Lezione 9 aprile

Thursday, April 9, 2020 4:02 PM

Alberi binari

algoritmi basati su DetI

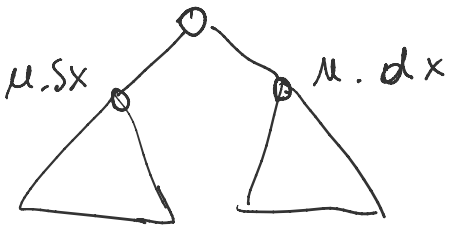
Dimensione
Altezza
Visite

} se Ricombinazione è $\Theta(1)$ ⇓ $\Theta(n)$

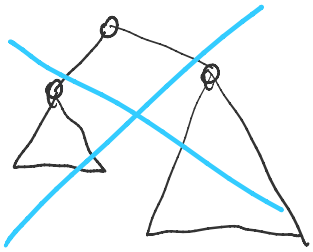
Applicazione di Post-visita

Stabilire se un alb. binario è perfettamente bilanciato

Input → puntatore all'albero
Output → booleano < true se l'alb è bil
false altrimenti



il sott. alb sin e destro devono perfettamente bilanciati e avere stessa altezza



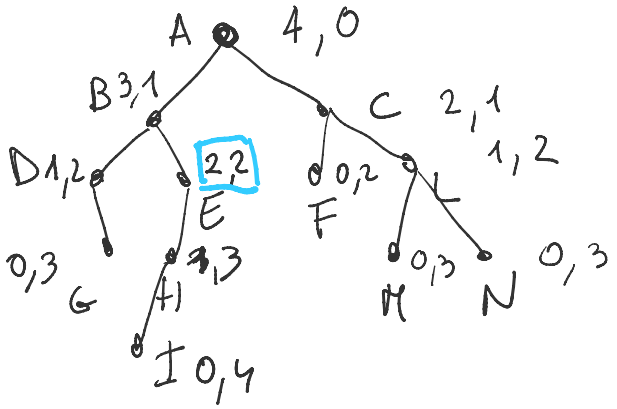
Scrivere una procedura che dato un alb. binario di puntatore u, stampi tutti i nodi cardine.

Def. un nodo x è detto cardine se

$$h(x) = pr(x)$$

$h(x)$ è la max distanza del nodo x dalle foglie più lontane

$pr(x)$ è la distanza dalla radice



altezza = dal basso verso l'alto

Bottom-up

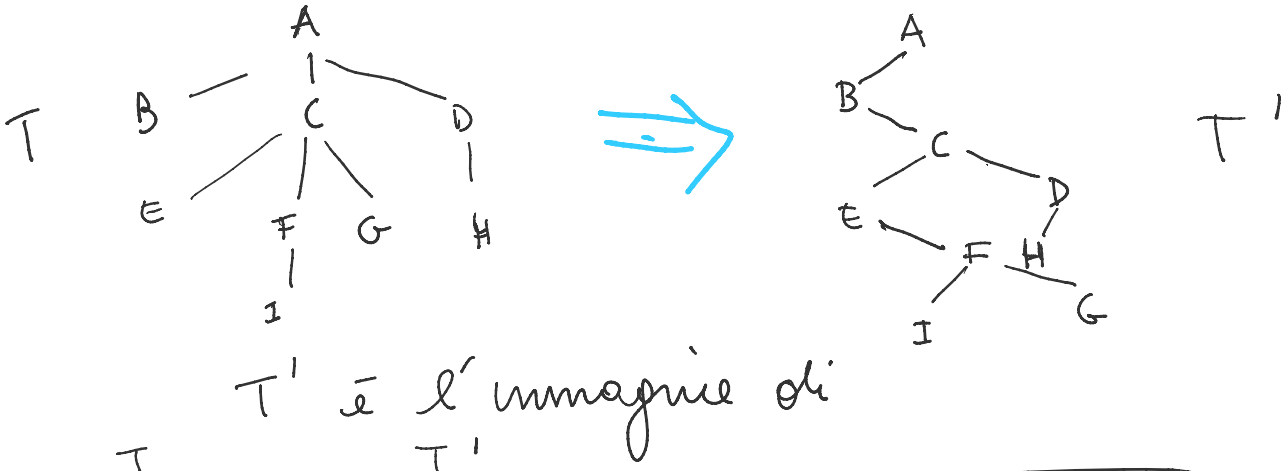
profondità: dall'alto verso il basso

Top-down

TRASFORMIAMO GLI ALBERI ORDINARI

↓

ALBERI BINARI



T' è l'immagine di T

T → T'

radice → radice

1° figlio → figlio sinistro

i-esimo figlio → figlio destro dell'(i-1)-esimo figlio in T'

PRE-Visita in alberi ordinati

1) esamino la radice
2) visito il primo figlio in PRE-order
⋮ ⋮
n+1) = l'ennesimo =

A B C E F I G D H — Previsita in T ≡
A B C E F I G D H — Previsita in T'

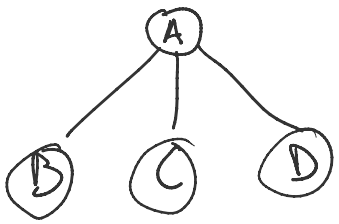
B E I F G C H D A — Post Visita in T ≠
I G F E H D C B A — Post Visita in T'

In Visita in T'

B E I F G C H D A

Visita simmetrica in T' ≡ PostVisita in T

alberi ordinati → ordinamento tra i figli di ciascun nodo



non esiste limite superiore al numero di figli.

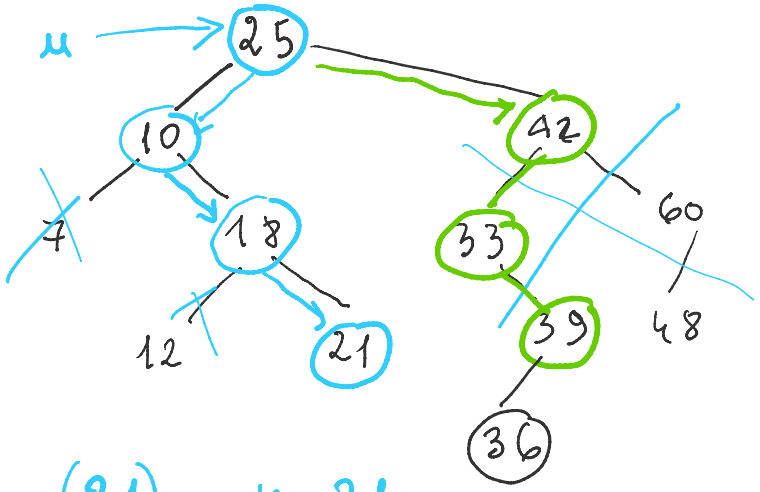
Alberi binari per rappresentare un dizionario e le sue operazioni

Dizionario insieme dinamico: varia nel tempo
- inserzione
- cancellazione
- ricerca

elemento: chiave, dati satellite

alberi binari di ricerca ABR
stile ricerca binaria

Per ogni chiave K:
- le chiavi ∈ sott. sin sono < K
- " ∈ " des " > K



ABR

Ricerca (21) K=21

Ricerca (36) → ricerca senza successo

I confronti nell'alg. di ricerca sono effettuati su un percorso radice - foglie

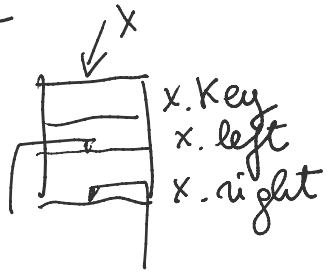
- h = altezza dell'ABR

- Ricerca è O(h)
- Inserzione è O(h)

Inserzione (36) - Ricerca (36)
effettuare l'inserzione nella posizione dove termina la ricerca

ITERATIVE-TREE-SEARCH(x, K):

```
while (x != NULL) && (k != x.key) {
    if k < x.key   x = x.left;
    else x = x.right;
} return x
```



Al termine: $k \in T$ se $x \neq NULL$
$k \notin T$ se $x = NULL$

```
TREE-INSERT (T, z);
y = NULL;
x = T.root;
while (x != NULL) {
    y = x;
    if (z.key < x.key) x = x.left;
    else x = x.right;
}
if y == NULL  T.root = z
else if (z.key < y.key) {
    y.left = z;
else
    y.right = z;
}
```

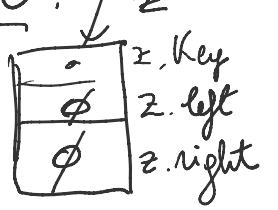
$O(h)$

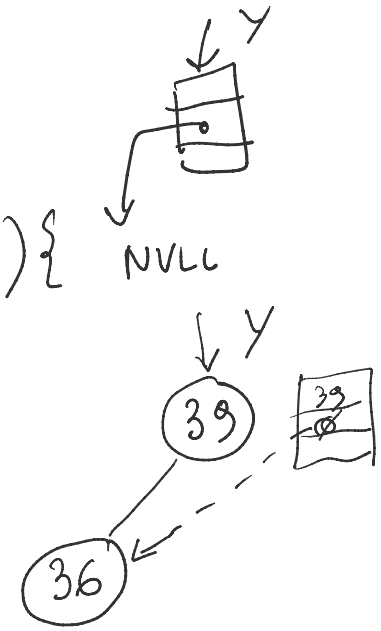
$\Theta(1)$

x = T.root

y = sta un passo indietro a x

right:

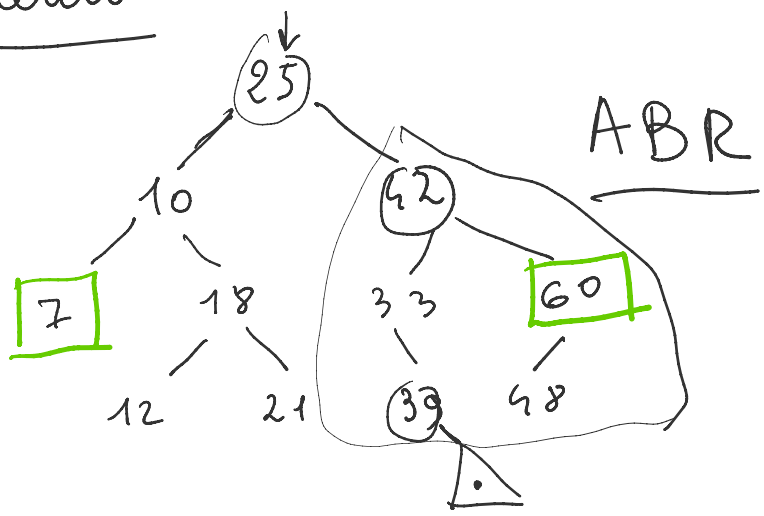




z.key $\notin$ T è già stato creato il nuovo nodo di puntatore z

Inserzione $O(h)$

Ordinamento



7 10 12 18 21 25 33 39 42 48 60

INVISITA ≡ sequenza ordinata

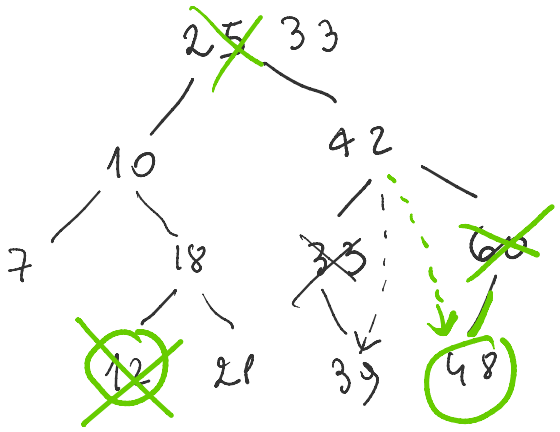
Complessità di una visita $\Theta(n)$

- min $O(h)$
- max $O(h)$
- predecessore $O(h)$
- successore (x): 2 casi $O(h)$

1) il puntatore destro di $K \neq NULL$
è il minimo del sott. destro

2) il puntatore destro di $K = NULL$
(il successore sta più in alto di K nell'albero).
il successore è l'ultimo nodo nel percorso radice - K in cui sono sceso a sinistra.

Cancellazione (K)



succ(25) = 33

k foglia → facile

k con un puntatore = NULL → facile

k ha 2 puntori ≠ NULL → diff-cile

si elimina pred(k) o succ(k)
e si sostituisce a k.

succ(k) = il min del sottoalb. destro(k)
ha il punt. sin = NULL
= può sostituire k nell'albero

Cancellazione $O(h)$

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

alb. degenere