binario**



a	b	c	d	e	f
0	101	100	111	1101	1100

**L'albero ha tante **foglie** quanti sono i caratteri dell'alfabeto**

**L'algoritmo di decodifica trova un cammino dalla radice ad una foglia per ogni carattere riconosciuto**



**Problema:** dato un alfabeto e la frequenza dei suoi caratteri, costruire un codice ottimo (che minimizza la lunghezza in bit delle codifiche)

### Algoritmo di Huffman

Costruisce l'albero binario in modo bottom-up

È un algoritmo **greedy**

## algoritmo di Huffman

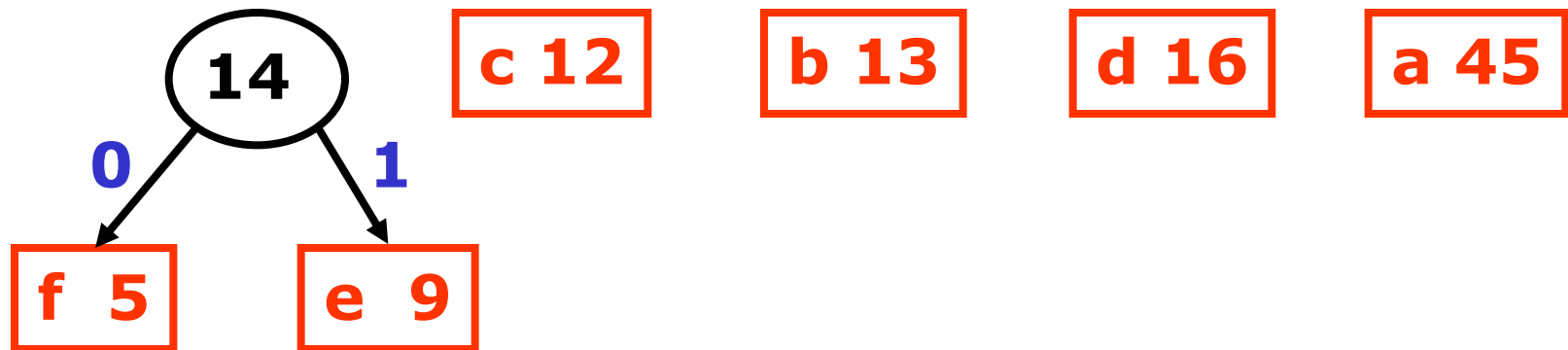
**Gestisce un foresta di alberi**

**All'inizio ci sono  $n$  alberi di un solo nodo con le frequenze dei caratteri.**

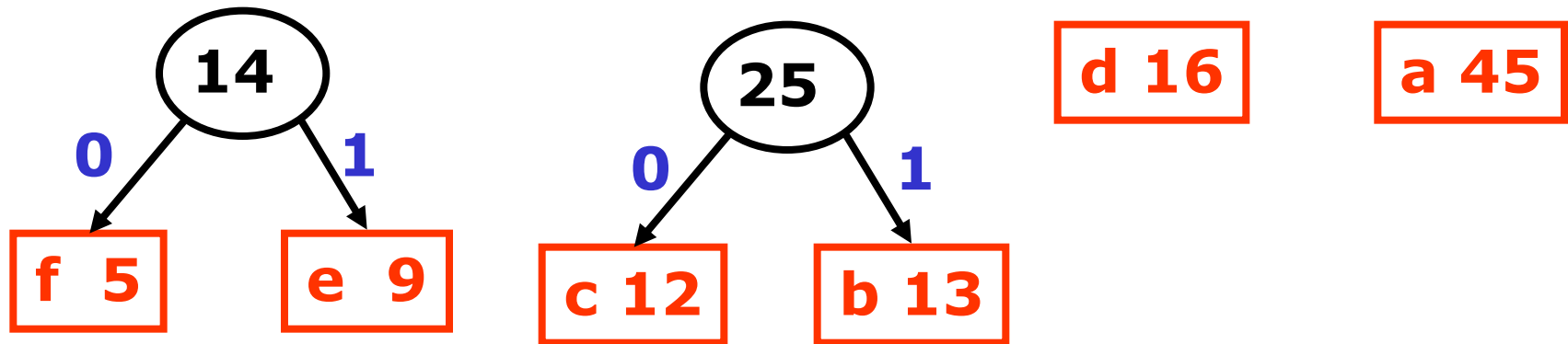
**Ad ogni passo**

- **vengono fusi i due alberi con *radice minore* introducendo una *nuova radice* avente come etichetta la somma delle due *radici***

## algoritmo di Huffman

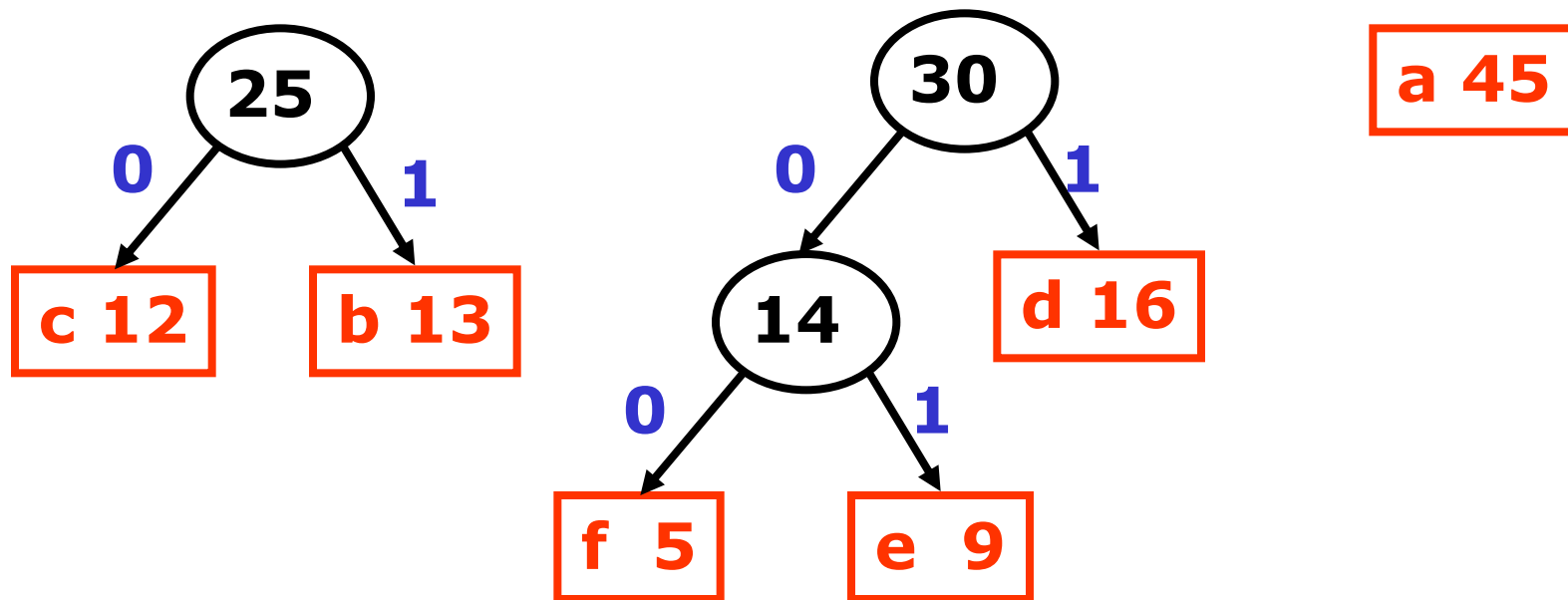


## algoritmo di Huffman

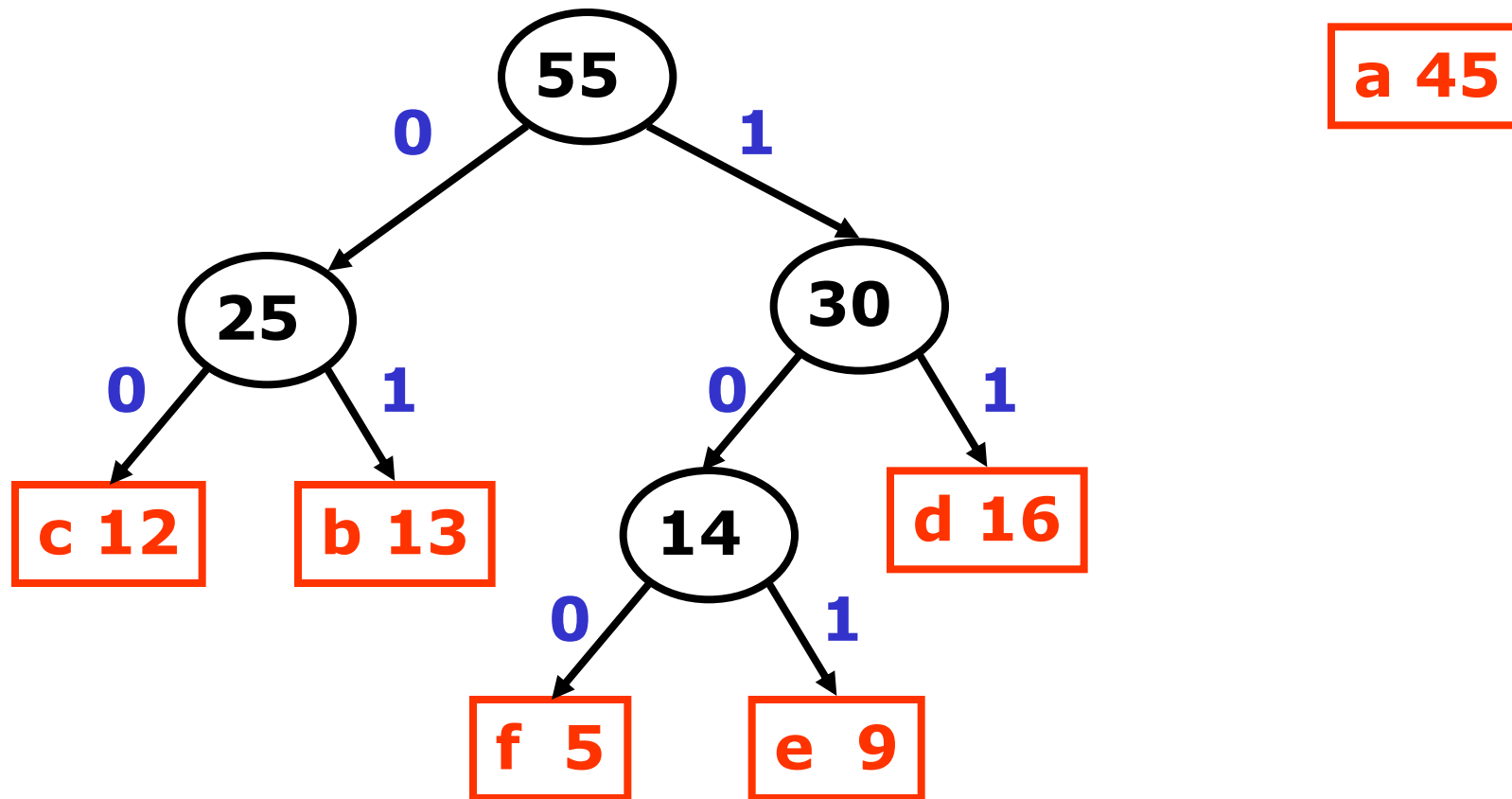




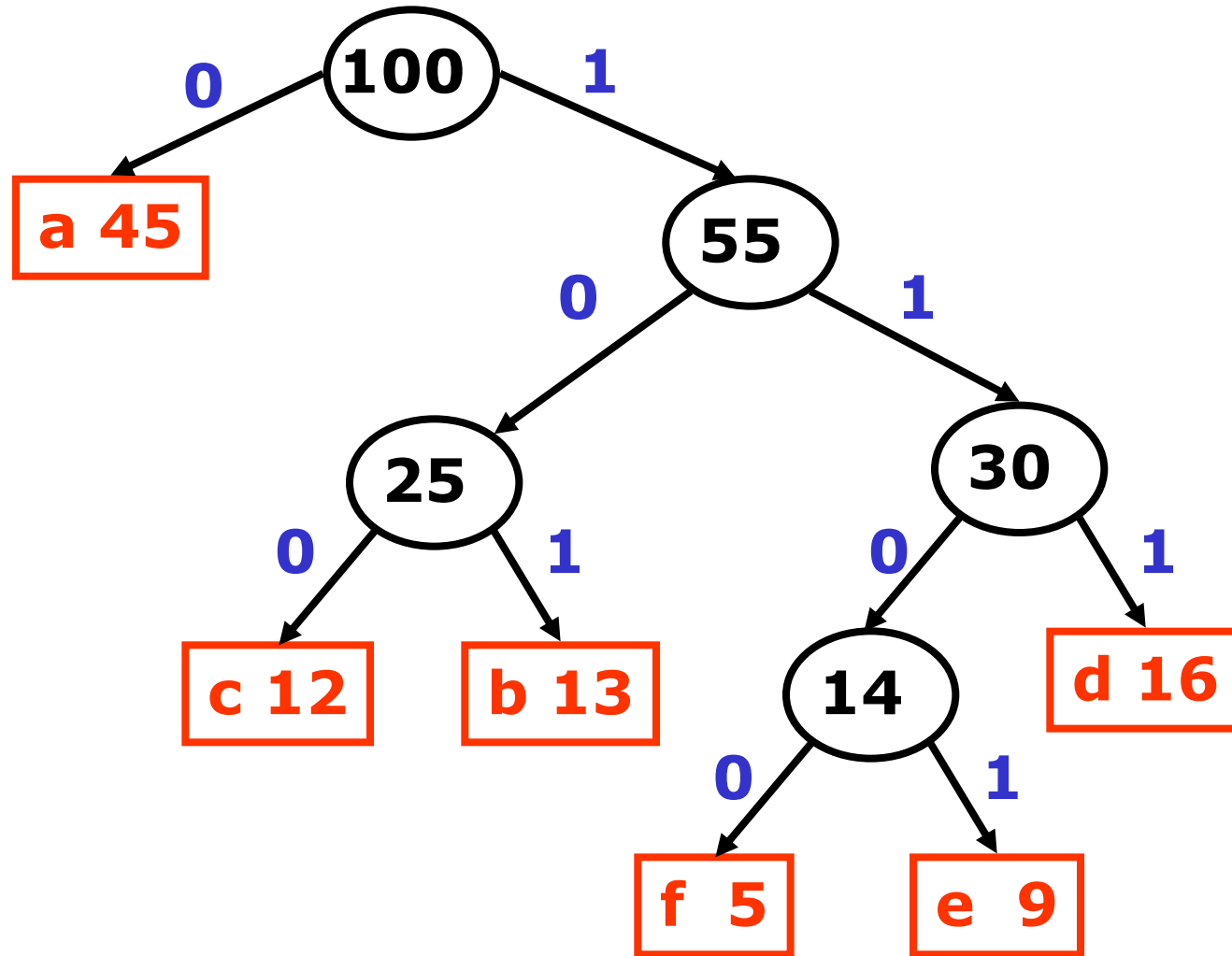
## algoritmo di Huffman



## algoritmo di Huffman



## algoritmo di Huffman



## algoritmo di Huffman: complessità

Gli alberi sono memorizzati in un **minheap** (heap con ordinamento inverso : la radice è il più piccolo)

Si fa un ciclo dove in ogni iterazione:

- vengono estratti i due alberi con **radice minore**
- vengono fusi in un nuovo albero avente come etichetta della radice la somma delle due **radici**
- l'albero risultante è inserito nello heap

il ciclo ha **n** iterazioni ed ogni iterazione ha complessità  **$O(\log n)$**  (si eseguono 2 estrazioni e un inserimento)

**$O(n \log n)$**

**La scelta locale è consistente con la situazione globale:**

**sistemando prima i nodi con minore frequenza, questi apparterranno ai livelli più alti dell'albero**

## Codice algoritmo Huffman

```
struct NodeH{  
    char symbol;    // carattere alfabeto  
    int freq;       // frequenza carattere  
    NodeH* left; NodeH* right;  
};
```

```
Node* huffman(Heap H, int n){  
    for(int i=0; i< n-1; i++) {  
        NodeH *t = new NodeH();  
        t->left = H.extract();  
        t->right = H.extract();  
        t->freq= t->left->freq + t->right->freq;    //somma le radici  
        H.insert(t);                                // inserimento nello heap  
    }  
    return H.extract();    //ritorna la radice dell'albero  
}
```

## Riferimenti Bibliografici

Demetrescu:

Capitolo 10.3

Cormen:

Capitolo 16

## Esercizio 1

Trovare la/le PLSC per le due sequenze:

**xyzzyx**      **e**      **xxyzxy**

		x	y	z	z	y	x
	0	0	0	0	0	0	0
x	0						
x	0						
y	0						
z	0						
x	0						
y	0						



## Esercizio 2

**Applicare l'algoritmo di Huffman per trovare un codice di compressione per l'alfabeto seguente con le frequenze indicate**

<b>A</b>	<b>13</b>
<b>B</b>	<b>16</b>
<b>C</b>	<b>12</b>
<b>D</b>	<b>10</b>
<b>E</b>	<b>15</b>
<b>F</b>	<b>30</b>
<b>G</b>	<b>4</b>