ALGORITMI E STRUTTURE DATI

Prof. Manuela Montangero

A.A. 2022/23

Algoritmi GREEDY: Codifica di Huffman

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Classificazione di problemi

I problemi possono essere classificati per
tipo di soluzione cercata

- PROBLEMI DECISIONALI: la risposta al problema è SI o NO (oppure vero o falso) a seconda che l'INPUT del problema soddisfi o meno una certa proprietà.
  Esempi: ricerca binaria, test di connettività di un grafo…
- PROBLEMI DI RICERCA: tra tutte le soluzioni possibili se ne cerca una (detta *soluzione ammissibile*) che soddisfa una certa condizione.
  Esempi: ordinare in una sequenza in di numeri in ordine crescente, determinare un albero di copertura per una visita DFS in-order…
- PROBLEMI DI OTTIMIZZAZIONE: ad ogni soluzione ammissibile è associato un costo e si cerca la *soluzione ottima*, ovvero una soluzione ammissibile di costo minimo o massimo.
  Esempio: determinare i cammini minimi da sorgente singola

funzione obiettivo

Algoritmi GREEDY

La tecnica **GREEDY** può essere applicata con successo se valgono le seguenti proprietà per il **PROBLEMA** e per l'**ALGORITMO**:

- Il **PROBLEMA** ha una **SOTTOSTUTTURA OTTIMA**:
  una soluzione ottima al problema contiene al suo interno le soluzioni dei sottoprobbroblemi
  Esempio: se un nodo v si trova sul cammino minimo P da s a u,
  allora la parte del cammino P da s a v è minimo per v

- **SCELTA GREEDY**:
  la scelta greedy permette di scegliere un elemento della soluzione che fa parte della soluzione ottima
  Esempio: la scelta greedy di Dijkstra è scegliere, ad ogni iterazione,
  il nodo nella coda con priorità con massima priorità

Compressione file di testo

PROBLEMA (informale):

Usare il minor numero di bit possible per codificare un file di testo

Cosa abbiamo? (INPUT)

- Alfabeto finito Σ di caratteri
- File di testo F con n caratteri di Σ
- Frequenza $f(c)$ di occorrenza per ogni carattere dell'alfabeto $c \in \Sigma$

Cosa vogliamo? (OUTPUT)

una codifica binaria di ogni carattere $cod : \Sigma \rightarrow \{0,1\}^*$ tale che

$$|F| = \sum_{c \in \Sigma} f(c)\,|cod(c)|$$

sia MINIMO

Compressione file di testo

ESEMPIO: $\Sigma = \{a, b, c, d, e, f\}$

carattere	a	b	c	d	e	f	|F|
frequenza	45%	13%	12%	16%	9%	5%	
ASCII	01100001	01100010	0110011	01100100	01100101	01100110	8n
Codifica 1	000	001	010	011	100	101	3n

Codifica 1: $|\Sigma| = 6 \rightarrow \lceil \log |\Sigma| \rceil$ bit sono sufficienti

DECODIFICA? Ogni tre bit del file compresso corrispondono ad un carattere

Compressione file di testo

ESEMPIO: $\Sigma = \{a, b, c, d, e, f\}$

carattere	a	b	c	d	e	f	|F|
frequenza	45%	13%	12%	16%	9%	5%	
ASCII	01100001	01100010	0110011	01100100	01100101	01100110	8n
Codifica 1	000	001	010	011	100	101	3n

Codifica 1: $|\Sigma| = 6 \rightarrow \lceil \log |\Sigma| \rceil$ bit sono sufficienti

Si può fare di meglio?

Compressione file di testo

ESEMPIO: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$

carattere	a	b	c	d	e	f	|F|
frequenza	45%	13%	12%	16%	9%	5%	
ASCII	01100001	01100010	0110011	01100100	01100101	01100110	8n
Codifica 1	000	001	010	011	100	101	3n
Codifica 2	0	100	101	111	1100	1101	2.24n

Codifica 2:

$$0.45n \cdot 1 + 0.13n \cdot 3 + 0.12n \cdot 3 + 0.16n \cdot 3 + 0.09n \cdot 4 + 0.05n \cdot 4 = 2.24n$$

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4)\frac{n}{100} = 2.24n$$

Compressione file di testo

ESEMPIO: $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$

carattere	a	b	c	d	e	f	|F|
frequenza	45%	13%	12%	16%	9%	5%	
ASCII	01100001	01100010	0110011	01100100	01100101	01100110	8n
Codifica 1	000	001	010	011	100	101	3n
Codifica 2	0	100	101	111	1100	1101	2.24n

DECODIFICA?

$$100010001010 \rightarrow 100|0|100|0|101|0 \rightarrow babaca$$

Compressione file di testo

Codifica a lunghezza variabile

una codifica qualunque funziona?

carattere	a	b	c	d	e	f	|F|
frequenza	45%	13%	12%	16%	9%	5%	2.24n
Codifica 3	0	1	10	11	100	101	1.56n

DECODIFICA?

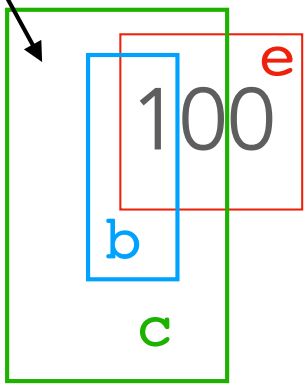
$$100010001010 \rightarrow 1|0|0|0|1|0|0|0|1|0|1|0 \rightarrow baaabaaababa$$
$$100010001010 \rightarrow 10|0|0|10|0|0|10|1|0 \rightarrow caacaacba$$
$$100010001010 \rightarrow 100|0|10|0|0|101|0 \rightarrow eacaafa$$
$$\ldots\ldots$$

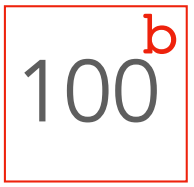
?

Codice a prefisso

Nessuna codifica è un prefisso di un'altra

carattere	a	b	c	d	e	f
Codifica 2	0	**100**	101	111	1100	1101
Codifica 3	0	**1**	**10**	11	**100**	101



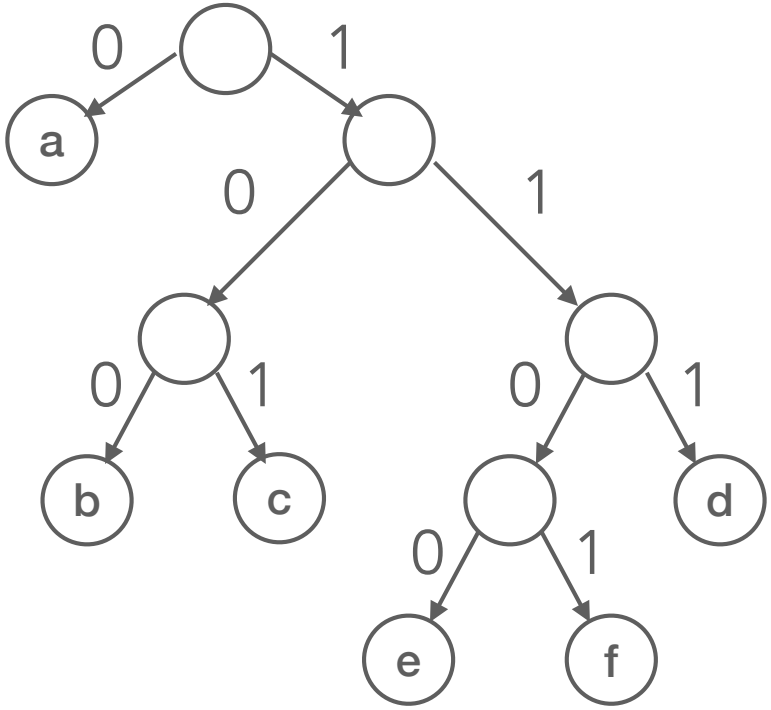


Rappresentazione ad albero e codifica

Albero di codifica:

- Etichette 0/1 sugli archi
  (0 sull'arco per figlio sinistra,
  1 sull'arco per figlio destro)
- Caratteri dell'alfabeto nelle foglie
- Cammino radice-foglia = codifica carattere

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



Algoritmo di decodifica:

```
parti dalla radice
while file non finito do
  leggi un bit
  if bit = 0
    then scendi a sinistra
    else scendi a destra
  if sei in una foglia
    then print carattere
         torna alla radice
```

Compressione file di testo

PROBLEMA

INPUT:

- Alfabeto finito Σ di caratteri
- File di testo F con n caratteri di Σ
- Frequenza $f(c)$ di occorrenza per ogni carattere dell'alfabeto $c \in \Sigma$

OUTPUT

una codifica binaria di ogni carattere $cod : \Sigma \rightarrow \{0,1\}^*$ tale che

$$|F| = \sum_{c \in \Sigma} f(c)\,|cod(c)|$$

sia MINIMO

OSSERVAZIONI:

- Rappresentiamo la codifica usando l'albero di codifica
- Ogni file ha la sua codifica ad-hoc
- Il file codificato deve essere corredato dell'albero per la decodifica

Algoritmo di Huffman

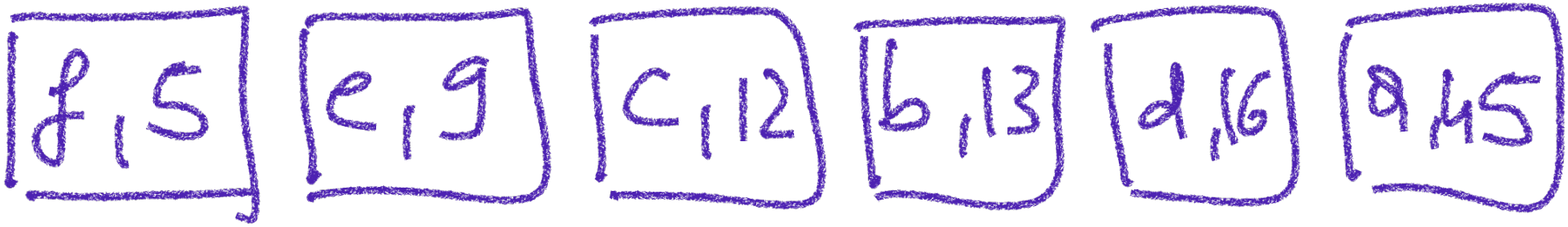
Si può dimostrare che il problema ha una sottostruttra OTTIMA

IDEE:

- Usare un algoritmo greedy
- Assegnare a caratteri meno frequenti codifiche più lunghe
- Assegnare a caratteri più frequenti codifiche più corte
- Assegnare a caratteri meno frequenti percorsi più lunghi sull'albero
- Assegnare a caratteri più frequenti codifiche più corte sull'albero

Algoritmo di Huffman

1) Costruire una lista ordinata di nodi foglia, uno per ogni carattere `c` dell'alfabeto, etichettati con la coppia `(c,f(c))`



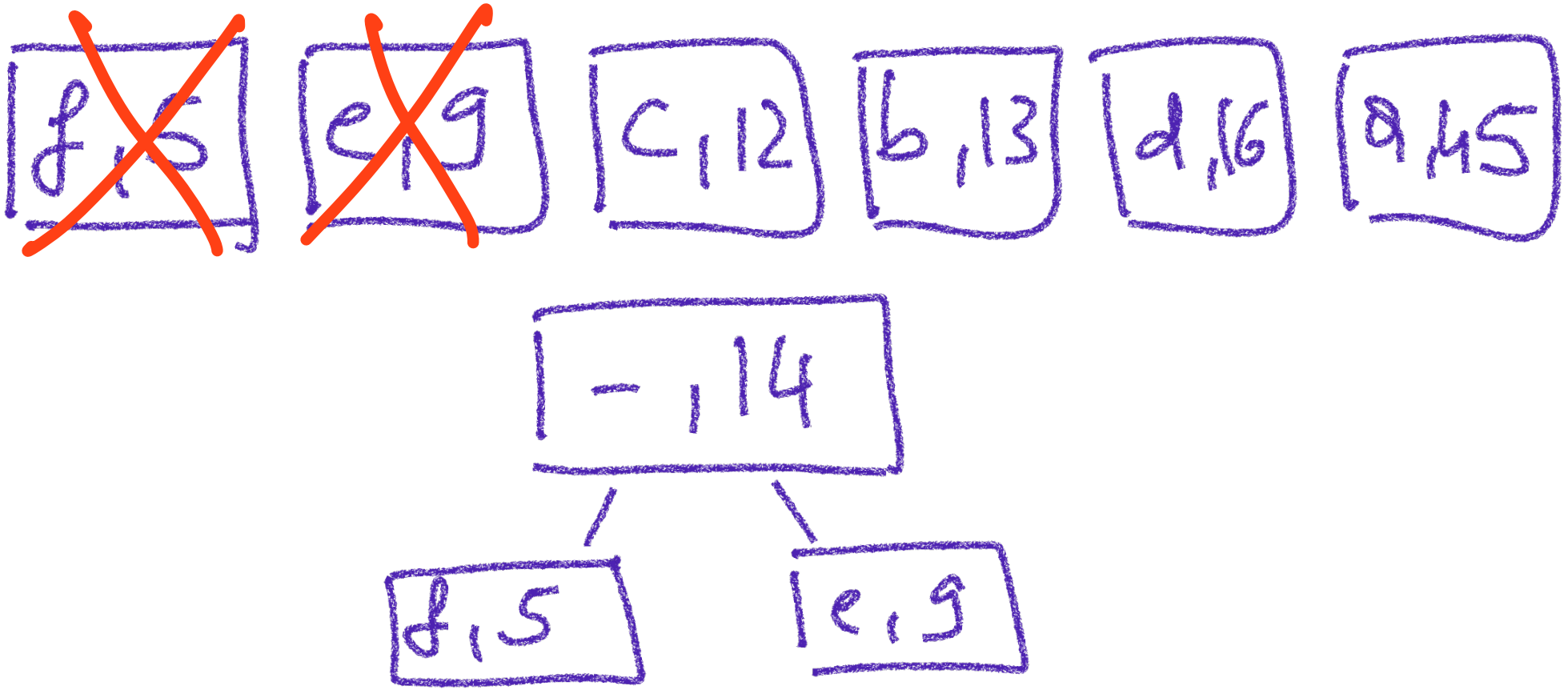
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata

SCELTA GREEDY
si può dimostrare che porta a costruire la soluzione OTTIMA

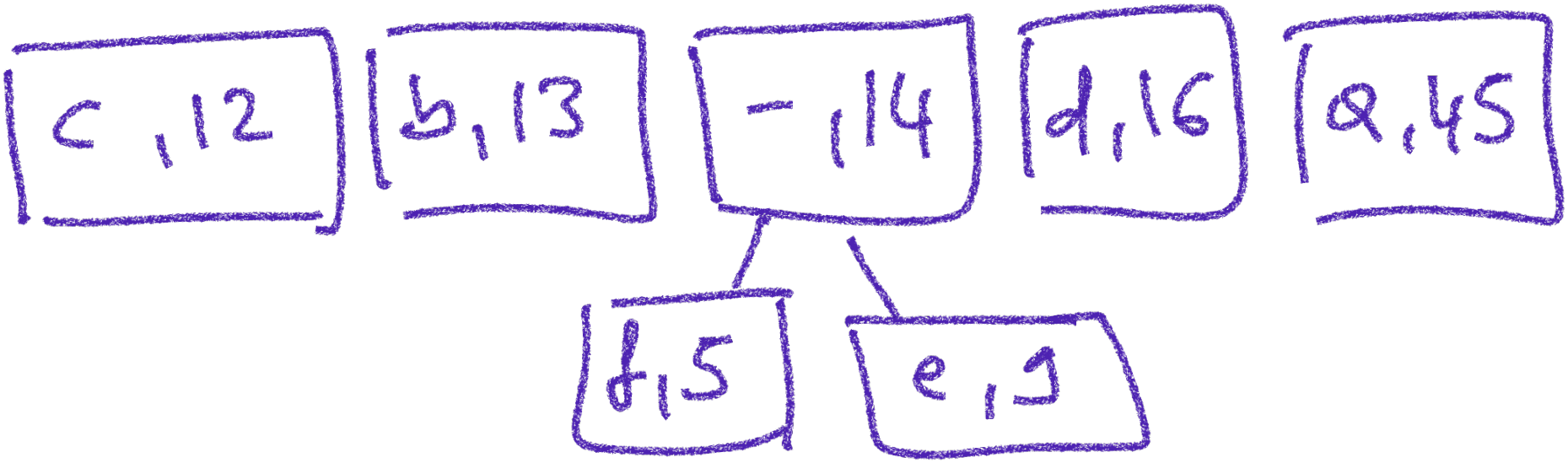
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



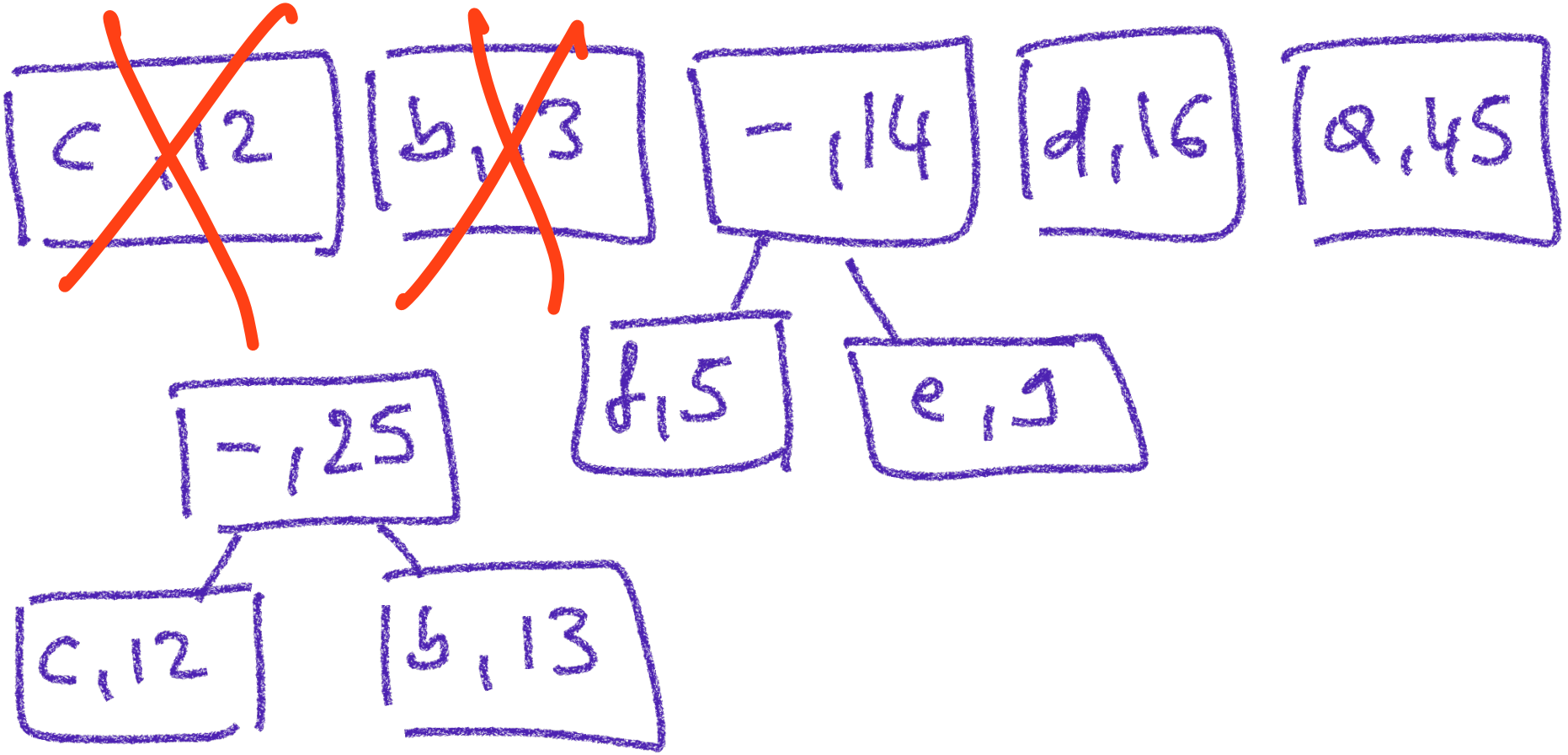
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



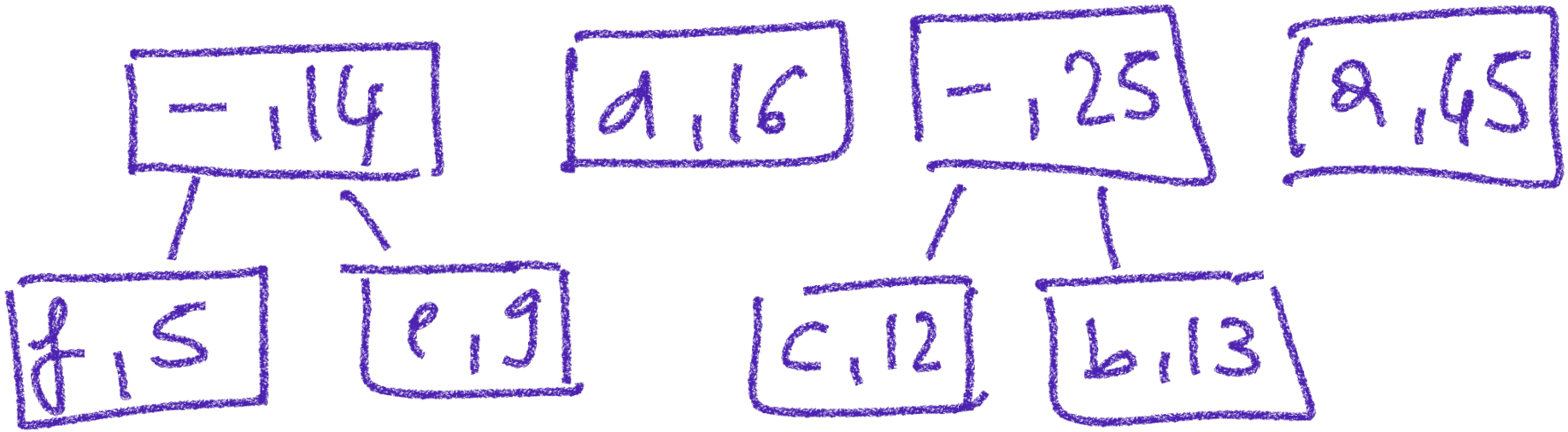
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



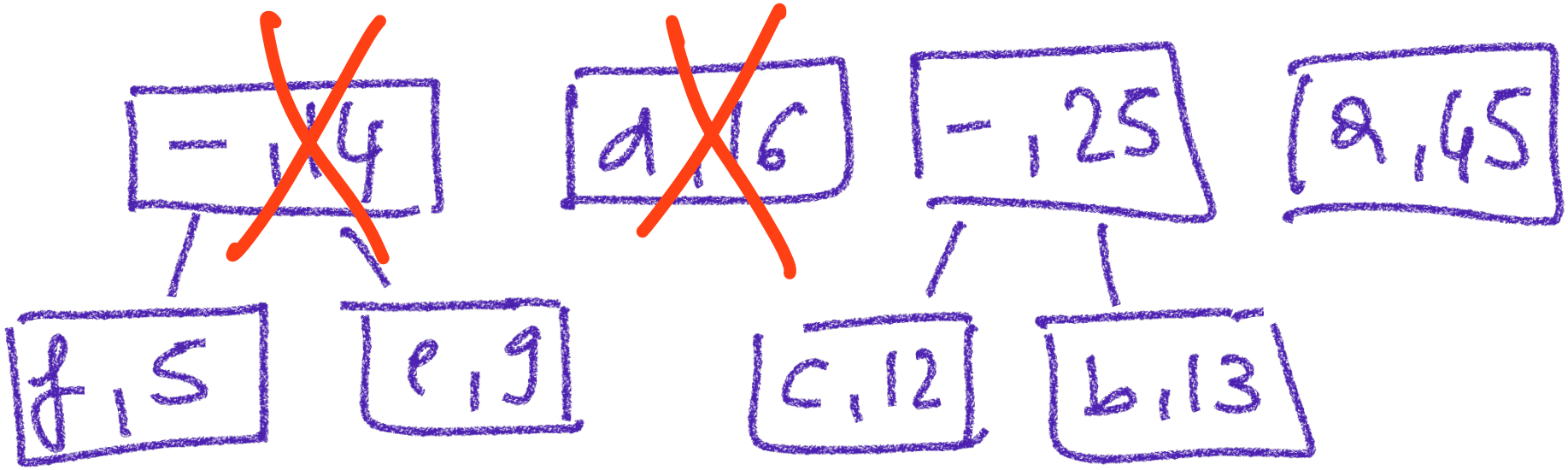
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



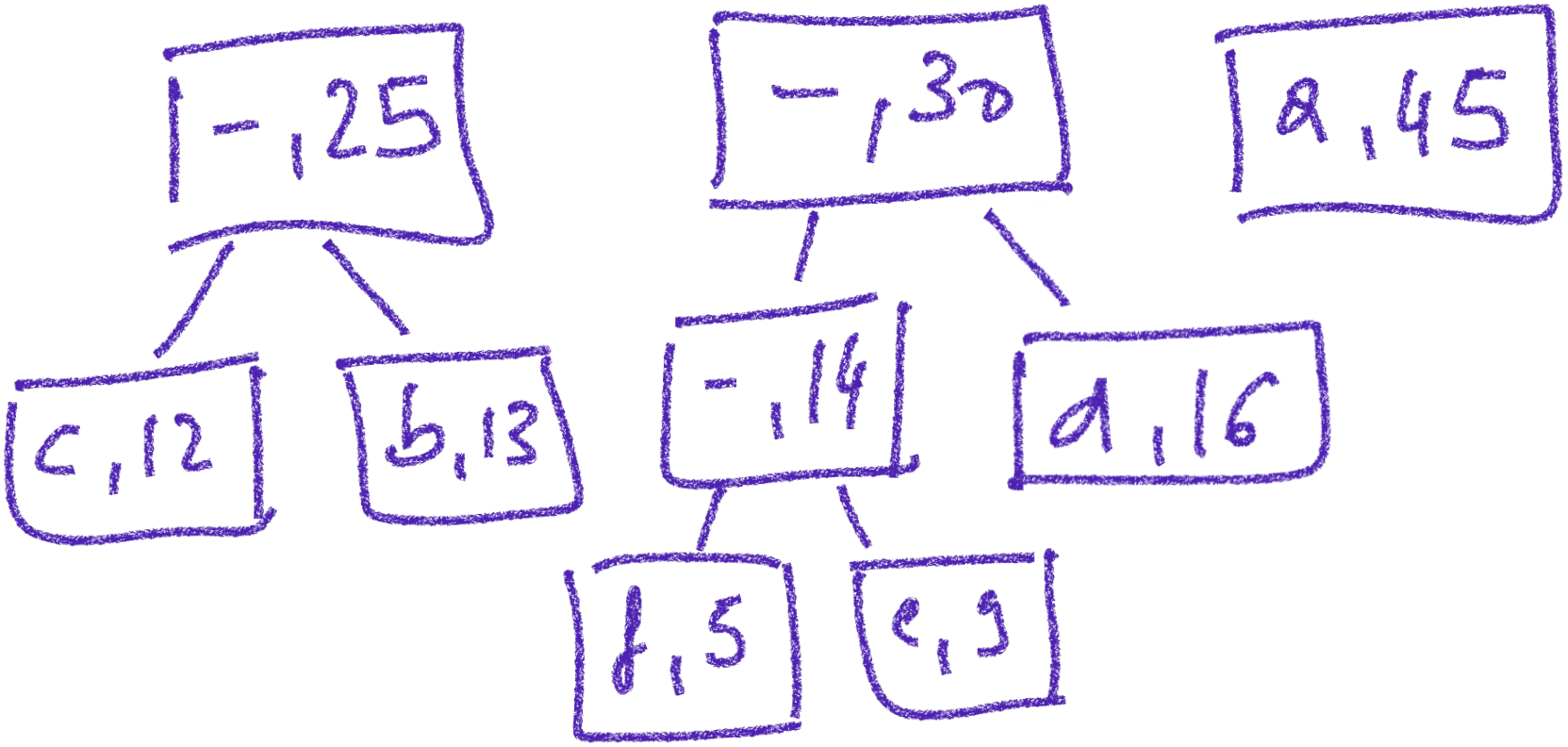
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



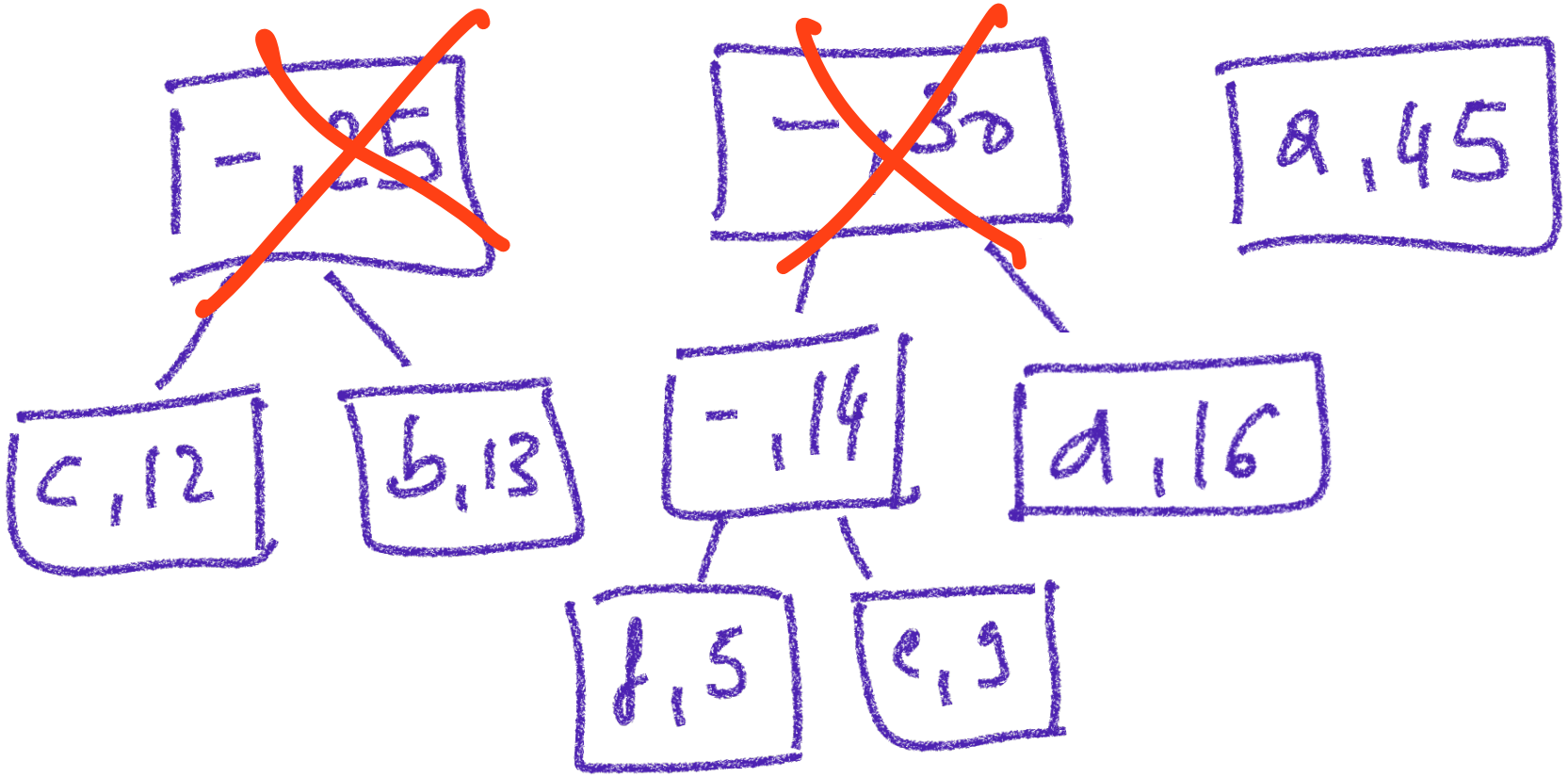
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



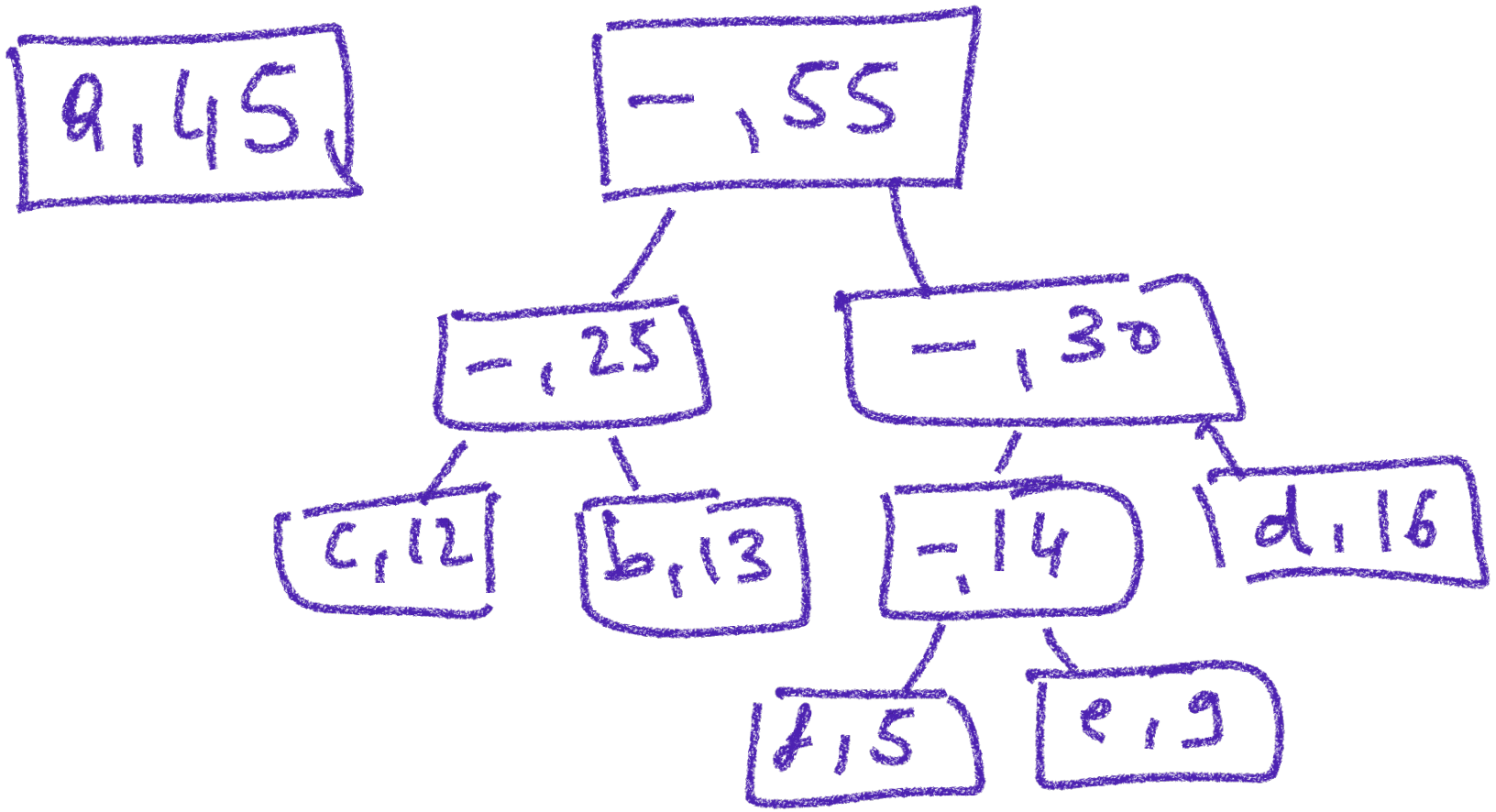
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



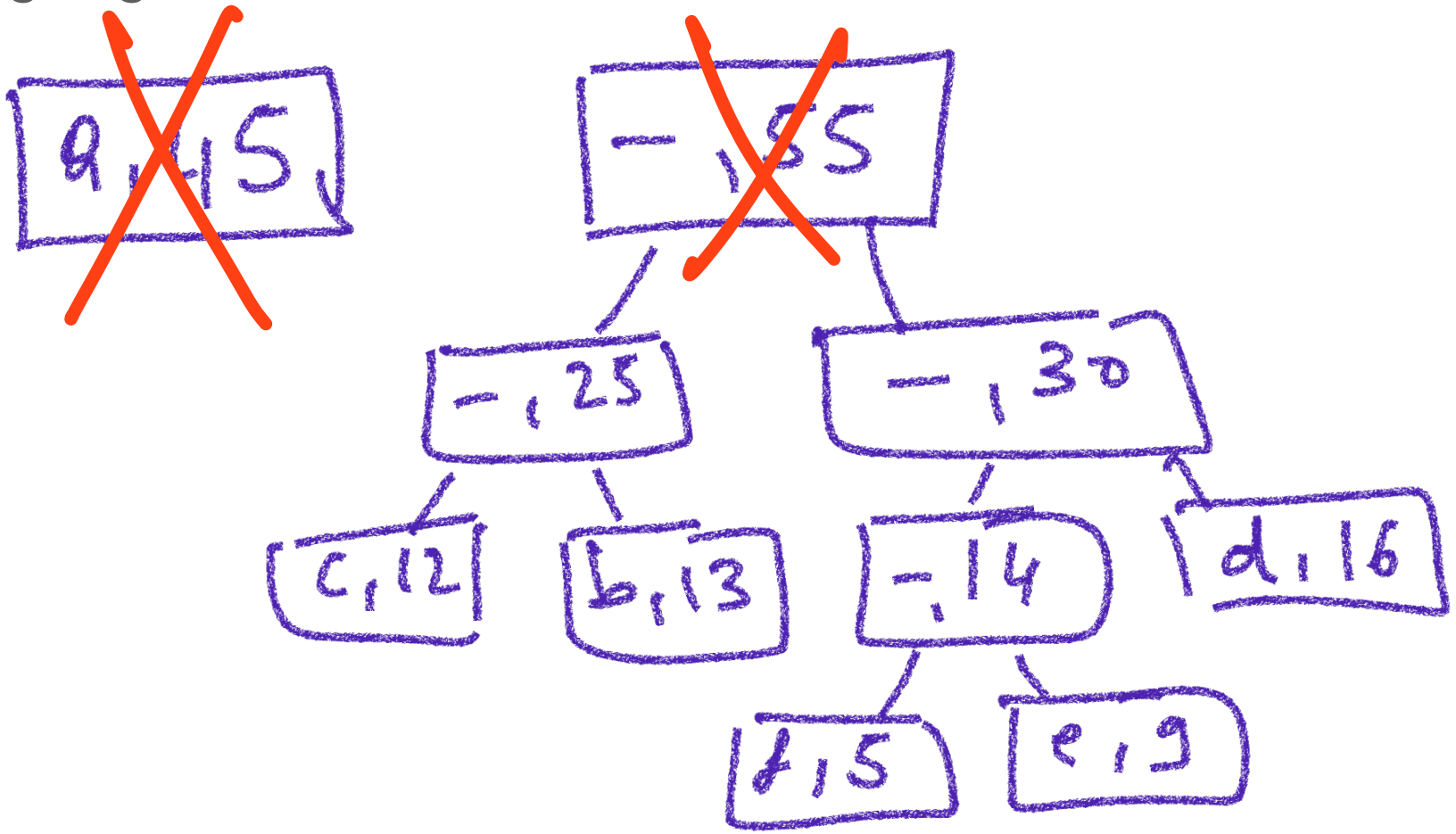
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



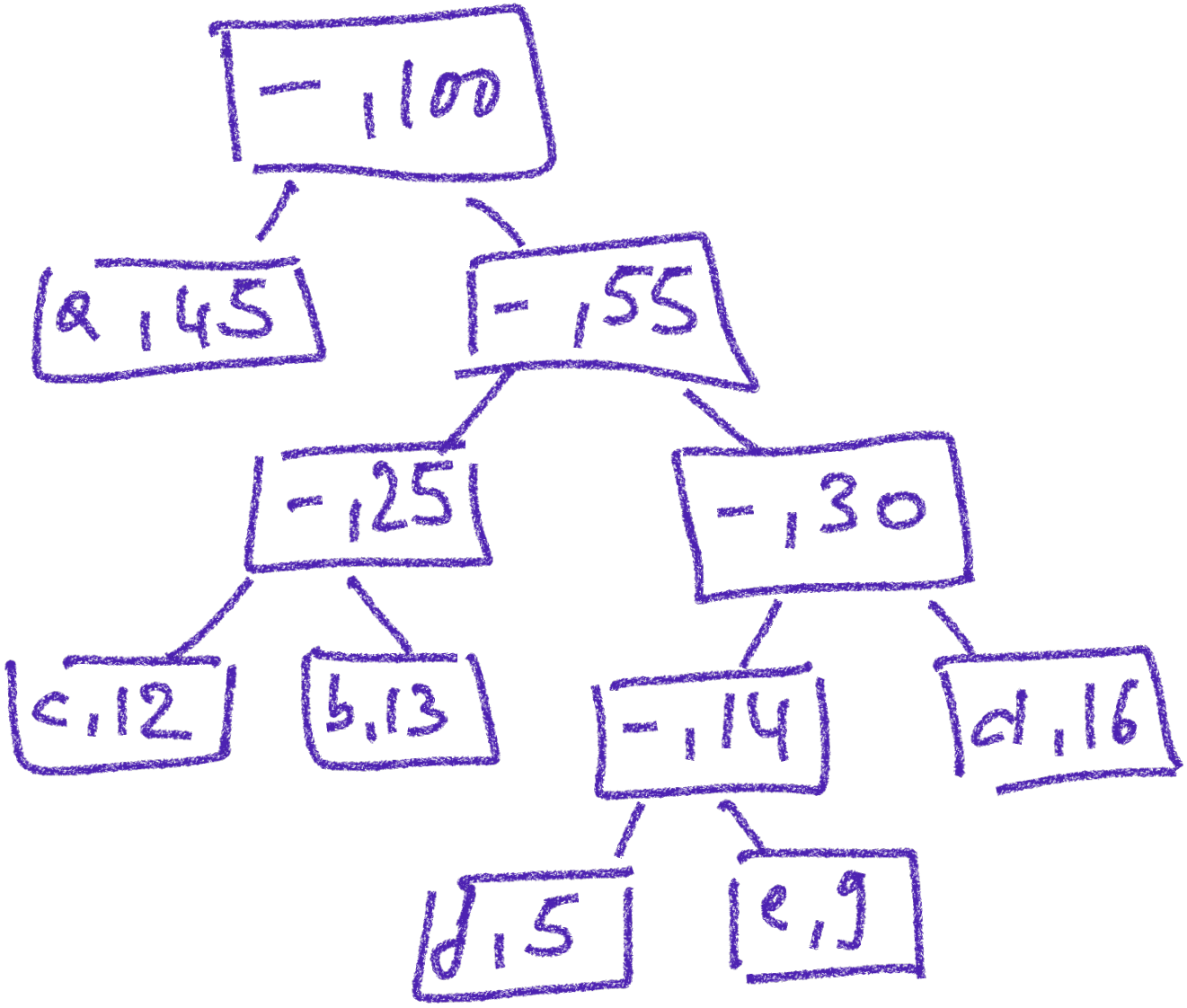
Algoritmo di Huffman

2) Rimuovere dalla lista i due nodi con frequenza minore f_x e f_y
3) Creare un nuovo nodo con etichetta $(-, f_x+f_y)$ e figli i nodi rimossi al passo precedente
4) Aggiungere il nuovo nodo nella lista ordinata



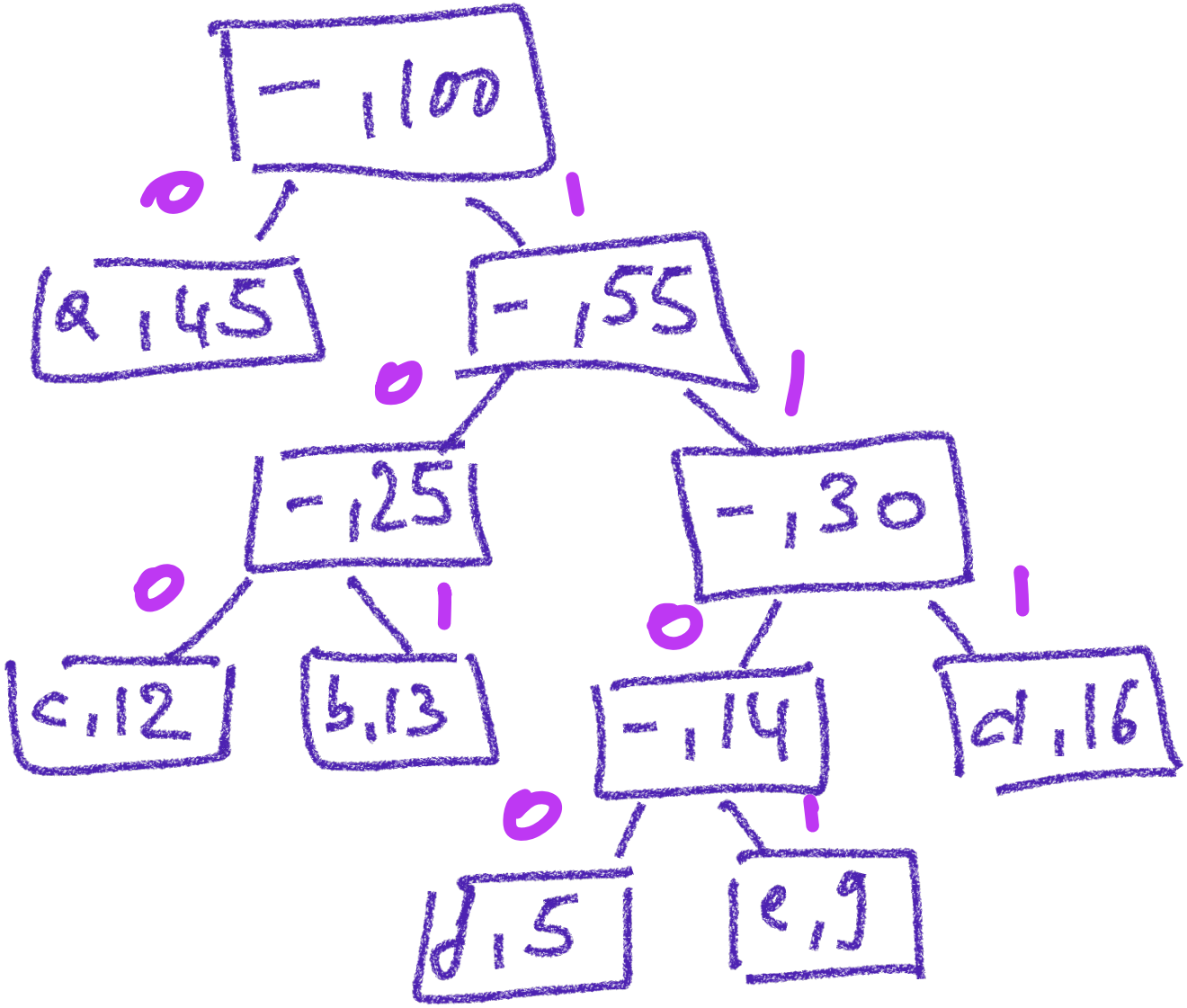
Algoritmo di Huffman

5) Si termina quando resta un solo nodo nella lista



Algoritmo di Huffman

6) Etichettare gli archi dell'albero



Algoritmo di Huffman

6) Etichettare gli archi dell'albero

a → 0

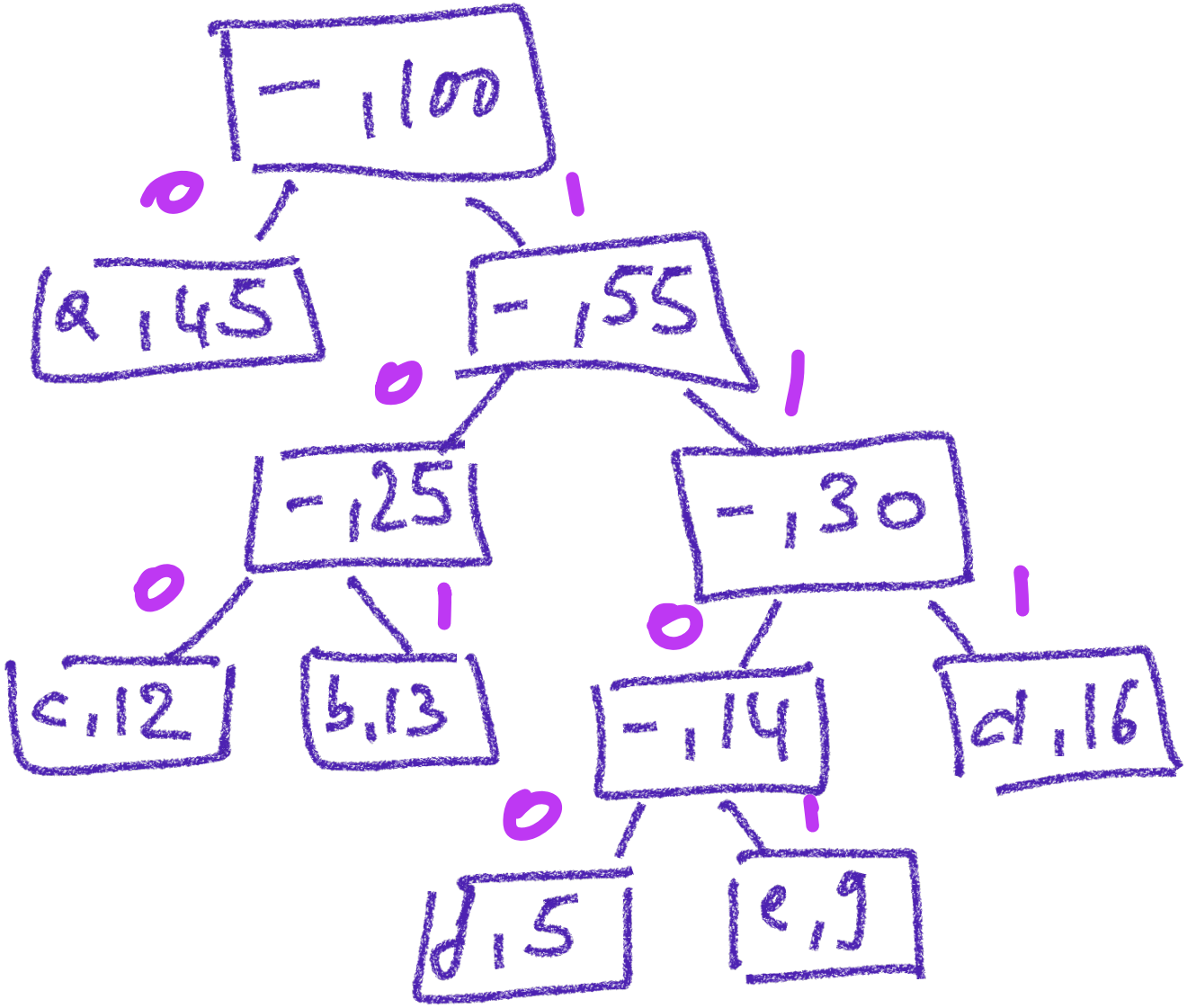
b → 101

c → 100

d → 111

e → 1101

f → 1100



Algoritmo di Huffman

Scelte implementative:

come realizziamo la lista ordinata?

CODA con PRIORITÀ

`(el, pr)`

albero di codifica
per insieme di caratteri

somma delle frequenze
nell'insieme di caratteri

Algoritmo di Huffman

```
Huffman(Σ[1..|Σ|],f[1..|Σ|],|Σ|))
Q := new_priority_queue()
for i = 1 to |Σ| do
  t := new_tree_node()
  t.c := Σ[i]
  t.fr := f[i]
  t.left := NIL
  t.right := NIL
  enQueue(Q,t,f[i])
for i = 1 to |Σ|-1 do
  t1 := DeQueue(Q)
  t2 := DeQueue(Q)
  t := new_tree_node()
  t.c := -
  t.fr := t1.fr + t2.fr
  t.left := t1
  t.right := t2
  enQueue(Q,t, t1.fr + t2.fr)
return DeQueue(Q)
```

```
INPUT
Σ[1..|Σ|] -> array caratteri alfabeto
f[1..|Σ|] -> frequenze caratteri
             nel testo
```

```
campi tree_node
c -> carattere  ∈ Σ ∪ { - }
fr -> somma frequenze
      caratteri nel sottoalbero
left -> puntatore albero sx
right -> puntatore albero dx
```

Si può dimostrare che determina la codifica ottima

Algoritmo di Huffman

```
Huffman(Σ[1..|Σ|],f[1..|Σ|],|Σ|)
Q := new_priority_queue()                    O(1)
for i = 1 to |Σ| do
 t := new_tree_node()
 t.c := Σ[i]
 t.fr := f[i]                                O(|Σ|log|Σ|)
 t.left := NIL
 t.right := NIL
 enQueue(Q,t,f[i])
for i = 1 to |Σ|-1 do
 t1 := DeQueue(Q)
 t2 := DeQueue(Q)
 t := new_tree_node()
 t.c := -
 t.fr := t1.fr + t2.fr
 t.left := t1
 t.right := t2
 enQueue(Q,t, t1.fr + t2.fr)
return DeQueue(Q)                            O(1)
```

Costo computazionale

$O(|\Sigma|\log|\Sigma|)$

2 DeQueue + 1 enQueue ad ogni iterazione

$O(\log|\Sigma|)$ ad operazione

|Σ|-1 iterazioni

$O(|\Sigma|\log|\Sigma|)$