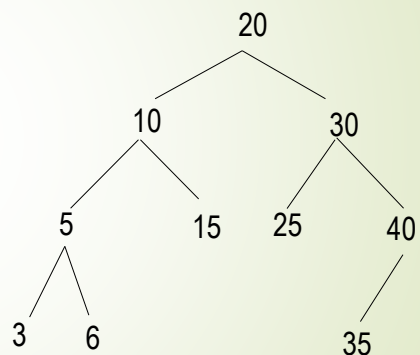


ADT Alberi binari di ricerca

- Utilizzati per la realizzazione di insiemi ordinati
- Operazioni efficienti di
 - ricerca
 - inserimento
 - cancellazione

Alberi binari di ricerca: definizione

- Se l'albero non è vuoto
 - ogni elemento del sottoalbero di sinistra precede ($<$) la radice
 - ogni elemento del sottoalbero di destra segue ($>$) la radice
 - i sottoalberi sinistro e destro sono alberi binari di ricerca



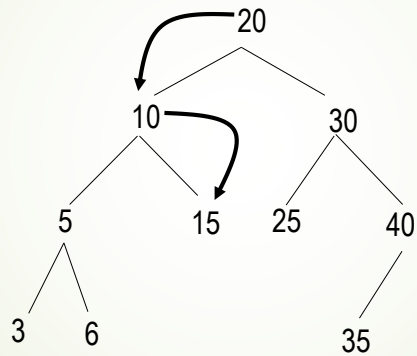
ADT: Binary Search Tree

Sintattica	Semantica
Nome del tipo: BST Tipi usati: Item, boolean	Dominio: $T = \text{nil} \mid T = \langle N, T1, T2 \rangle$ $N \in \text{NODO}$, T1 e T2 sono BST
$\text{newBST}() \rightarrow \text{BST}$	$\text{newBST}() \rightarrow T$ <ul style="list-style-type: none">Post: $T = \text{nil}$
$\text{isEmpty}(\text{BST}) \rightarrow \text{boolean}$ $\text{getLeft}(\text{BST}) \rightarrow \text{BST}$ $\text{getRight}(\text{BST}) \rightarrow \text{BST}$	$\text{isEmpty}(T) \rightarrow b$ <ul style="list-style-type: none">Post: se $T = \text{nil}$ allora $b = \text{true}$ altrimenti $b = \text{false}$
$\text{search}(\text{BST}, \text{Item}) \rightarrow \text{Item}$ $\text{min}(\text{BST}) \rightarrow \text{Item}$ $\text{max}(\text{BST}) \rightarrow \text{Item}$	$\text{search}(T, e) \rightarrow e'$ <ul style="list-style-type: none">Pre: $e \neq \text{nil}$Post: $e' = e$ se $e \in T$; $e' = \text{nil}$ altrimenti
$\text{insert}(\text{BST}, \text{Item}) \rightarrow \text{BST}$	$\text{insert}(T, e) \rightarrow T'$ <ul style="list-style-type: none">Post: T' contiene i nodi di T con l'aggiunta di e
$\text{delete}(\text{BST}, \text{Item}) \rightarrow \text{BST}$	$\text{delete}(T, e) \rightarrow T'$ <ul style="list-style-type: none">Pre: T non è vuotoPost: $T' = T - \{e\}$

Operazioni: search

- Se l'albero è vuoto allora restituisce null
- Se l'elemento cercato coincide con la radice dell'albero restituisce l'item della radice
- Se l'elemento cercato è minore della radice restituisce il risultato della ricerca dell'elemento nel sottoalbero sinistro
- Se l'elemento cercato è maggiore della radice restituisce il risultato della ricerca dell'elemento nel sottoalbero destro

Esempio: ricerca di 15



Operazioni: min (max)

- Algoritmo ricorsivo:
 - Se l'albero è vuoto allora restituisci null
 - Se non esiste un sottoalbero sinistro (destro), ritorna l'item associato alla radice
 - Se esiste un sottoalbero sinistro (destro) effettua la ricerca del minimo (massimo) nel sottoalbero sinistro (destro)
- Pseudocodice versione iterativa:

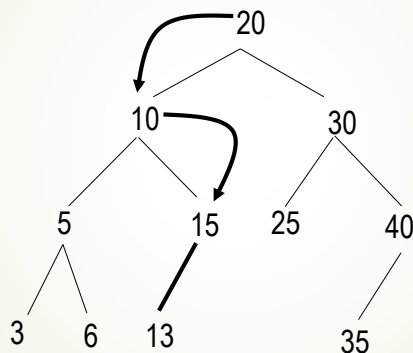
```
Tree_minimum(x)
  while (x.left != NULL)
    x = x.left;
  return x;
```

Operazioni: insert

► Inserimento di un elemento

- Se l'albero è vuoto allora crea un nuovo albero con un solo elemento
- Se l'albero non è vuoto
 - se l'elemento coincide con la radice non si fa niente (elemento già presente)
 - se l'elemento è minore della radice allora lo inserisce nel sottoalbero sinistro
 - se l'elemento è maggiore della radice allora lo inserisce nel sottoalbero destro

Esempio: inserimento di 13



✂ ADT Alberi binari di ricerca

✂ Cancellazione

✂ Analisi

✂ Alberi Bilanciati

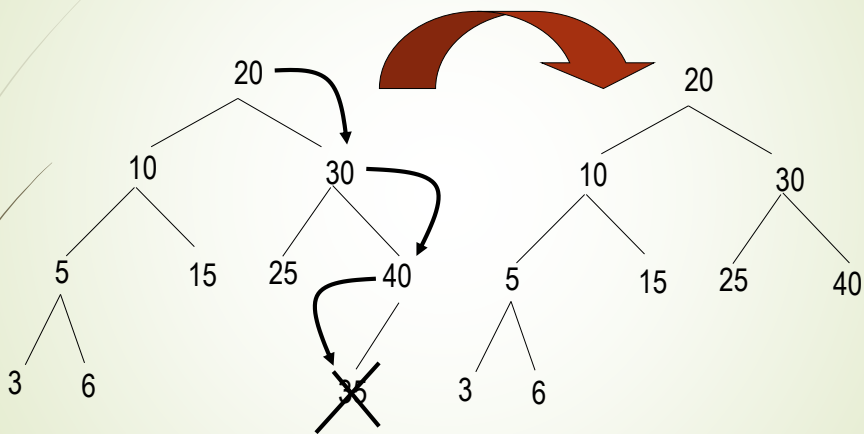
✂ Alberi AVL

✂ Heap

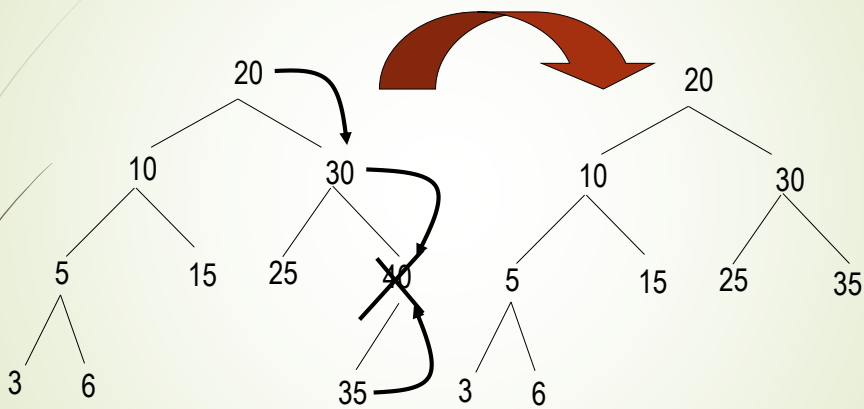
Operazioni: delete

- Si cerca ricorsivamente il nodo da rimuovere
- Trovato il nodo
 - Caso 1: se il nodo ha al più un solo sottoalbero di radice r
 - Si bypassa il nodo da rimuovere agganciando direttamente il suo unico sottoalbero al padre
 - Si rimuove il nodo

Esempio: Eliminazione di 35



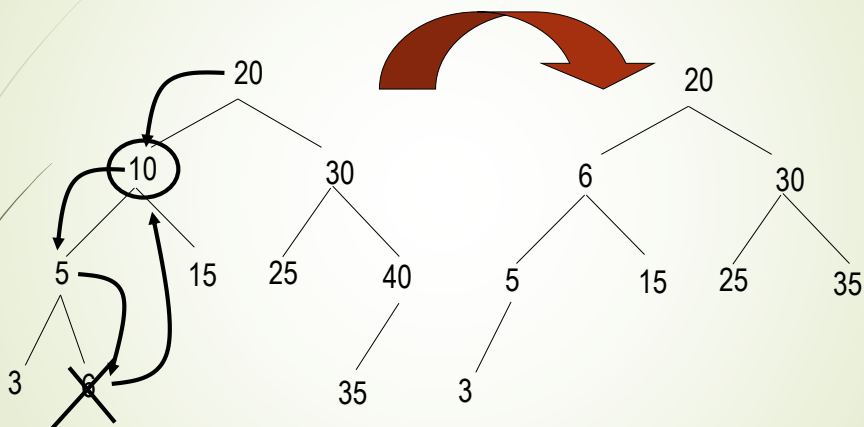
Esempio: Eliminazione di 40



Operazioni: delete

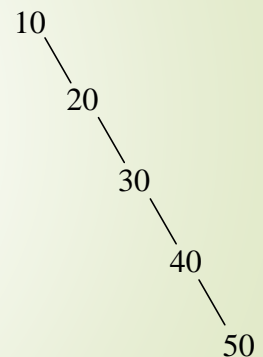
- Caso 2: il nodo ha entrambi i sottoalberi
 - Si sostituisce l'elemento da eliminare con il max nel sottoalbero sinistro (da notare che tale elemento non ha sottoalbero destro, la cui radice altrimenti sarebbe maggiore)
 - Alternativamente si cerca e si sostituisce con l'elemento minimo nel sottoalbero destro
 - Si chiama ricorsivamente la delete sul sottoalbero sinistro del nodo contenente l'elemento max
- L'albero risultante è un ABR

Esempio: Eliminazione di 10



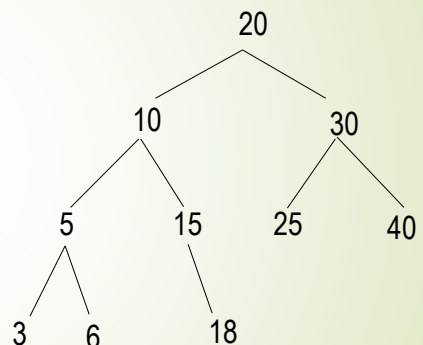
Complessità delle operazioni


- Le operazioni sull'albero binario di ricerca hanno complessità $O(h)$, dove h è l'altezza dell'albero.
 - Nel caso peggiore, il nodo da cercare (inserire o eliminare) si troverà a distanza h dalla radice
 - In un albero bilanciato con n nodi l'altezza dell'albero è $\log_2 n$
- Purtroppo la nostra implementazione non garantisce che l'albero sia bilanciato
 - Es: creare l'albero e inserire i nodi le cui etichette sono ordinate in modo crescente (10, 20, 30, 40, 50)



Alberi bilanciati e alberi Δ -bilanciati

- Un albero binario di ricerca si dice Δ -bilanciato se per ogni nodo la differenza (in valore assoluto) tra le altezze dei suoi due sottoalberi è minore o uguale a Δ
- Per $\Delta = 1$ si parla di alberi bilanciati





Alberi Δ -bilanciati

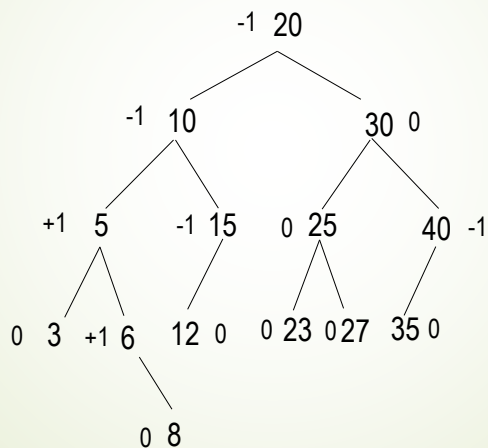
- Le operazioni sull'albero binario di ricerca hanno complessità logaritmica se l'albero è Δ bilanciato
- Si può dimostrare che l'altezza dell'albero è $\Delta + \log_2 n$



Alberi AVL

- Un esempio di alberi bilanciati sono gli alberi AVL
 - Dal nome dei suoi ideatori (Adel'son, Vel'skii e Landis)
- Per prevenire il non bilanciamento bisogna aggiungere un marcatore ad ogni nodo, che può assumere i seguenti valori:
 - -1 , se l'altezza del sottoalbero sinistro è maggiore (di 1) dell'altezza del sottoalbero destro
 - 0 , se l'altezza del sottoalbero sinistro è uguale all'altezza del sottoalbero destro
 - $+1$, se l'altezza del sottoalbero sinistro è minore (di 1) dell'altezza del sottoalbero destro

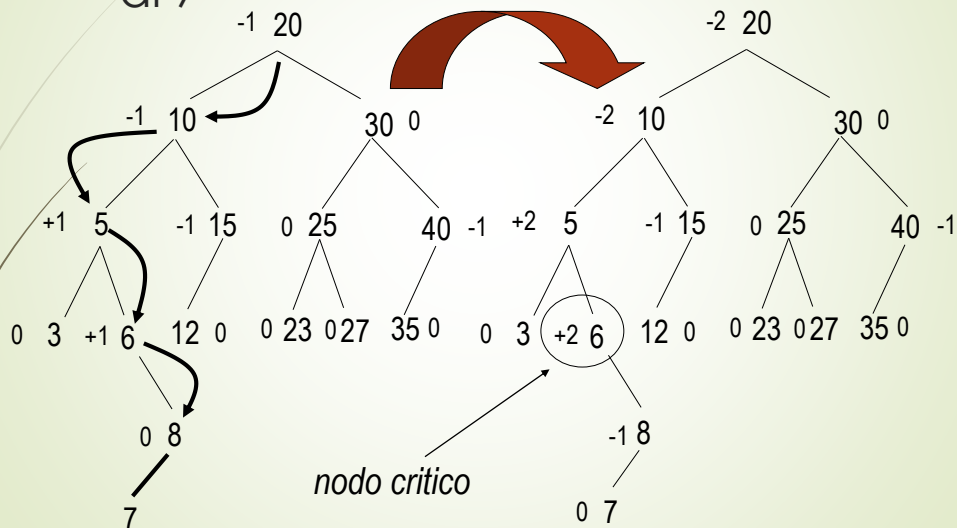
Esempio di albero AVL



Ribilanciamento di alberi AVL

- Un inserimento di una foglia può provocare uno sbilanciamento dell'albero
 - Per almeno uno dei nodi l'indicatore non rispetta più uno dei tre stati precedenti
- In tal caso bisogna ribilanciare l'albero con operazioni di rotazione (semplice o doppia) agendo sul nodo x a profondità massima che presenta un non bilanciamento
 - Tale nodo viene detto nodo critico e si trova sul percorso che va dalla radice al nodo foglia inserito
- Considerazioni simili si possono fare anche per la rimozione di un nodo ...

Esempio di sbilanciamento: inserimento di 7



Ribilanciamento con Rotazioni

Rotazione semplice

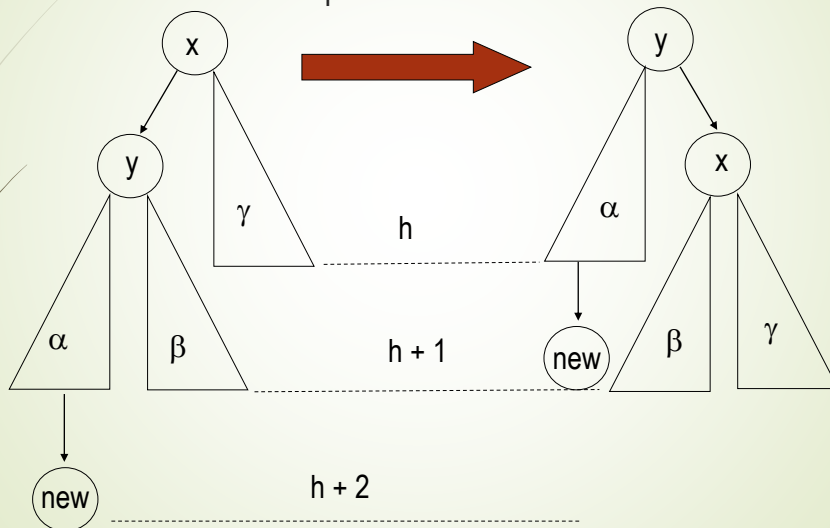
- Inserimento nel sottoalbero sinistro del figlio sinistro del nodo critico
- Inserimento nel sottoalbero destro del figlio destro del nodo critico

Rotazione doppia

- Inserimento nel sottoalbero destro del figlio sinistro del nodo critico
- Inserimento nel sottoalbero sinistro del figlio destro del nodo critico

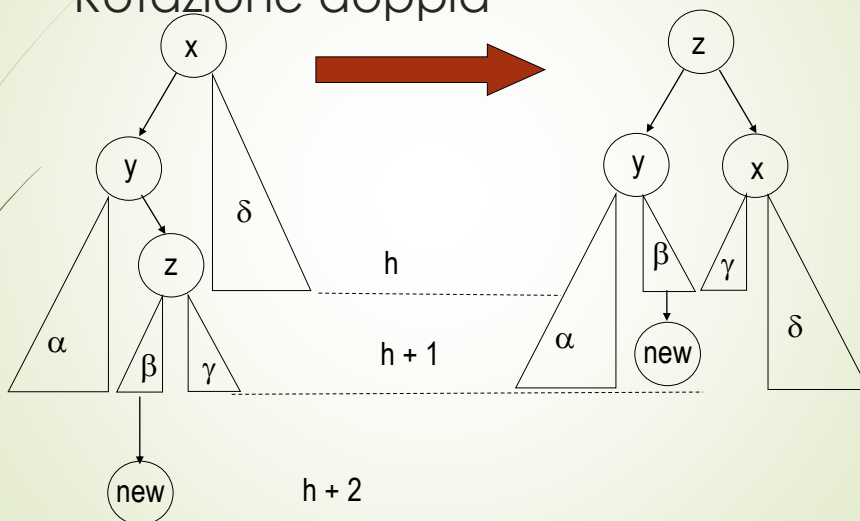
Ribilanciamento di alberi AVL

Rotazione semplice



Ribilanciamento di alberi AVL:

Rotazione doppia



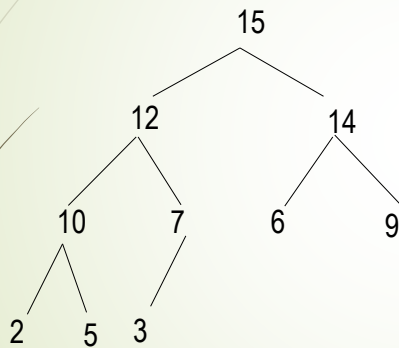
Heap

- Un **heap** è un albero binario bilanciato con le seguenti proprietà
 - Le foglie (nodi a livello h) sono tutte addossate a sinistra
 - ogni nodo v ha la caratteristica che l'informazione ad esso associata è la più grande tra tutte le informazioni presenti nel sottoalbero che ha v come radice
- Usato per realizzare code a priorità
 - Le operazioni sono inserimento di un elemento e rimozione del max

Realizzazione di un heap

- Per le sue caratteristiche un heap può essere realizzato con un array
 - I nodi sono disposti nell'array per livelli
 - La radice occupa la posizione 0
 - Se un nodo occupa la posizione i , il suo figlio sinistro occupa la posizione $2*i+1$ e il suo figlio destro occupa la posizione $2*i+2$

Esempio di Heap



rappresentazione

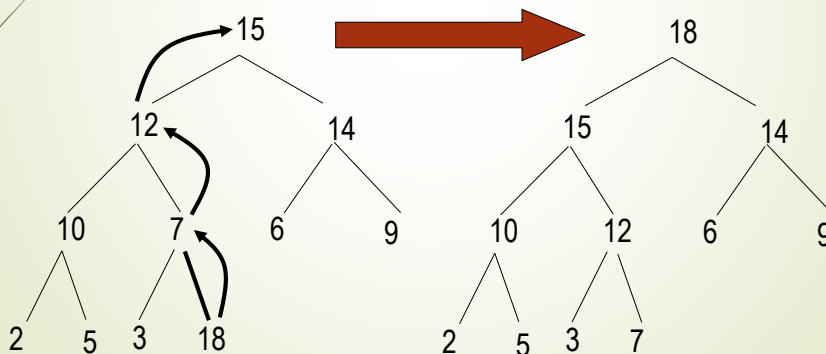
0	1	2	3	4	5	6	7	8	9
15	12	14	10	7	6	9	2	5	3

Esempio:

- Il nodo con etichetta 14 occupa la posizione 2;
- Il suo figlio sinistro (6) occupa la posizione $2*i+1 = 2*2+1 = 5$;
- Il suo figlio destro (9) occupa la posizione $2*i+2 = 2*2+2 = 6$;

Inserimento

- Si inserisce il nuovo nodo come ultima foglia
- Il nodo inserito risale lungo il percorso che porta alla radice per individuare la posizione giusta
- Esempio: inserimento di 18



Rimozione

- Si rimuove sempre la radice (l'informazione maggiore, in quanto un heap viene usato per realizzare code a priorità) e si pone l'ultima foglia al posto della radice
- Si scambia l'informazione contenuta nella radice con la maggiore dei suoi due sottoalberi e si ripete il procedimento fino ad arrivare ad una foglia

