

# Algoritmi e Strutture Dati

Foglio 7  
04/04/2025

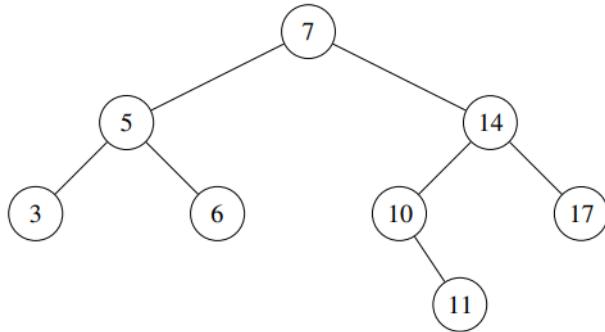
**Esercizio 1.** Disegnate degli alberi binari di ricerca di altezza 2, 3, 4, 5 e 6 per l'insieme delle chiavi  $\{1, 4, 5, 10, 16, 17, 21\}$ .

**Esercizio 2.** Dimostrate che, poiché l'ordinamento per confronti di  $n$  elementi richiede un tempo  $\Omega(n \log n)$  nel caso peggiore, allora un qualsiasi algoritmo basato sui confronti utilizzato per costruire un albero binario di ricerca con  $n$  elementi richiede un tempo  $\Omega(n \log n)$  nel caso peggiore.

**Esercizio 3.** Sia  $T$  un albero binario di ricerca (con chiavi tutte diverse) e sia  $x$  un nodo foglia e  $y$  suo padre. Dimostrate che o  $y.key$  è la più piccola chiave di  $T$  che è maggiore di  $x.key$ , oppure è la più grande chiave di  $T$  che è minore di  $x.key$ .

**Esercizio 4.** Potete ordinare un insieme di  $n$  numeri costruendo prima un albero binario di ricerca che contiene questi numeri e poi stampando i numeri con una dfs eseguita in-order. Quali sono i tempi di esecuzione nel caso peggiore e nel caso migliore per questo algoritmo di ordinamento?

**Esercizio 5.** Disegnate l'albero binario di ricerca ottenuto dall'albero in figura dopo la cancellazione della radice.

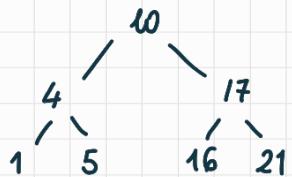


**Esercizio 6.** Disegnate gli alberi rosso-neri che si ottengono dopo aver inserito in successione le chiavi 41, 38, 31, 12, 19, 8 in un albero rosso-nero inizialmente vuoto.

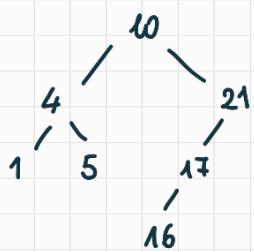
**Esercizio 7.** Sia  $T$  un albero rosso-nero contenente  $n$  chiavi distinte. Fornite un algoritmo che, dato un intero  $k$ , restituisce la chiave  $x$  di  $T$  che minimizza  $|k - x|$ . Il tempo di esecuzione deve essere  $O(\log n)$  nel caso peggiore.

**Esercizio 1.** Disegnate degli alberi binari di ricerca di altezza 2, 3, 4, 5 e 6 per l'insieme delle chiavi {1, 4, 5, 10, 16, 17, 21}.

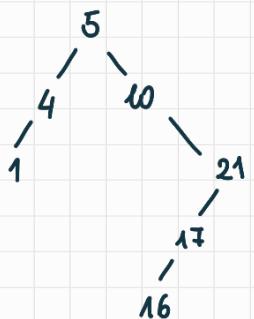
- o CASO  $h=2$  (OK, num. massimo di chiavi è  $2^{h+1}-1 \rightarrow 3-1 = 7$ )



- o CASO  $h=3$



- o CASO  $h=4$



**Esercizio 2.** Dimostrate che, poiché l'ordinamento per confronti di  $n$  elementi richiede un tempo  $\Omega(n \log n)$  nel caso peggiore, allora un qualsiasi algoritmo basato sui confronti utilizzato per costruire un albero binario di ricerca con  $n$  elementi richiede un tempo  $\Omega(n \log n)$  nel caso peggiore.

- Supponiamo che il tempo di esecuzione per costruire un BST sia  $\Omega(f)$  caso peggiore, utilizzando i confronti:  $\exists n_0, c > 0 : T(n) \geq c f \quad \forall n \geq n_0$
- Posso effettuare un ordinamento delle chiavi semplicemente utilizzando una DFS.
- Il costo complessivo di questo algoritmo di ordinamento sarebbe, nel caso peggiore:

$$\underbrace{T(n)}_{\text{costruzione BST}} + \underbrace{\Theta(n)}_{\text{DFS}} = \underbrace{\Omega(n \log n)}_{\text{lower bound ordin. ai confronti}}$$

Ma quindi stiamo dicendo che

$$\exists \bar{n}, a, b > 0 : T(n) + an \geq cn \log n$$

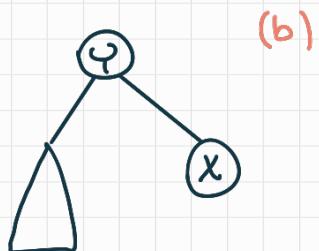
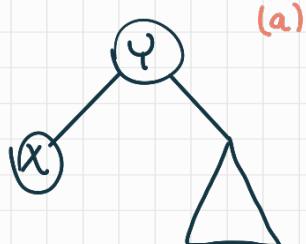
$$\hookrightarrow \text{ ``} : T(n) \geq cn \log n - an = \Omega(n \log n)$$

$$\Rightarrow T(n) = \Omega(n \log n).$$

Il che conclude la dimostrazione.

**Esercizio 3.** Sia  $T$  un albero binario di ricerca (con chiavi tutte diverse) e sia  $x$  un nodo foglia e  $y$  suo padre. Dimostrate che o  $y.key$  è la più piccola chiave di  $T$  che è maggiore di  $x.key$ , oppure è la più grande chiave di  $T$  che è minore di  $x.key$ .<sup>(1)</sup><sup>(2)</sup>

- Sono possibili due casi :



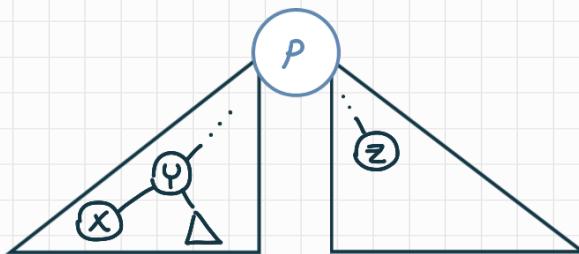
Quindi stiamo dicendo che nel caso (a) applichiamo (1) mentre (2) nel caso (b).

- Caso (a)

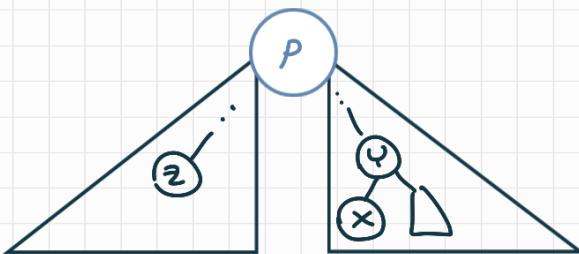
- Per assurdo supp. esista un nodo  $z$  per cui  
 $x.key < z.key < y.key$

Significa che  $z$  si trova fuori dal sottoalbero  $D_x$  di  $y$ .

- Prendiamo il primo antenato comune a  $x$  e  $z$ , chiamiamolo  $p$ . Per costruzione, ne del BST,  $x$  e  $z$  si trovano in due sottoalberi diversi di  $p$  (se fossero nello stesso si avrebbe  $z.key > y.key$  o  $z.key < x.key$ )
- Se  $z$  e' in quello  $S_x$  allora sicuramente  $z$  e' in quello  $D_x$   
 $\Rightarrow$  contraddizione, si avrebbe  $z.key > y.key > x.key$



- Viceversa, se  $z$  e' in quello  $D_x$ ,  $z$  e' in quello  $S_x$   
 $\Rightarrow$  contraddizione, si avrebbe  $z.key < y.key$



- Caso (b) e' analogo .

□

**Esercizio 4.** Potete ordinare un insieme di  $n$  numeri costruendo prima un albero binario di ricerca che contiene questi numeri e poi stampando i numeri con una dfs eseguita in-order. Quali sono i tempi di esecuzione nel caso peggiore e nel caso migliore per questo algoritmo di ordinamento?

BST - ORDER (A) :

```
T = BST-BUILD(A)
// allocate memory for vec. B
pos = 1
BST-ORDER-REC(T.root, B, pos)
// end BST-ORDER
```



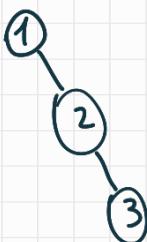
$\Theta(n)$

BST-ORDER-REC(node, B, pos) :

```
if node == NIL
    return pos
pos = BST-ORDER-REC(node.left, B, pos)
B[pos] = node.value
pos = pos + 1
pos = BST-ORDER-REC(node.right, B, pos)
return pos
// end BST-ORDER-REC
```

- C'è distinzione tra caso migliore e peggiore ?

1)



$$T_1(n) = \sum_{i=0}^{n-1} i = \Theta(n^2)$$

~~⊗~~

○

○

○

○

1

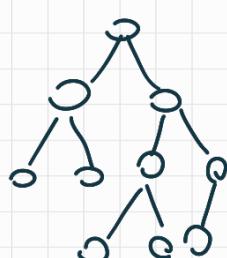
2

3

4

5

2)

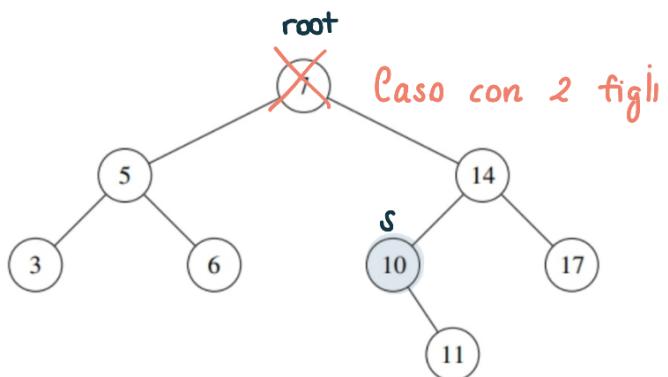


$$T_2(n) = \sum_{i=1}^n \log(i) \leq \sum_{i=1}^n \log(n) = n \log(n).$$

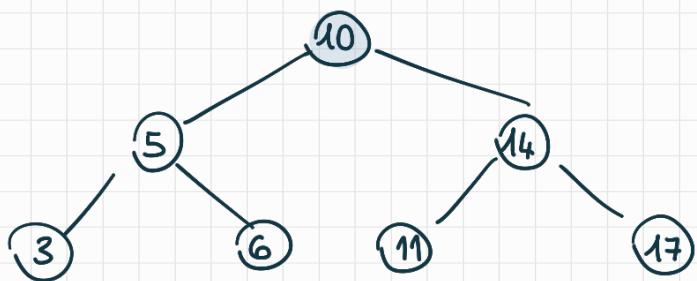
$$h(i) = \log(i)$$

$$\log 1 < \log 2 < \log 3 < \dots < \log(n-1) < \log(n)$$

**Esercizio 5.** Disegnate l'albero binario di ricerca ottenuto dall'albero in figura dopo la cancellazione della radice.



- Step 1: trovo successore di root : 10
- Step 2: link del sottoalbero dx di s al s.parent
- Step 3: copia di s.value in root.value



**Esercizio 6.** Disegnate gli alberi rosso-neri che si ottengono dopo aver inserito in successione le chiavi 41, 38, 31, 12, 19, 8 in un albero rosso-nero inizialmente vuoto.

o Possibili casi: come distinguerli facilmente?

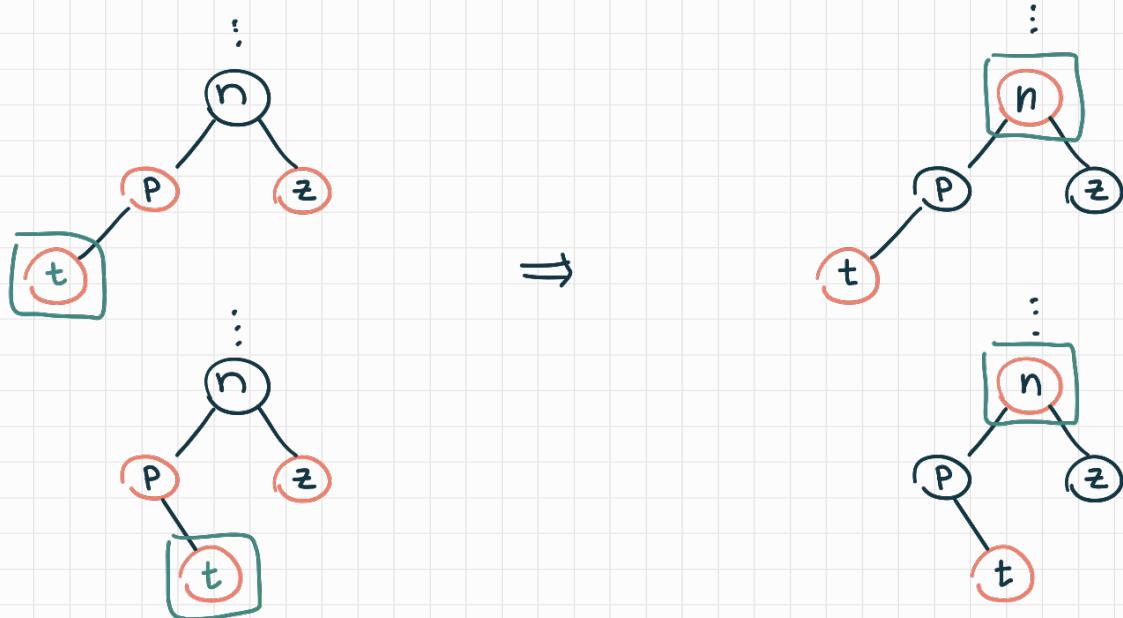
① t non ha padre  $\rightarrow$  primo nodo



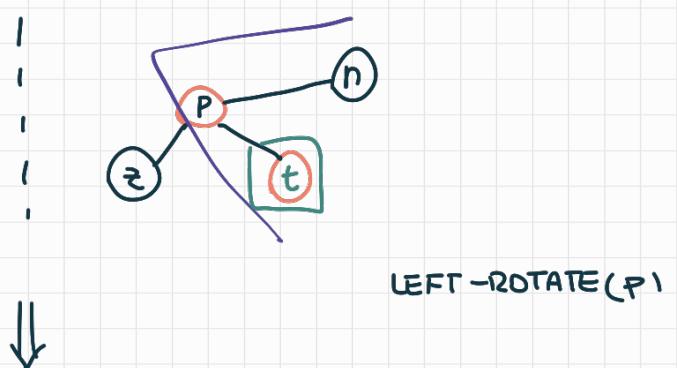
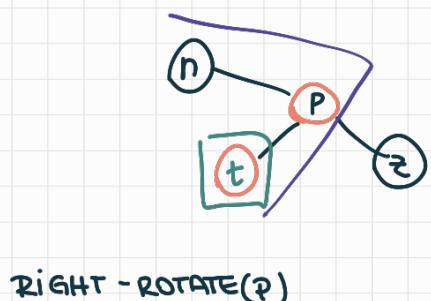
② p nero  $\rightarrow$  tutto ok

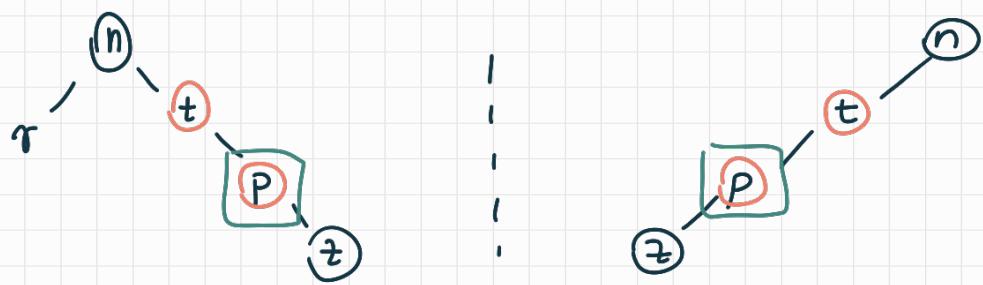
③ p, z rossi

disposizione indifferente



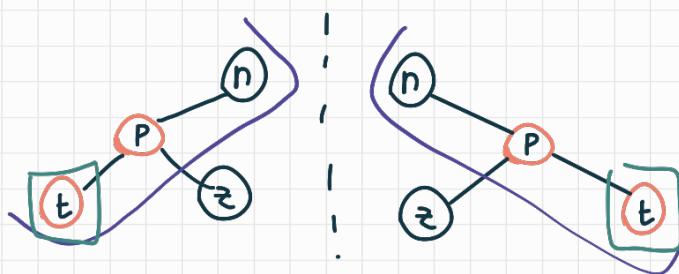
④ p rosso, z nero . (a) e (b) sono uno lo speculare dell'altro





ottengo il caso 5

(5)

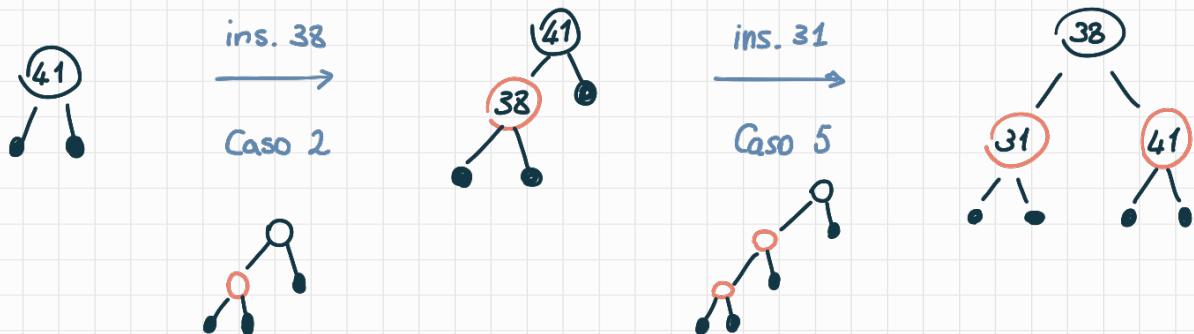


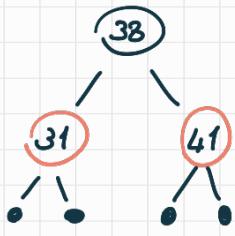
RIGHT-ROTATE( $p$ )

LEFT-ROTATE( $r$ )



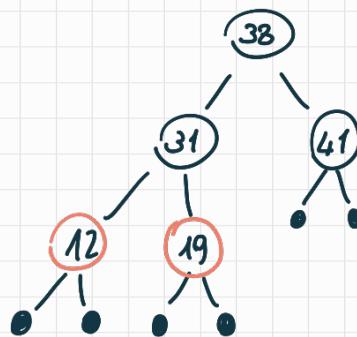
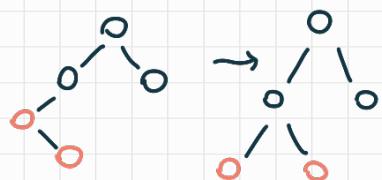
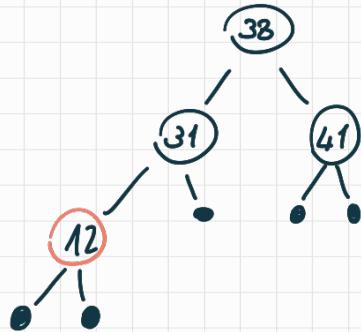
- Inseriamo le chiavi 41, 38, 31, 12, 19, 8



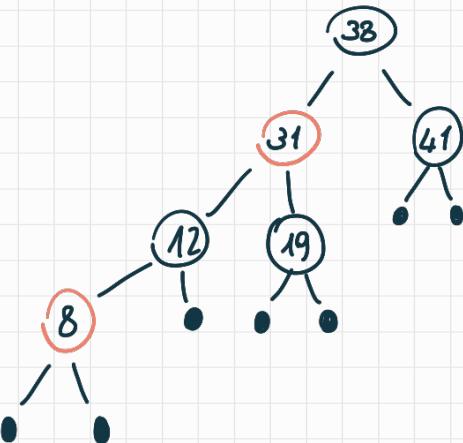


ins. 12  
Caso 3

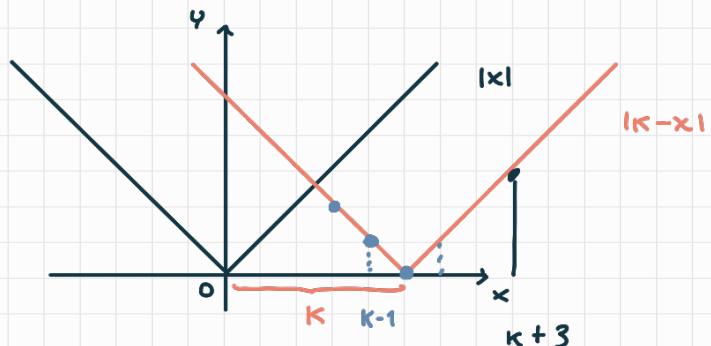
ins. 19  
Caso 4



ins. 8  
Caso 3



**Esercizio 7.** Sia  $T$  un albero rosso-nero contenente  $n$  chiavi distinte. Fornite un algoritmo che, dato un intero  $k$ , restituisce la chiave  $x$  di  $T$  che minimizza  $|k - x|$ . Il tempo di esecuzione deve essere  $O(\log n)$  nel caso peggiore.



- Per minimizzare  $|K - x|$ , posso controllare se in  $T$  ci sono i valori

$K, K \pm 1, K \pm 2, \dots$

ossia cerco:

- il più grande valore minore o uguale a  $K$   $\leftrightarrow$  aka il PREDECESSORE
- il più piccolo valore maggiore o uguale a  $K$   $\leftrightarrow$  aka il SUCCESSORE

- Dunque:

MINIMIZE-F( $K, T$ ) :

```
if (SEARCH( $K, T$ )) O(h)
    return  $K$ 
```

```
INSERT( $K, T$ )
 $P \leftarrow \text{PRED}(K, T)$  O(h)
 $S \leftarrow \text{SUCC}(K, T)$  O(h)
DELETE( $K, T$ )
```

```
return  $|K - P| < |K - S| ? P : S$ 
```