Alberi binari

Dato un insieme S ($n = |S|$), T si dice albero bin. se è una 3-partita della forma $S_L \cup \{r\} \cup S_R$ con $S_L = S_R = \emptyset$ (in questo caso si dice foglia) oppure r è radice di chi non è vuoto in $\{S_L, S_R\}$ e a sua volta un alb. bin. (def. ricorsiva).

→ a livello implem si crea una struct che alberi:

- r → radice
- sx → punt. a radice di S_L
- dx → " di S_R

event / NULL

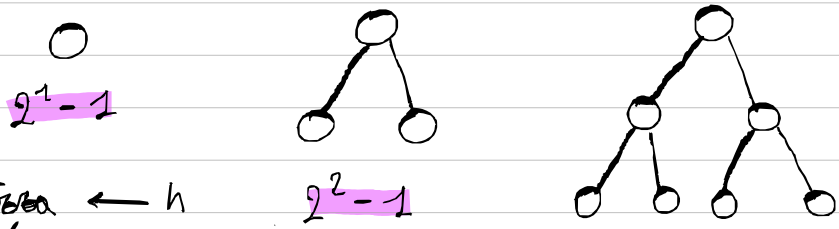
→ per convenzione, $S = \emptyset \rightarrow T = NULL$.

Definiamo ALTEZZA dell'albero il num. massimo di archi in un cammino radice - foglia contando solo da padre in figlio.

→ per calcolare la dimensione o l'altezza dell'albero si applica il paradigma del DIVIDE ET IMPERA — entrambi sono $O(n)$.

→ per calcolare l'altezza di ogni nodo (la "profondità" che ha nell'albero) si applica lo stesso procedimento in top-down (sempre $O(m)$).

Un albero binario si dice COMPLETAMENTE BILANCIATO se a ogni livello c'è il massimo numero di nodi e foglie.



altezza $\leftarrow h$

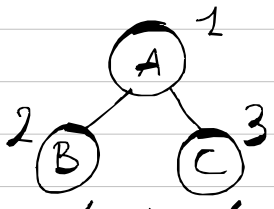
→ $\#\text{nodi} = \sum_{i=0} 2^i =$

$= 2^h - 1$

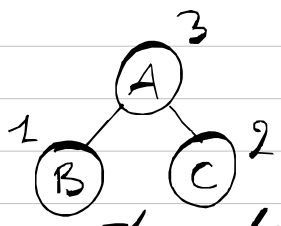
→ usando il DIVIDE ET IMPERA si può scrivere un algoritmo che verifica se l'albero è bilanciato e che calcola contemporaneamente l'altezza in $O(n)$.

Ci sono 4 tipi di visita su un albero binario:

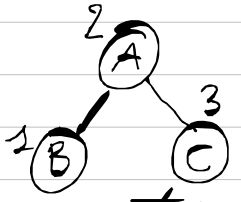
- **anticipato**: si stampa il nodo e si passa ricorsivamente alla visita di S_L e poi di S_R (la radice viene prima → anticipato).
- **posticipato**: si passa ricorsivamente a S_L, poi a S_R e solo dopo si stampa il padre (quindi "posticipato")
- **simmetrico**: prima S_L, poi la radice, poi S_R (il padre sta in mezzo → simmetrico)
- **ad ampiezza**: in ordine sulla profondità, da sinistra verso destra (si usano le code)



anticipato



posticipato



simmetrico

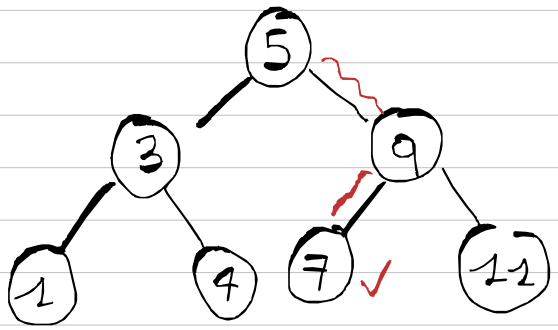
Un **albero binario di ricerca** (BST) è un albero binario in cui $S_L \leq M \leq S_R$ per ogni sottoalbero.

esempio se si vuole cercare il numero intero 1 contenuto in un array ordinato, si prende l'el centrale, lo si confronta e, se l'intero è $>$, si ripete l'alg nella "parte destra", se $<$ " sinistra, se $=$ ci si ferma, se non si può ripetere l'elem. non appartiene (**binary search**).

→ a livello di ricerca è come stare usando un BST.

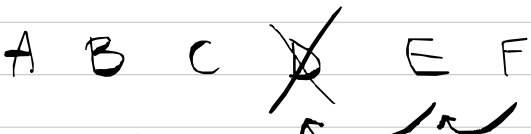
$A = [1\ 3\ 4\ 5\ 7\ 9\ 11]$

$m = 3$



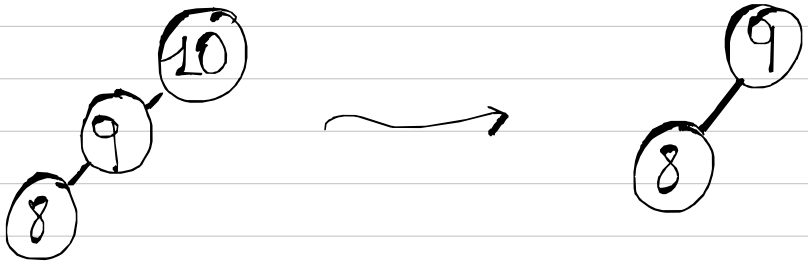
Questa tale impl. permette la ricerca in $O(\log n)$

→ l'inserimento di un nodo su un BST è molto semplice, mentre bisogna essere accorti sulla cancellazione: se la pensiamo in funzione del binary search è semplice capire cosa bisogna fare:

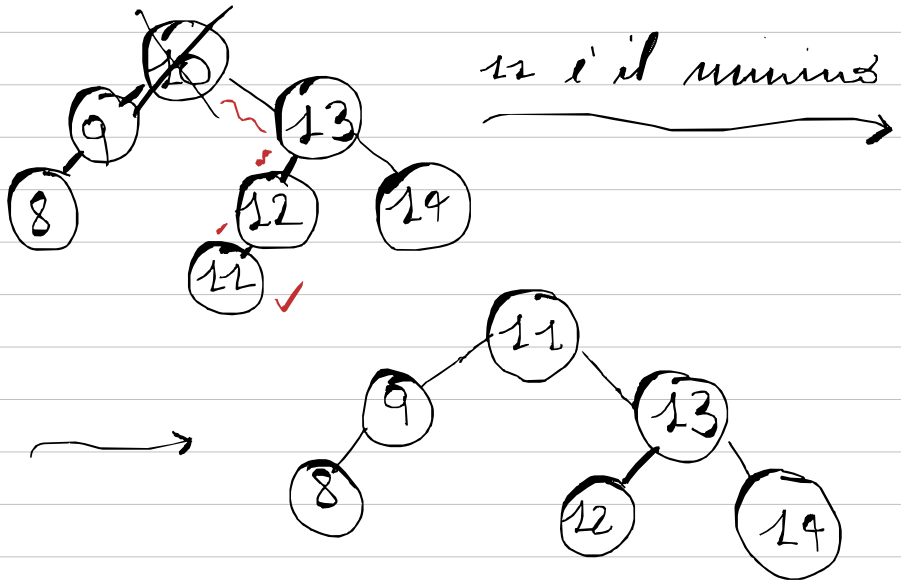


ovvero il minimo dei maggiori di D prende il suo posto.

Operativamente se A non ha figli destri, si sostituisce A come in una lista



altrimenti si prende il minimo di A.dx, come visto:



Bilanciamento di un albero

→ un albero binario può avere come minimo altezza $\log_2(n)$ — se è completo — e $n-1$ — se è "impostato a lista".

Un albero binario si dice BILANCIATO se la sua altezza è nell'ordine di $\log(n)$.

→ l'idea è che un BST che riceve un input random è molto probabilmente bilanciato.

L'analisi in qst caso si rifà a qll del QS randomizzato. Infatti l'altezza che si ottiene è esattam. il max dei num dei confr effett da una foglia (ogni confr sceglie la portà, o sx o dx).

Quindi... $E\left[\sum_j X_{ij}\right] = \sum_j E[X_{ij}] \leq$
$$\leq \sum_j \frac{2}{j-i+1} \sim O(\log(n)).$$

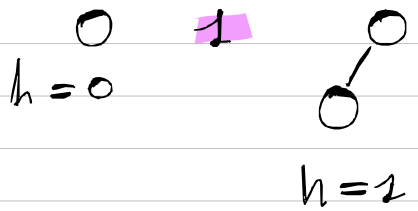
AVL Tree

→ è vero che nel caso medio il BST sarà bilanciato, ma vorremmo che h sia nell'ordine di $\log(n)$ anche nel caso pessimo; ci sono ~~tante~~ sol a qst. problema, una è rappr. dagli AVL Trees.

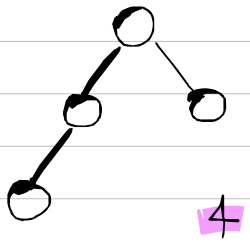
Def. Un albero si dice 1-bilanciato se $\forall$ u nodo $|h(u.sx) - h(u.dx)| \leq 1$ (ossia le alt. si discostano di al più 1).

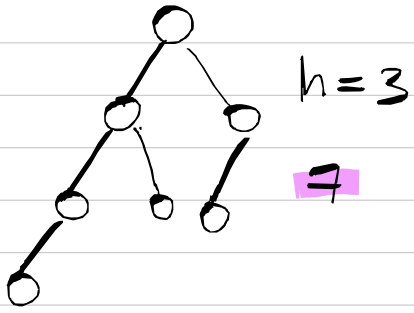
Prop. Un albero 1-bilanciato è bilanciato

→ si fa per induzione sugli alberi di Fibonacci:

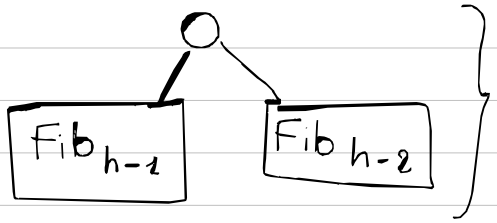


1-bilanciati minimali nel numero di nodi per una certa altezza





Si continua facendo:



Se n_h è il num di nodi di Fib_h, allora:

$$\begin{cases} n_h = 1 + n_{h-1} + n_{h-2} \\ n_0 = 1 \\ n_1 = 2 \end{cases}$$

In particolare $n_h = F_{n+3} + 1$, e quindi $n_h \geq c^h$ per una qlc c legata a φ.

Dunque $n \geq n_h \geq c^h \rightsquigarrow$

nodi in un 1-bal di alt. h

$\rightsquigarrow h = O(\log(n))$. ■

Def Chiamiamo AVL tree un BST 1-bilanciato.

→ la ricerca su un AVL funziona esattamente come su un BST.

→ la cancellazione: all'inizio si mette un "bollo di cancellato" sui nodi cancellati e quando diventano troppi (e.g. una prop. cost. di n), si ricostruisce l'AVL da zero (senza i nodi marcati).

L'inserimento è la parte più delicata e ingegnosa di un AVL.

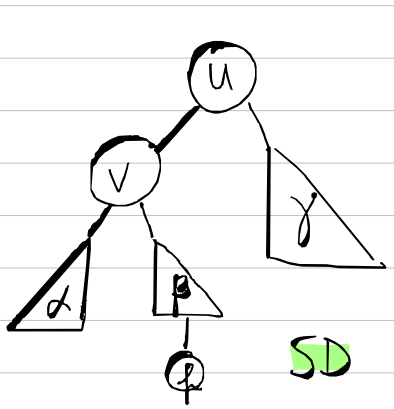
Si crea ancora una foglia come in un normale BST, e poi si effettuano delle op. su l'albero a partire da un nodo critico.

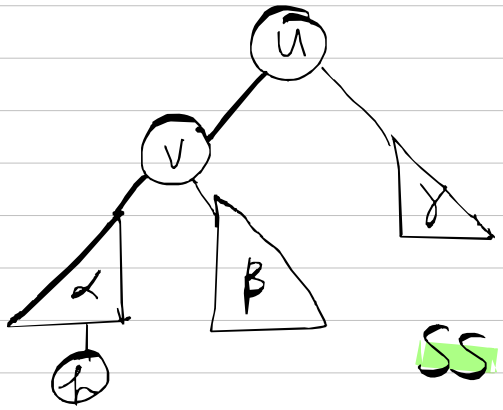
→ a livello di implementazione ogni nodo ha anche un attr. h — la sua altezza.

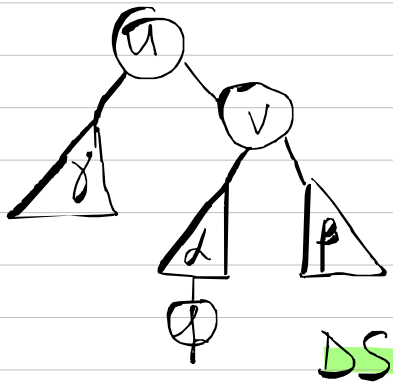
Def. Un nodo u si dice CRITICO risp. alla foglia f se:

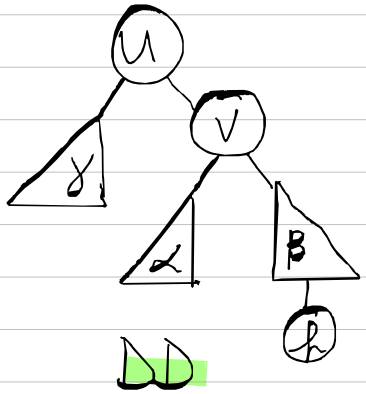
(i) è antenato di f
(ii) $|h(u.sx) - h(u.dx)| = 2$
(iii) è il nodo più vicino a f con tale proprietà

Possiamo dunque che u è il nodo critico di f, allora ci sono quattro situazioni (S sinistro, D destro):

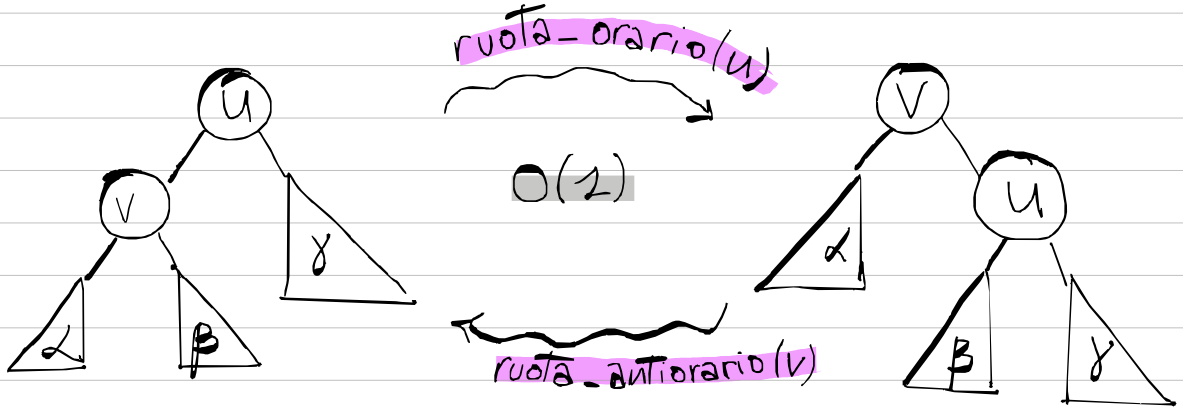








Date le seguenti due rotazioni si può bilanciare con una loro combinazione tutti i quattro casi:



- **SS**: ruota_orario(u)
- **SD**: ruota_antiorario(u.sx);
  ruota_orario(u)
- **DS**: ruota_orario(u.dx);
  ruota_antiorario(u)
- **DD**: ruota_antiorario(u.dx)

→ questo permette di effettuare l'inserimento in $O(\log(n))$.

a $O(1)$ si aggiunge la ricerca del nodo critico