Algoritmi di compressione

Liceo G.B. Brocchi
Classe 3AQSA – Compresenza Informatica - Arte
Bassano del Grappa, Ottobre 2022

Che cos'è l'informazione digitale

- Immaginate di dover codificare digitalmente un testo che utilizza l'alfabeto di simboli *S = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'}*
- All'interno del testo, ciascuno degli 8 simboli compare con probabilità pari a 1/8
- Sarebbe utile dare una definizione rigorosa del concetto di **quantità di informazione associata ad un simbolo $s_i \in S$**
- In informatica riusciamo già a misurare l'informazione tramite l'unità di misura fondamentale chiamata **bit** *(binary digit)*
- Ad esempio, diciamo che per rappresentare gli interi compresi tra 0 e 255 servono **8 bit**
  - di conseguenza, diciamo che un intero compreso tra 0 e 255 *pesa* 8 bit, siano essi nella memoria di un computer o trasmessi digitalmente in un canale di comunicazione

Interessante, ma c'è un modo più matematico-formale di spiegare la stessa cosa?

Che cos'è l'informazione digitale

- Immaginate di dover codificare digitalmente un testo scritto con l'alfabeto di simboli

  $S = \{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'\}$

- Questa volta però sappiamo che il testo contiene il simbolo 'A' con probabilità pari a 1
  - Siamo cioè sicuri che il testo sarà fatto così:
    - «AAAAAAAAAAAAAAAAAAAAA …. AAAAAAAAAAAAAA»

- Che quantità di informazione è associata a ciascun simbolo dell'alfabeto?

- Hint: pensate all'informazione come all'*imprevedibilità di un fatto*

Che cos'è l'informazione digitale

- Un testo contenente solo il simbolo 'A' con probabilità pari a 1 è ***completamente prevedibile***
- Esempio: se un telegiornale parlasse sempre della stessa notizia ogni giorno da anni, questa notizia costituirebbe un'*informazione*?
- Se il testo contenesse il simbolo 'A' con probabilità ½, e 'B' con probabilità ½ (gli altri simboli con probabilità 0) il testo sarebbe non sarebbe completamente prevedibile.
  - quanti bit serviranno per rappresentare il contenuto di un testo costruito in questo modo?
    - 'A' -> 0
    - 'B' -> 1

1 bit

Che cos'è l'informazione digitale

- Se il testo contenesse il simbolo 'A' con probabilità ¼, 'B' con probabilità ¼, 'G' con probabilità ¼, 'H' con probabilità ¼, gli altri simboli con probabilità 0, quanti bit servirebbero per rappresentare ciascun simbolo?
  - 'A' -> 00
  - 'B' -> 01
  - 'G' -> 10
  - 'H' -> 11

2 bit

Che cos'è l'informazione digitale

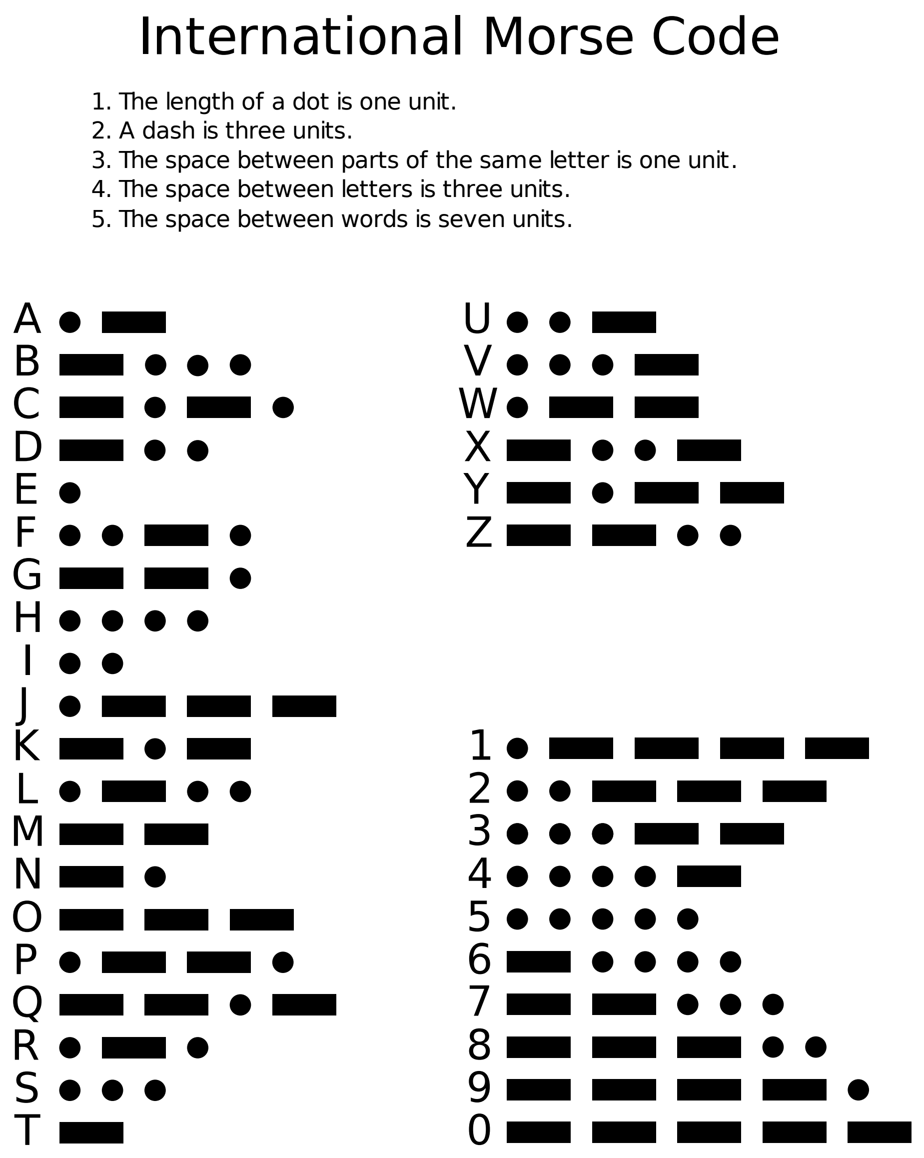
- Consideriamo ora l'alfabeto di dimensione 16 *S = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'L', 'M', 'N', 'O', 'P', 'Q', 'R'}*
- Sappiamo che 'E' compare con probabilità ½
- Sappiamo che 'A', 'B', 'C', 'D', 'O', 'P', 'Q', 'R' compaiono ciascuno con probabilità 1/16
- Intuitivamente, servono più bit per rappresentare 'E' o per rappresentare uno tra i simboli tra i simboli che compaiono con probabilità 1/16?

È più prevedibile la presenza di 'E' o di un simbolo tra 'A', 'B', 'C', 'D', 'O', 'P', 'Q', 'R'?

Excursus: il codice Morse

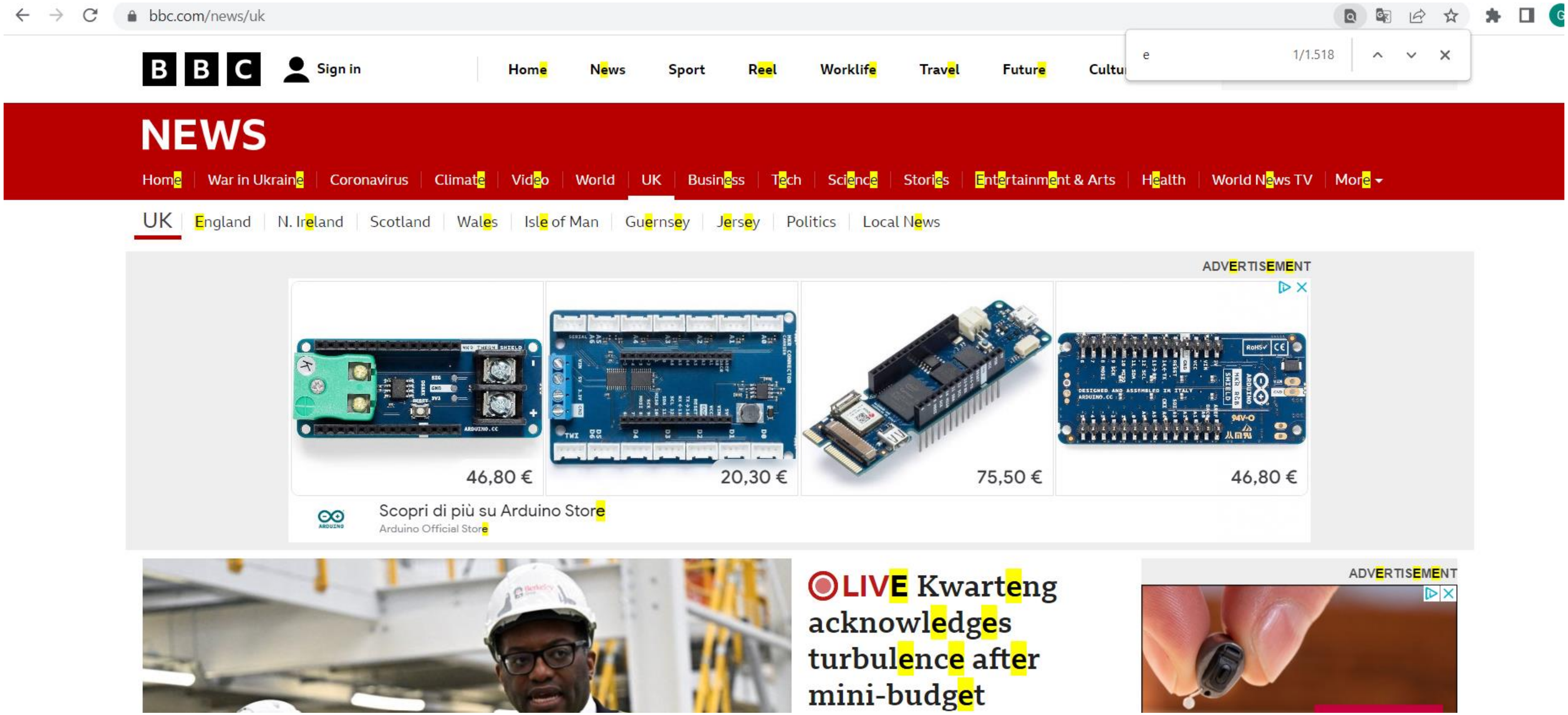
- https://en.wikipedia.org/wiki/Samuel_Morse

Il codice a destra è pensato per la lingua inglese.
I codici delle lettere hanno tutti la stessa lunghezza?



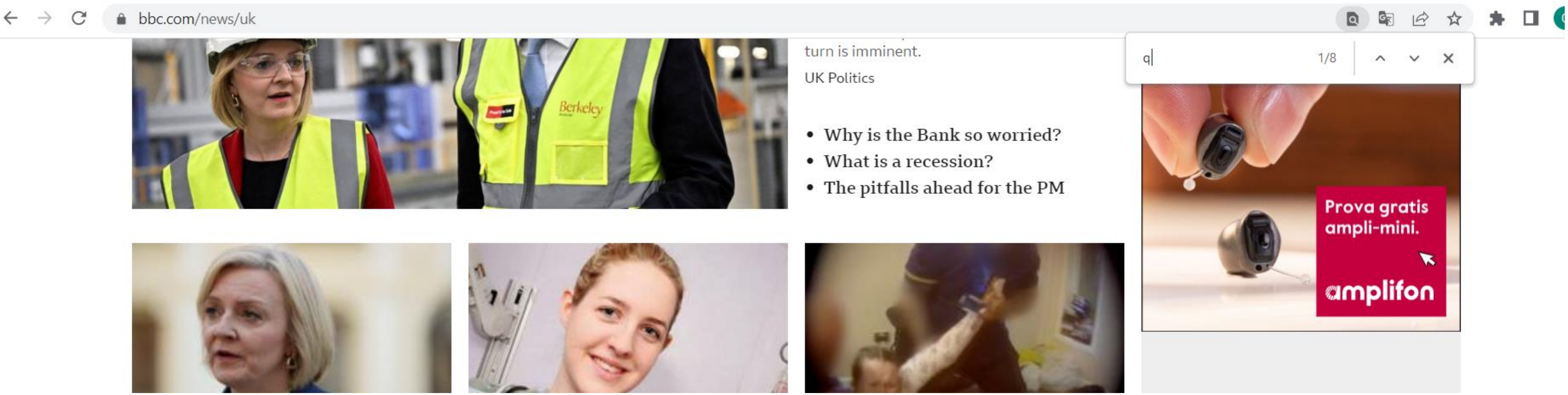
Excursus: il codice Morse

1518 occorrenze del carattere 'e'



Excursus: il codice Morse

8 occorrenze del carattere ‘q’



L'informazione secondo Claude E. Shannon

- https://en.wikipedia.org/wiki/Claude_Shannon

- $S = \{s_1, s_2, \; s_3 \; \ldots sn\}$ (alfabeto di simboli)

- Self-information of s_i. Quantità di informazione contenuta in

$$\mathbf{s_i = \log(1/p_i)}$$

- p_i è la probabilità con cui compare s_i nella sorgente di informazione

Che cos'è l'informazione digitale

- $S = \{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'\}$

- Ciascun simbolo compare con probabilità 1/8

- Ogni simbolo trasporta $\log_2(1/(1/8))$ bit di informazione: **3 bit**

- S = un alfabeto di 32 caratteri
    - Ciascuna lettera compare con probabilità 1/32
        - Ogni lettera trasporta $\log_2(1/(1/32))$ bit di informazione: **5 bit**

Che cos'è l'informazione digitale

- Quindi Samuel Morse ci aveva più o meno azzeccato (1 secolo prima della definizione di Shannon)
- Si tratta già di uno schema di compressione! Morse avrebbe potuto ignorare la probabilità di occorrenza dei caratteri in inglese e codificare tutte le lettere con 5 o 6 «bit»!
- Se caratteri più frequenti della lingua inglese venissero codificati con codici della stessa lunghezza di quelli meno frequenti si sprecherebbe un sacco di «banda»
- Quindi, ad un simbolo viene associato un codice di lunghezza inversamente proporzionale alla sua frequenza (o *probabilità di occorrenza*)
- Codici di questo tipo si chiamano **Variable Length Coding**

Run Length Coding

- Un file è una fonte di informazione con memoria: ad un certo punto del file, possiamo dire quali sono i bit precedenti e quali i successivi. Sfruttiamo questa *memoria*
- Un semplice esempio in C++:

```
struct run {
	bool bit;
	int length;
};

int main() {
	bool source[10] = {1, 0, 0, 0, 0, 0, 0, 0, 0, 0};
	cout << sizeof(source) << endl;
	struct run r1 = {1, 1};
	struct run r2 = {0, 9};
	cout << sizeof(r1) + sizeof(r2) << endl;
}
```

Huffman codes (David A. Huffman, 1952)

- Consideriamo i dati da comprimere come una sequenza di caratteri
- Abbiamo bisogno di una tabella che contenga la frequenza di utilizzo di ciascun carattere all'interno file di testo da comprimere
- Immaginiamo di avere un file composto da 1.0E+05 (100 000 caratteri)
- Nel testo compaiono soltanto 6 caratteri diversi, che potrebbero essere:
  - `a, b, c, d, e, f`
  - `a` compare 45 000 volte
- Se usassimo un *fixed-length code* avremmo bisogno di 3 bit per ciascun carattere:
  - con 2 bit arriviamo a 2^2 caratteri, che è < 6
  - con 3 bit arriviamo a 2^3 caratteri, che è > 6

Huffman codes

- 100 000 caratteri, 3 bit per caratteri
- Il file peserà 300 000 bit, ossia 37 500 byte, circa 37 KB

Can we do better?

Huffman codes

- Samuel Morse, già nel XIX secolo, codificava le lettere più frequenti con simboli più brevi, arrivando a produrre un **variable-length code**
- Consideriamo una codifica detta **prefix code**, per cui vale sempre la seguente condizione:
  - se il simbolo $\mathbf{a_1}$ è codificato con la stringa binaria (codeword) $\mathbf{c_1}$ e il simbolo **a2** è codificato con la stringa binaria **c2**, allora **c1** non è un prefisso di e **c2** non è un prefisso di **c1**

Huffman codes

	a	b	c	d	e	f
frequency (in thousands)	**45**	**13**	**12**	**16**	**9**	**5**
fixed-length codeword	**000**	**001**	**010**	**011**	**100**	**101**
non-prefix, variable-length codeword	**0**	**01**	**011**	**0111**	**01111**	**011111**
prefix, variable-length codeword	**0**	**101**	**100**	**111**	**1101**	**1100**

Spiegare perché è un prefix code

spiegare perché non è un prefix code

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	**0**	**101**	**100**	**111**	**1101**	**1100**

- I prefix code sono perfetti per essere decodificati. Ecco un esempio. Immaginare un software decodificatore che inizia a leggere la stringa seguente da sinistra

001011101

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	0	101	100	111	1101	1100

001011101

0 corrisponde ad ‘a’
Vai avanti di 1

Output: a

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	0	101	100	111	1101	1100

001011101

0 corrisponde ad ‘a’
Vai avanti di 1

Output: aa

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	0	101	100	111	1101	1100

001011101

1 non è una codeword
10 non è una codeword
101 codifica 'b'
Vai avanti di 3

Output: aab

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	0	101	100	111	1101	1100

001011101

1 non è una codeword
11 non è una codeword
110 non è una codeword
1101 codifica 'e'
Vai avanti di 4

Output: aabe

Huffman codes

	a	b	c	d	e	f
prefix, variable-length codeword	0	101	100	111	1101	1100

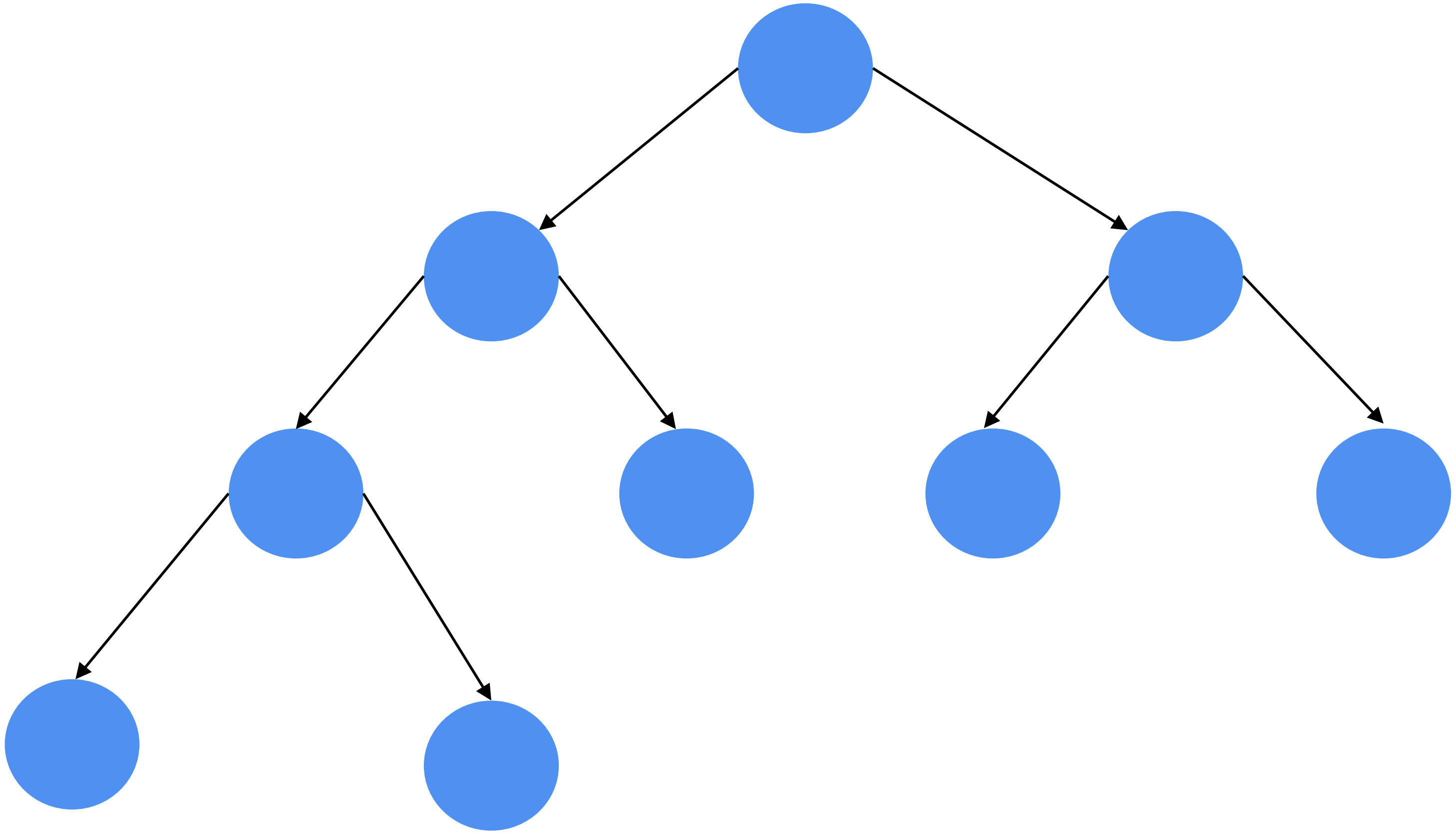
001011101

HALT PROGRAM!

Output: aabe

L'algoritmo di Huffman

Un albero binario è (definizione <u>ricorsiva</u>):
1) **l'albero vuoto**
2) **un nodo radice collegato a 2 alberi binari figli**



L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**

Iterare le seguenti operazioni fintantoché ci sono caratteri nell'elenco:

1) consideriamo 2 caratteri di frequenza più bassa
2) li rimuoviamo dall'elenco dei caratteri
3) creiamo un nodo dell'albero contenente la somma delle frequenze dei 2 caratteri estratti, che ha come figli i 2 caratteri estratti dall'elenco
4) inseriamo il nodo creato nell'elenco

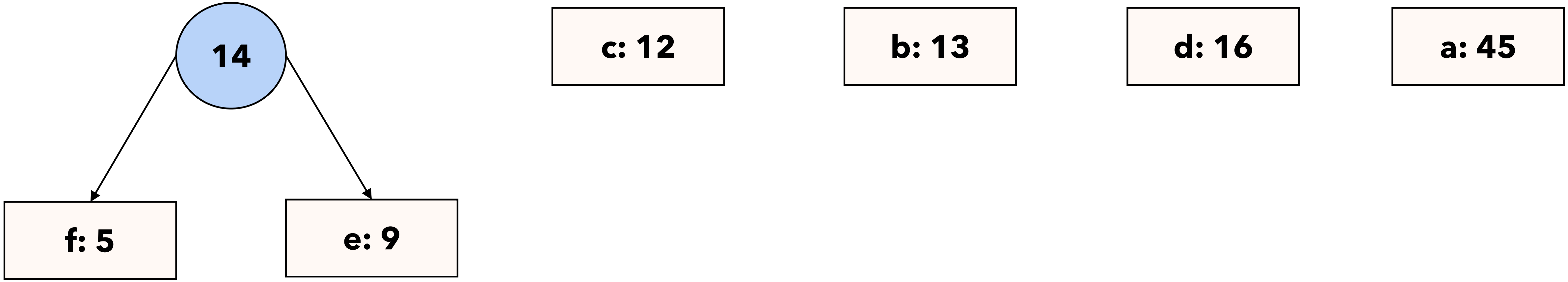
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**

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

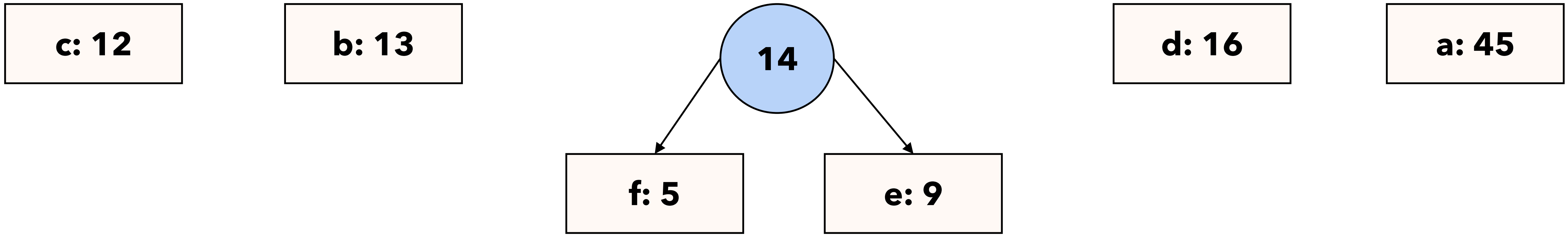
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**



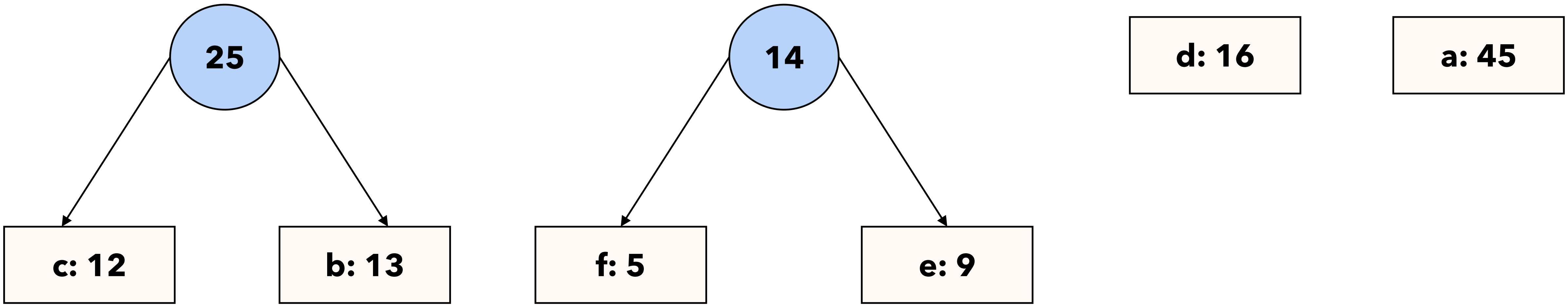
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	5	9	12	13	16	45



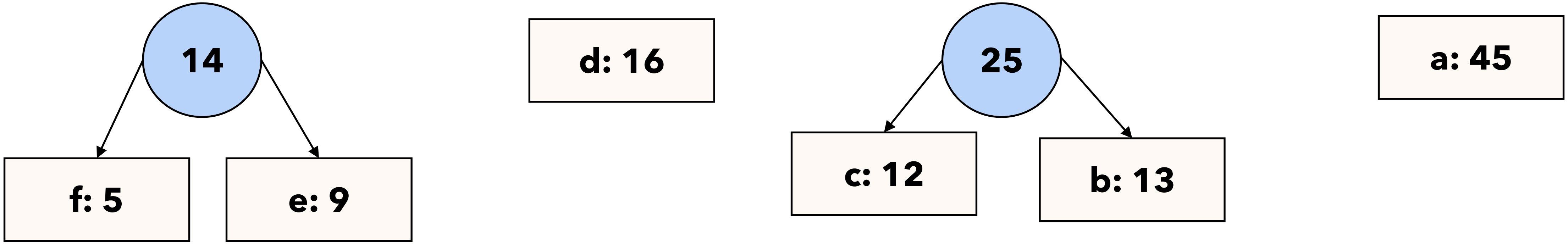
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**



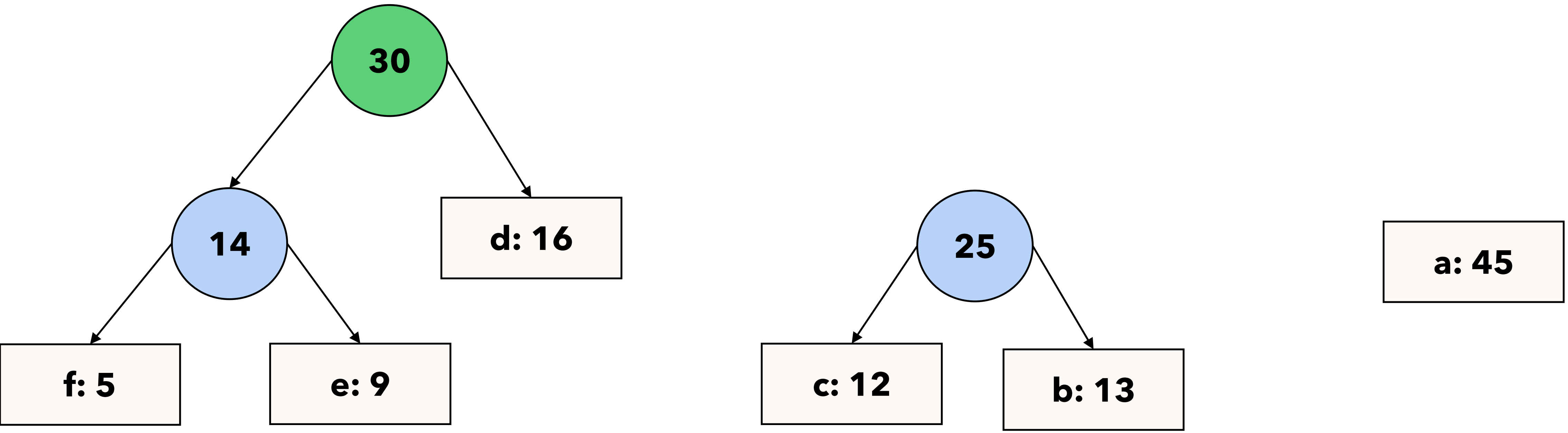
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	5	9	12	13	16	45



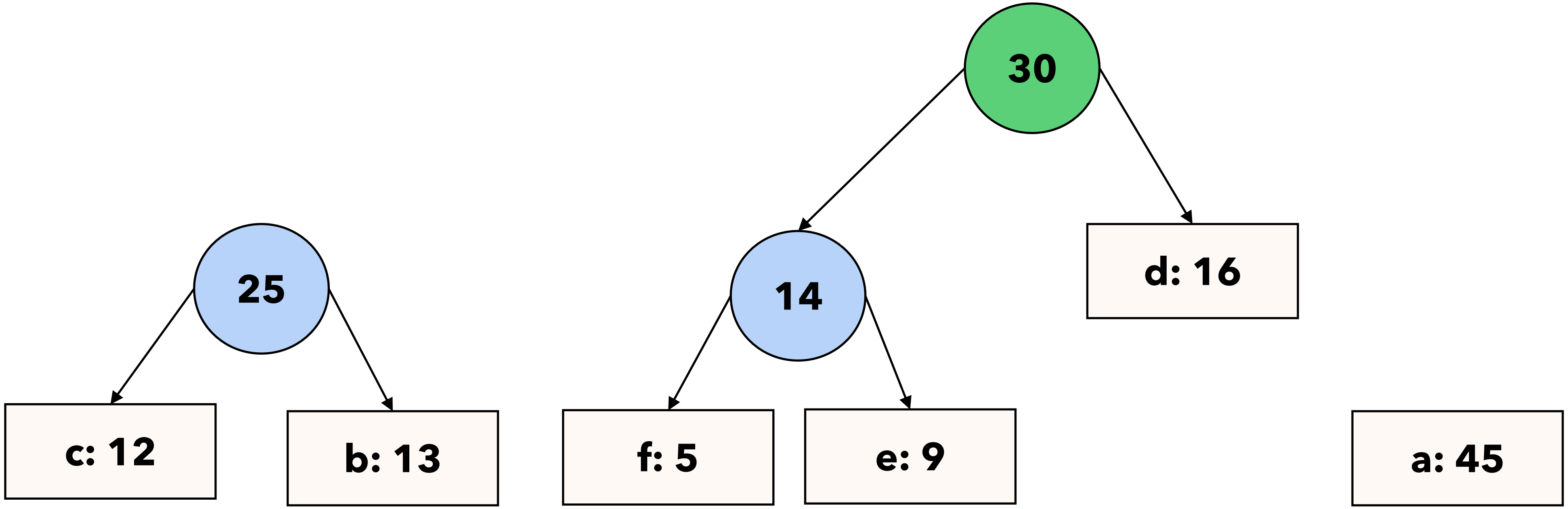
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**



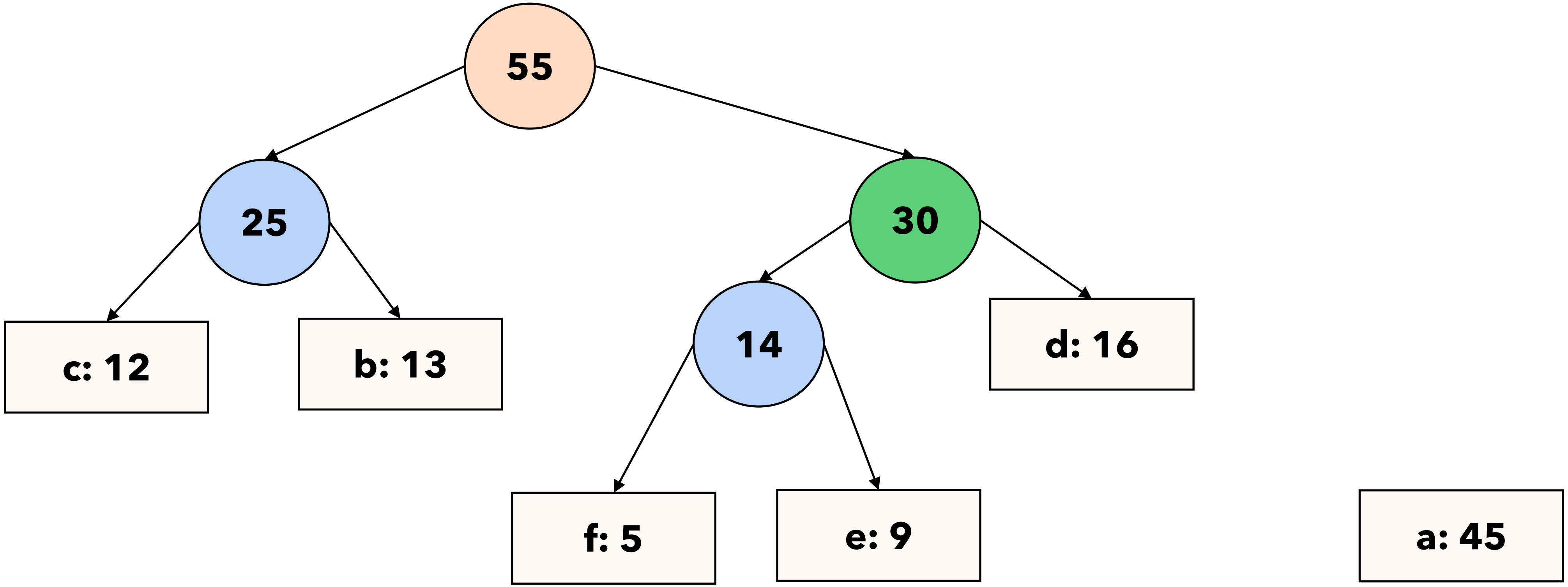
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	5	9	12	13	16	45



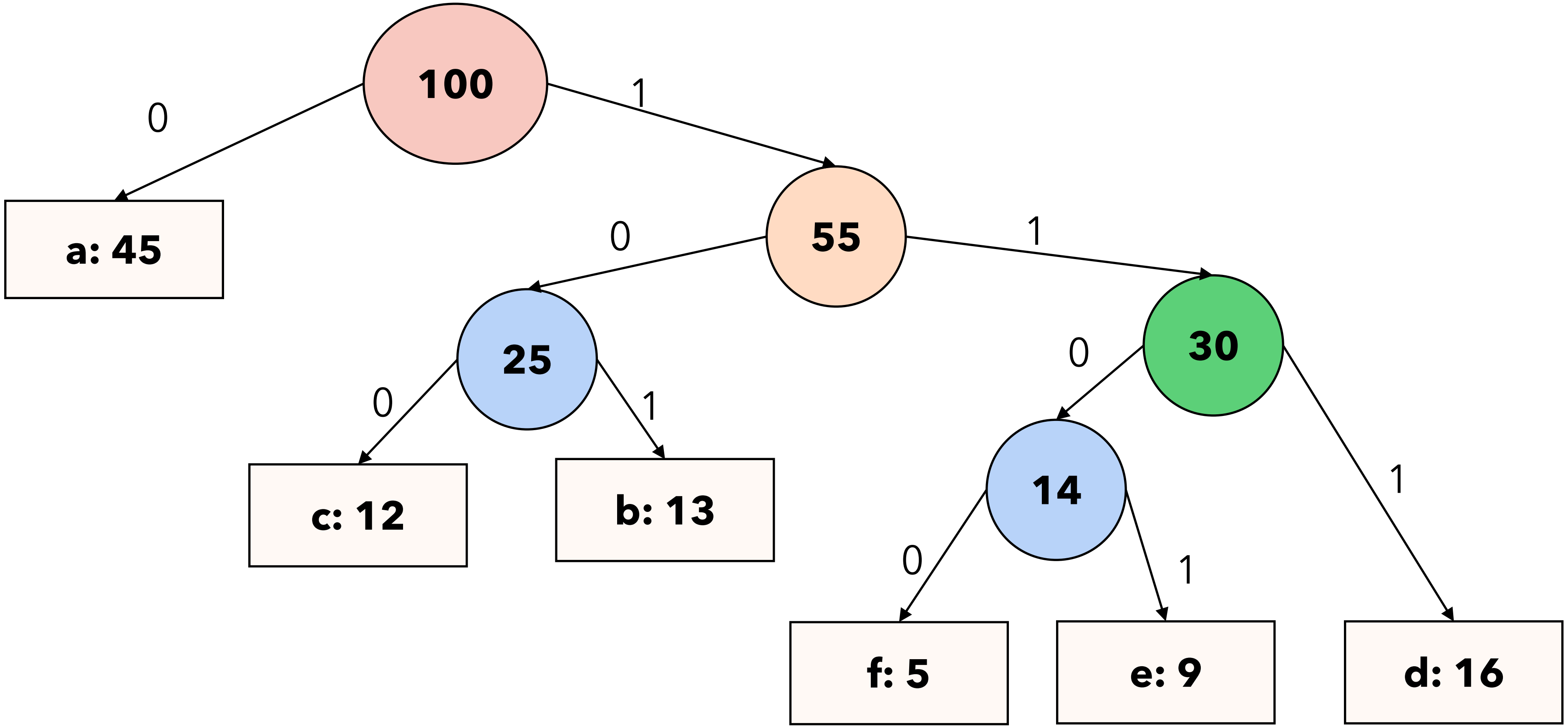
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	5	9	12	13	16	45



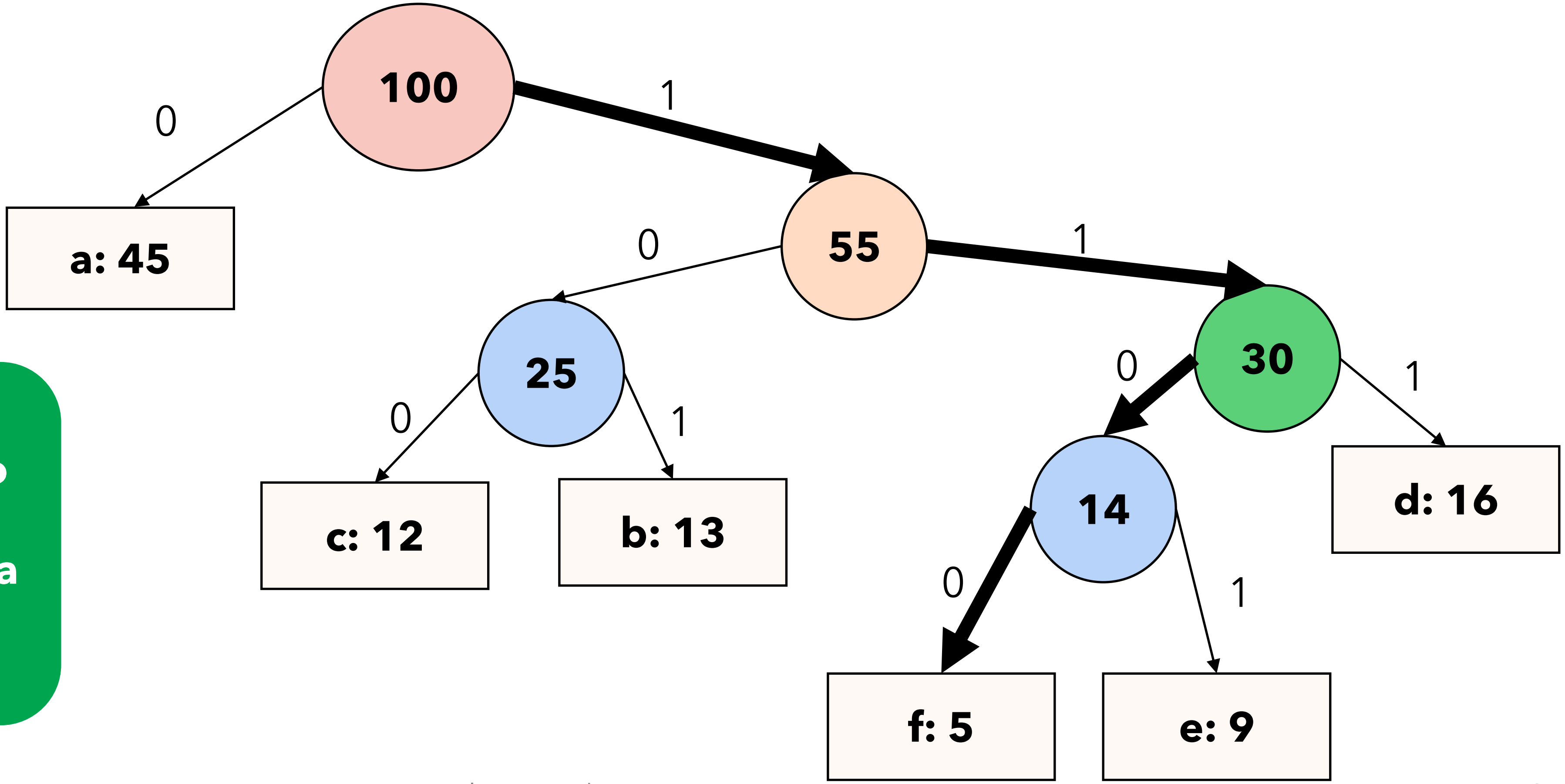
L'algoritmo di Huffman

	f	e	c	b	d	a
frequency (in thousands)	**5**	**9**	**12**	**13**	**16**	**45**



L'algoritmo di Huffman

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



il codice di un carattere è prodotto dal percorso dalla radice fino alla foglia corrispondente al carattere stesso