Lezione 7 aprile 2020

Tuesday, April 7, 2020 4:01 PM

Alberi

Algoritmi ricorsivi
tipo Divide et Impera
Alberi binari

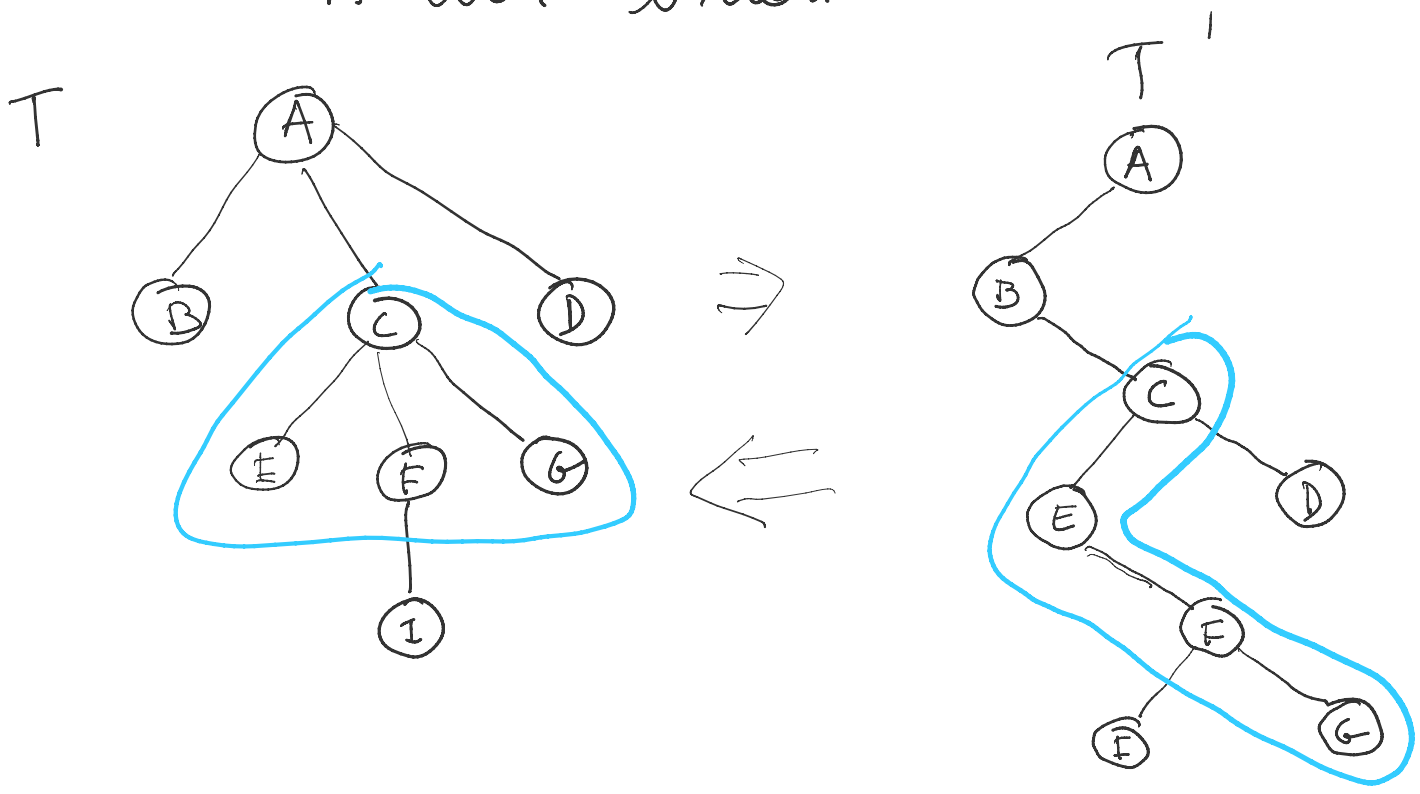


immagine binaria di T

radice di T ≡ radice di T'

primo figlio ≡ figlio sinistro

figlio i-esimo ≡ figlio destro in T' dell'i-1 figlio in T

Dimensione (T) ≡ Dimensione (T')

Altezza (T) ? Altezza (T) calcolata su T' ?

Visita anticipata (T) (PREVISITA)

1) esamina la radice
2) Visita il suo figlio in ordine anticipato
⋮

n+1) Visita l'r-esimo figlio di ord. anticipato

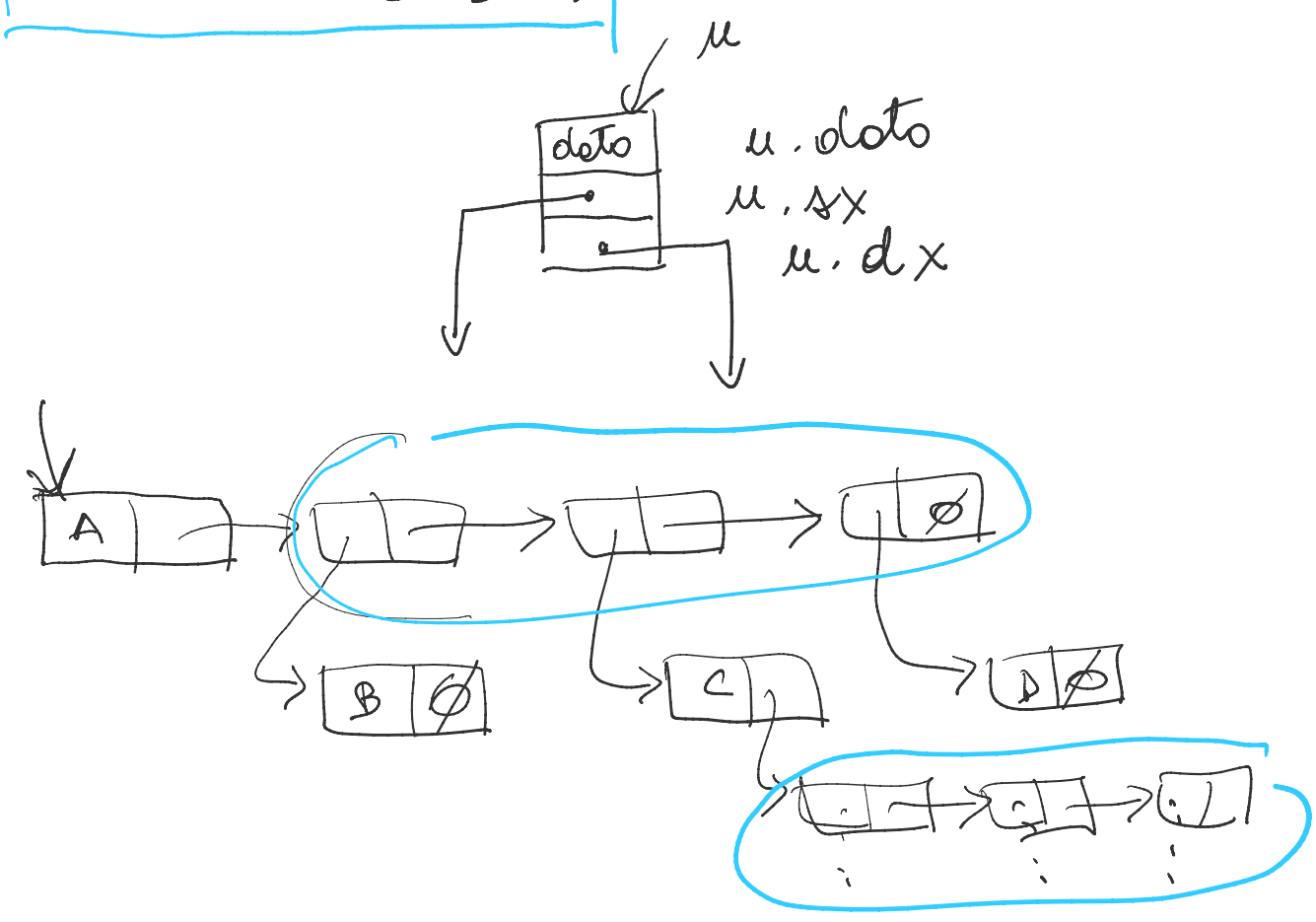
A B C E F I G D su T
A B C E F I G D su T' } corrispondono

Post visita

B E I F G C D A su T
I G F E D C B A su T'

Infisita su T' (solo per alberi binari)

B E I F G C D A



Alberi binari

per rendere efficienti le operazioni del

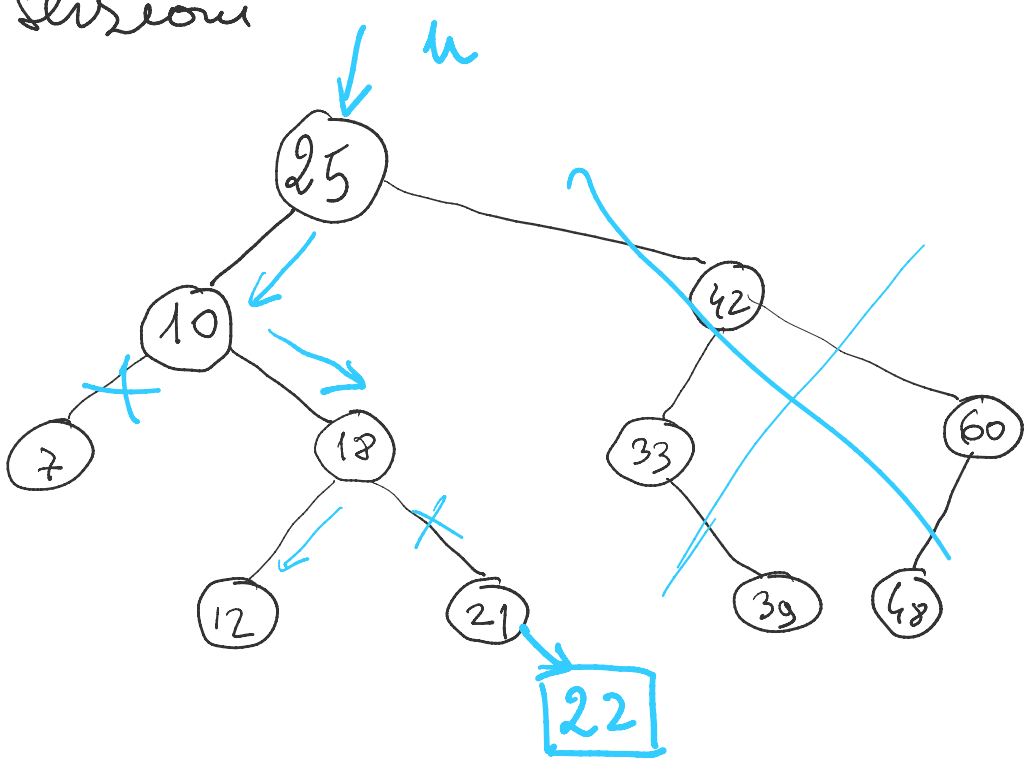
Dizionario
- ricerca(k)
- inserzione
- cancellazione

Heap
- ricerca max
- Enqueue
- Dequeue

Alberi binari di ricerca

- alberi binari qualsiasi
- per ogni nodo vale la proprietà:
  - le chiavi del sottoalbero sinistro sono minori della chiave del nodo
  - le chiavi del sottoalbero destro sono maggiori della chiave del nodo

? Perché non un array + ricerca binaria?
Struttura fissa non richiede a sequenze di inserzioni



Ricerca(K) K = 12 → K ∈ T K = 22 → K ∉ T

```
Ricerca ABR (u, K);
if (u == NULL) return NULL;
   else { if (u->dato == K) return u;
          else if (K < u.dato)
                  return Ricerca ABR(u->sx, K);
               else return Ricerca ABR(u->dx, K);
   }
```

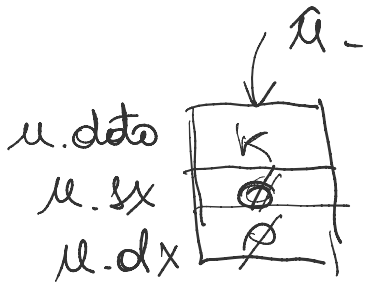
Complessità ?

Ricerca $O(h)$ h = altezza dell'ABR

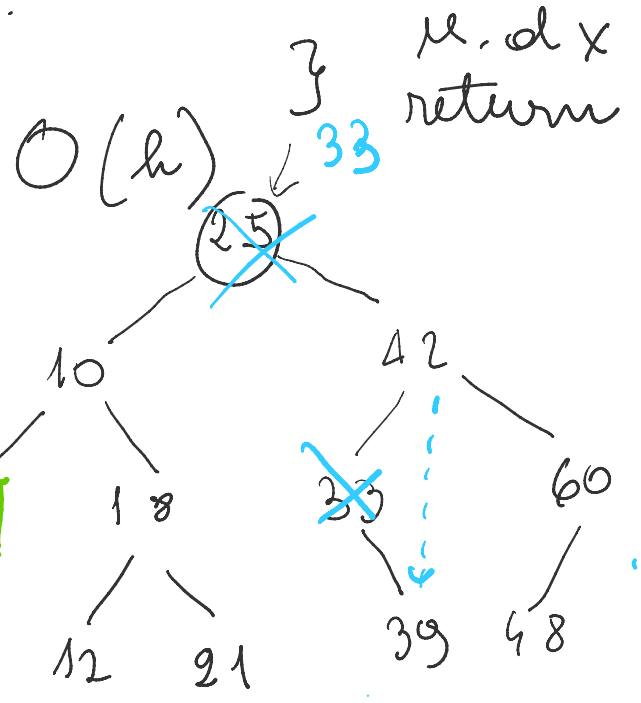
Inserzione (u, K) Ricerca di K
inserzione al punto di terminazione
$O(h)$

Cancellazione (K) $O(h)$

```
Inserisci (u, K):
if (u == Null) {
        u = NuovoNodo(u);
        u.sx = u.dx = Null
        u.dato = K;
} else if (K < u.dato)
        u.sx = Inserisci (u.sx, K);
else
        u.dx = Inserisci(u.dx, K);
return u;
}
```



$O(h)$



cancella 25

- precedente 21
- successivo 33

Cancellazione (K)

K foglia — assegnare $\emptyset$ al puntatore dx o sx del padre

K ha un puntatore $= \emptyset$ — bypass

K ha entrambi i puntatori $\neq \emptyset$

selezioniamo il successivo di K
≡ min del sottoalbero destro di K
ha sicuramente il sott. sinistro $= \emptyset$

- può sostituire K
- può essere eliminato facilmente

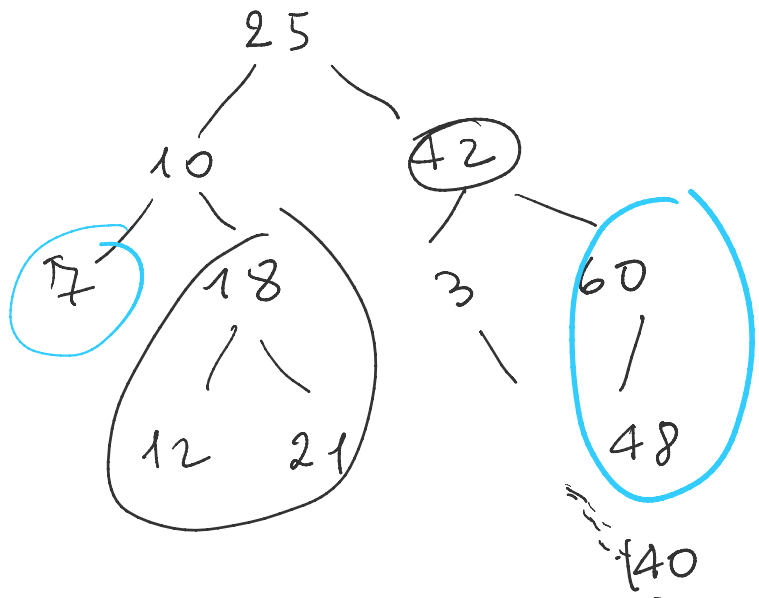
- cerca il successivo di K
- eliminalo dall'albero
- scambialo con K.

$O(h)$

Albero binario di ricerca

- ricerca $O(h)$
- inserzione $O(h)$
- cancellazione $O(h)$

ORDINAMENTO ≡ INVISITA



Succ (39)

$u.dx = \emptyset$

succ(39)=42

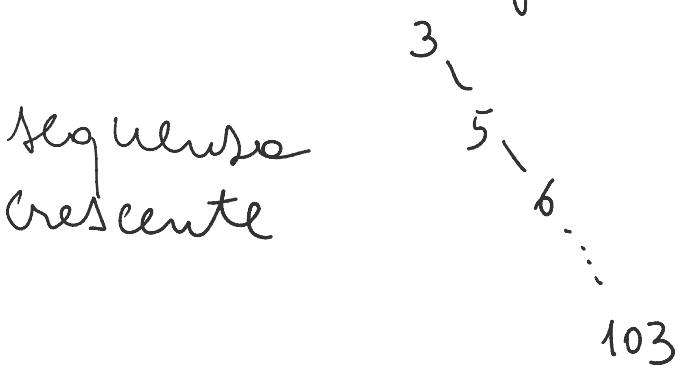
7 10 12 18 21 25 33 39 42 8 60

$T(n) = T(n_s) + T(n_d) + \quad (1) = \Theta(n)$

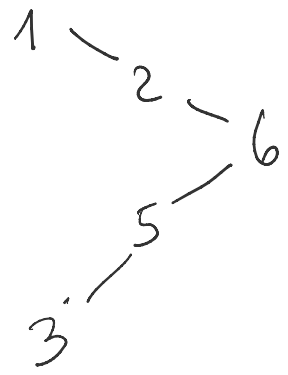
- min $O(h)$
- max $O(h)$
- prec (K) $O(h)$
- succ (K) durante la ri di K ci ricordiamo l'eventuale successivo (l'el. a max profondità) per cui siamo scesi a sinistra

Se $u.dx = NULL$ l'ho trovato altrimenti cerco il min del sottoalbero destro di K.

h ? Se l'albero è costruito per successive inserzioni dipende dai dati di input.

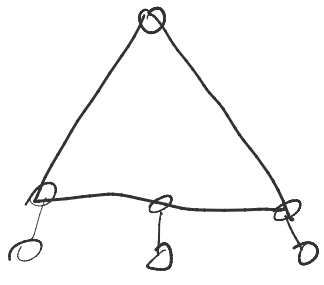


$h = \Theta(n)$



tenuti $2^i g - 2^{ag}$

$h = \Theta(n)$



$$\Theta(\log n) \leq h \leq \Theta(n)$$

Caso pessimo è molto brutto

Caso medio $h = O(\log n)$ (omen

Per avere la sicurezza di fare operazioni in $O(\log n)$ dobbiamo tenere sotto controllo l'altezza max raggiunta dagli alberi

Alberi binari di ricerca bilanciati in altezza

- Alberi 2-3 $\Rightarrow$ B-alberi
- Alberi rosso-neri
- Alberi AVL
  Adelson-Velsky
  Landis