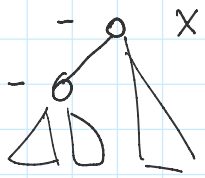
Lezione 17 aprile

Friday, April 17, 2020 11:01 AM

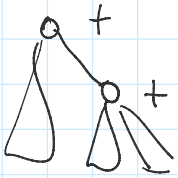
Rotazioni per AVL

fattore di bilancio $\begin{cases} -1 \\ 0 \\ +1 \end{cases}$

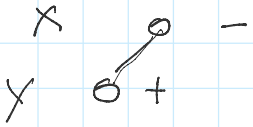
x nodo critico = il nodo a max profondità che si è sbilanciati



SS

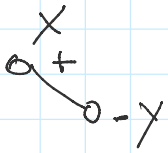


DD

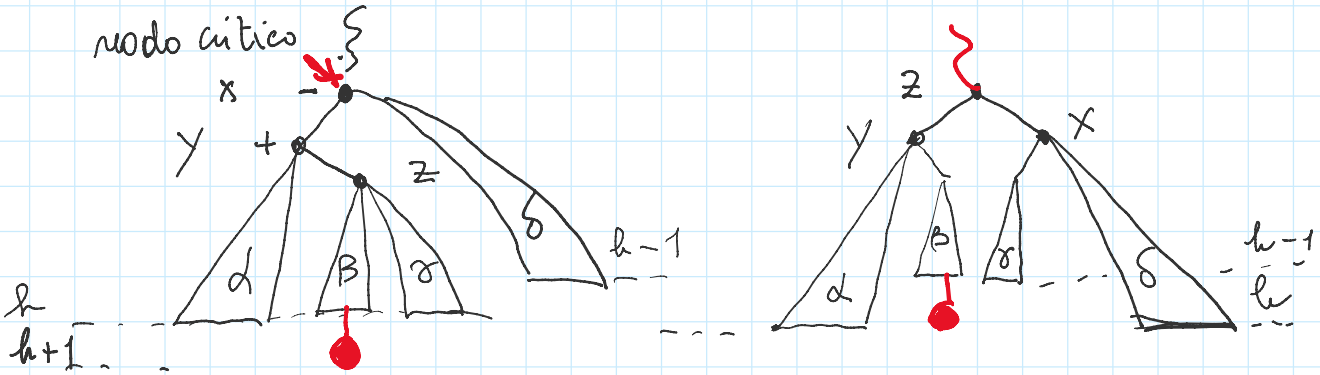


SD

rotazione doppia



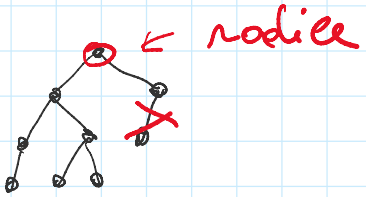
DS



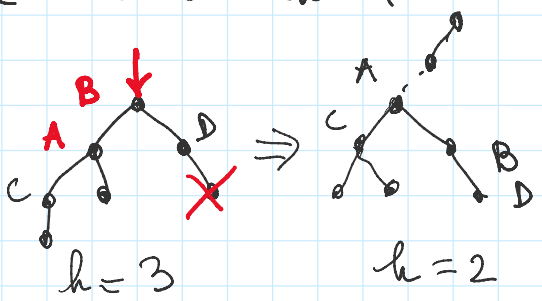
Complessità: $\Theta(1)$ operazioni per la sistemazione dei puntatori

Inserzione: Ricerca $O(\log n)$
Inserzione $\Theta(1)$
Rotazione $\Theta(1)$
} $O(\log n)$

Cancellazione



- Si esegue una rotazione opportuna
- Si ripristina il bilanciamento del sottoalbero attraverso rotazioni.
- Non è detto che l'altezza del sottoalbero in cui si opera rimanga la stessa rispetto a prima della cancellazione

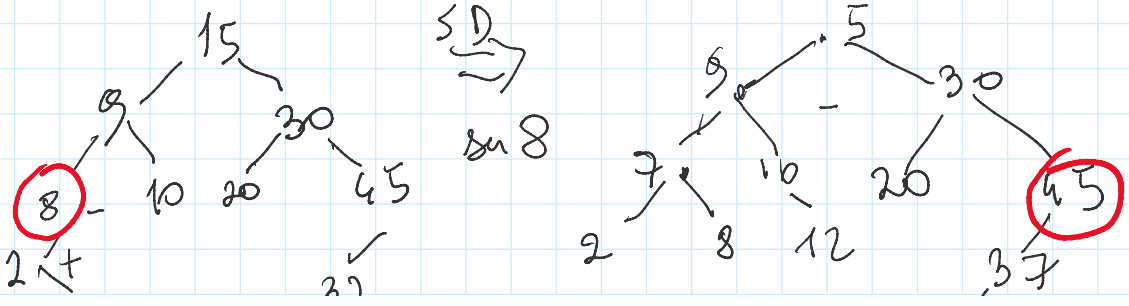


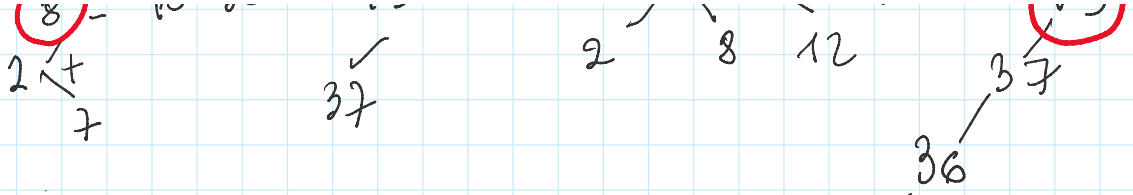
Cancellazione può richiedere fino a $O(\log n)$ rotazioni.

- Ricerca $O(\log n)$
- Cancellazione
- rotazioni $O(\log n)$

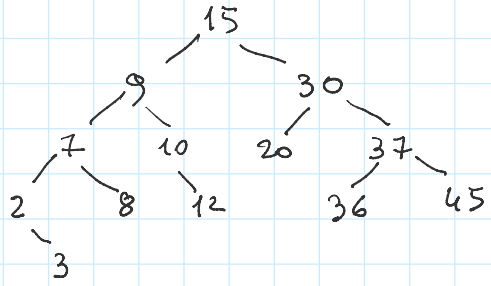
$O(\log n)$

15, 9, 30, 8, 10, 20, 45, 2, 37, 7, 12, 3, 36



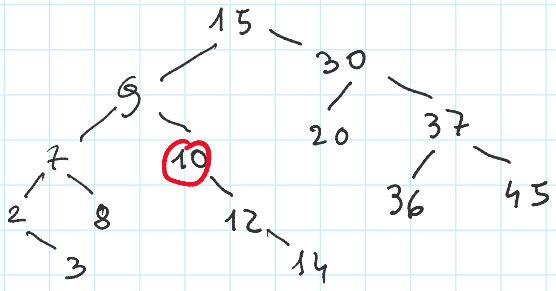


DD su 45

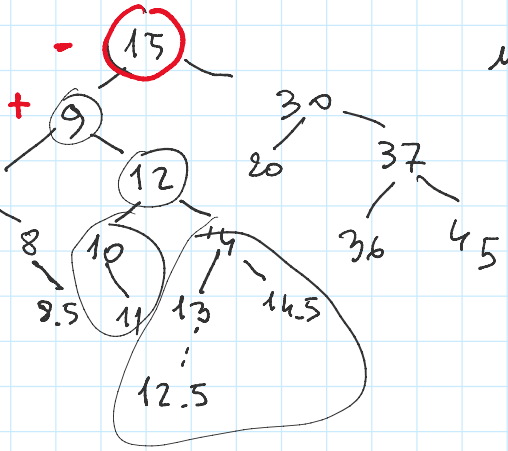


SS su 10

inserisci 14

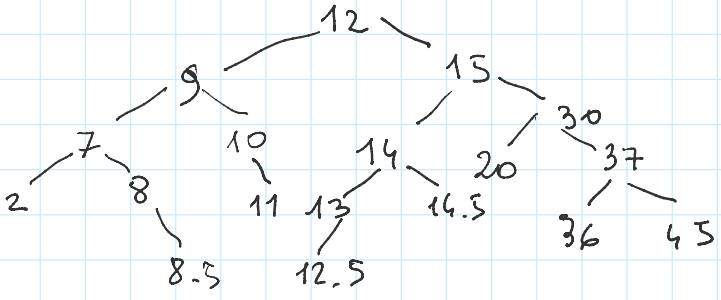


cancella 3



ris. 14.5
ris 11
ris. 8.5
ris. 12.5

rotazione SD su 15

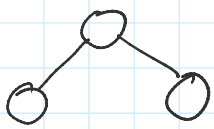


ISBIL?(u): output (<booleano, alt.>

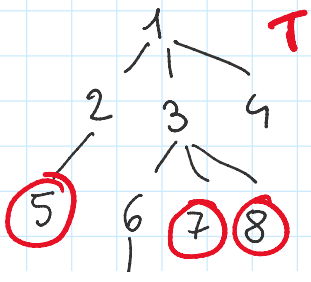
```
if (u == NULL) return <true, -1>;
  else {
    <bil_sx , alt_sx> = ISBIL?(u.sx);
    <bil_dx , alt_dx> = ISBIL?(u.dx);
    bil = bil_sx && bil_dx && (alt_sx == alt_dx)
    alt = max(alt_sx , alt_dx) + 1;
    return <bil , alt>;
  }
```

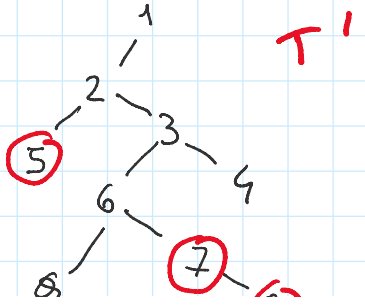
Trasformazioni ~~tra~~ da alberi ordinati a alberi binari.

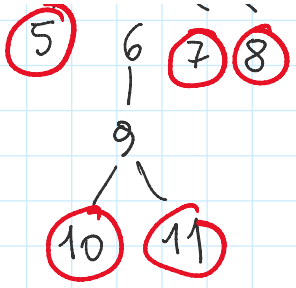
Dato un albero binario T' immagine di un albero ordinato T contare il numero di foglie dell'albero originale.

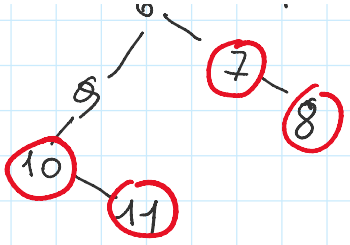


primo figlio a sinistra
fratello a destra









Un nodo in T' è foglia in T se il sottoalbero sin = ϕ

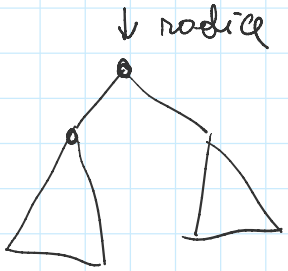
```
Foglie (u):
if (u == NULL) return 0;
if (u.sx == NULL) return 1 + Foglie(u.dx);
  else return Foglie(u.sx) + Foglie(u.dx);
```

$T(n) = O(n)$

Calcolare l'altezza di T, dato T' immagine binaria.

Stampare tutti i nodi cordine di un albero binario.

Nodo cordine = profondità = altezza

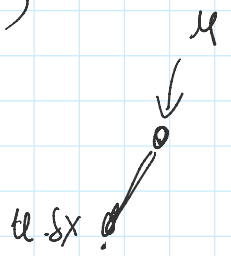


$\max(alt_{sin}, alt_{des}) + 1$

stfunct (u, 0)
↑ puntat. nodo ↑ prof.

output (altezza)

1ª chiamata Caroline (u, 0)

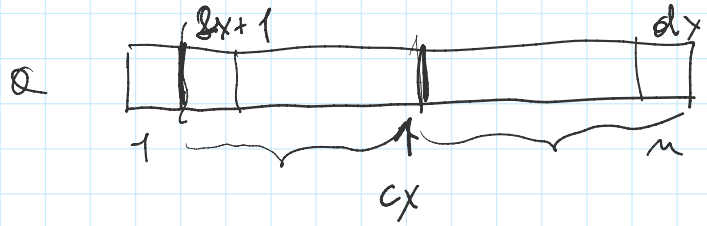


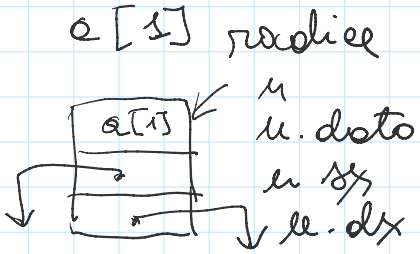

```
Caroline (u, p):
if (u == null) return -1;
  else {

  elts_x = Caroline (u.sx, p+1);
  eltd_x = Caroline (u.dx, p+1);
  altezza = max (elts_x, eltd_x) + 1;
  if (altezza == p) print u.dato;
  return u;
}
```

$T(n) = O(n)$

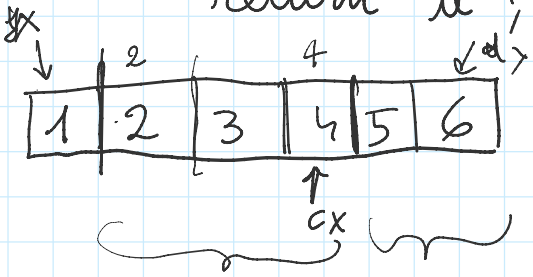
Dato un array a di n el. progettare un algoritmo che costruisca ricorsivamente in tempo $O(n)$ un albero binario bilanciato tale che a[i] sia l'i-esimo campo u.dato in ordine di visita anticipato.

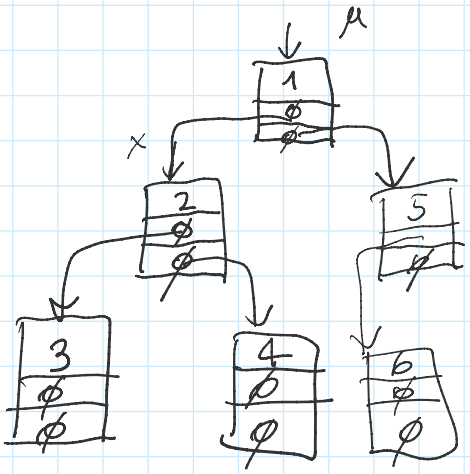




Anticipato (a, sx, dx):

```
Anticipato (a, sx, dx):
if (sx > dx) return null;
u = Nuovonodo ();
u.dato = a[sx]; u.sx = u.dx = NULL;
cx = (sx + dx + 1) / 2;
x = Anticipato (a, sx+1, cx);
u.sx = x;
y = Anticipato (a, cx+1, dx);
u.dx = y;
return u;
```





$T(n) = O(n)$