Corso di Laurea in INFORMATICA

a.a. 2011-2012

Algoritmi e Strutture Dati

MODULO 8

STRUTTURE NON LINEARI

Alberi binari di ricerca

Ricerca

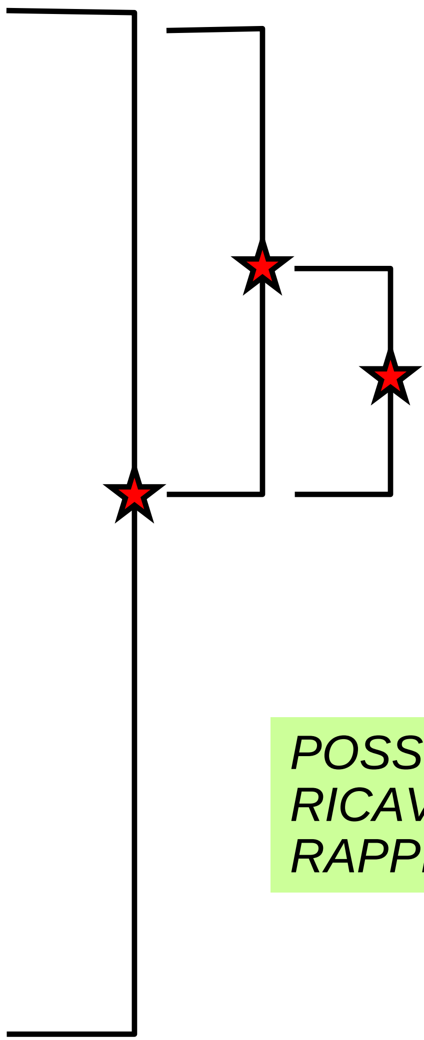
- Molte applicazioni richiedono un insieme dinamico che fornisca solo operazioni di inserimento/cancellazione e ricerca o al più la ricerca di elementi particolari della collezione come il massimo/minimo o il predecessore/successore

PROBLEMA: RICERCA BINARIA DI UN NOME IN UNA TABELLA ORDINATA

ALDO
ALCESTE
ALFIO
ARMANDO
BALDOVINO
CARLO
CLORINDA
DARIO
ERMINIA
EUSTACHIO
GOFFREDO
GUGLIELMO
MARCO
MARIO
RAIMONDO
ROBERTO
TANCREDI
UGO

RICERCA DELLA CHIAVE CORRISPONDENTE A "CLORINDA"

ALDO
ALCESTE
ALFIO
ARMANDO
BALDOVINO
CARLO
CLORINDA
DARIO
ERMINIA
EUSTACHIO
GOFFREDO
GUGLIELMO
MARCO
MARIO
RAIMONDO
ROBERTO
TANCREDI

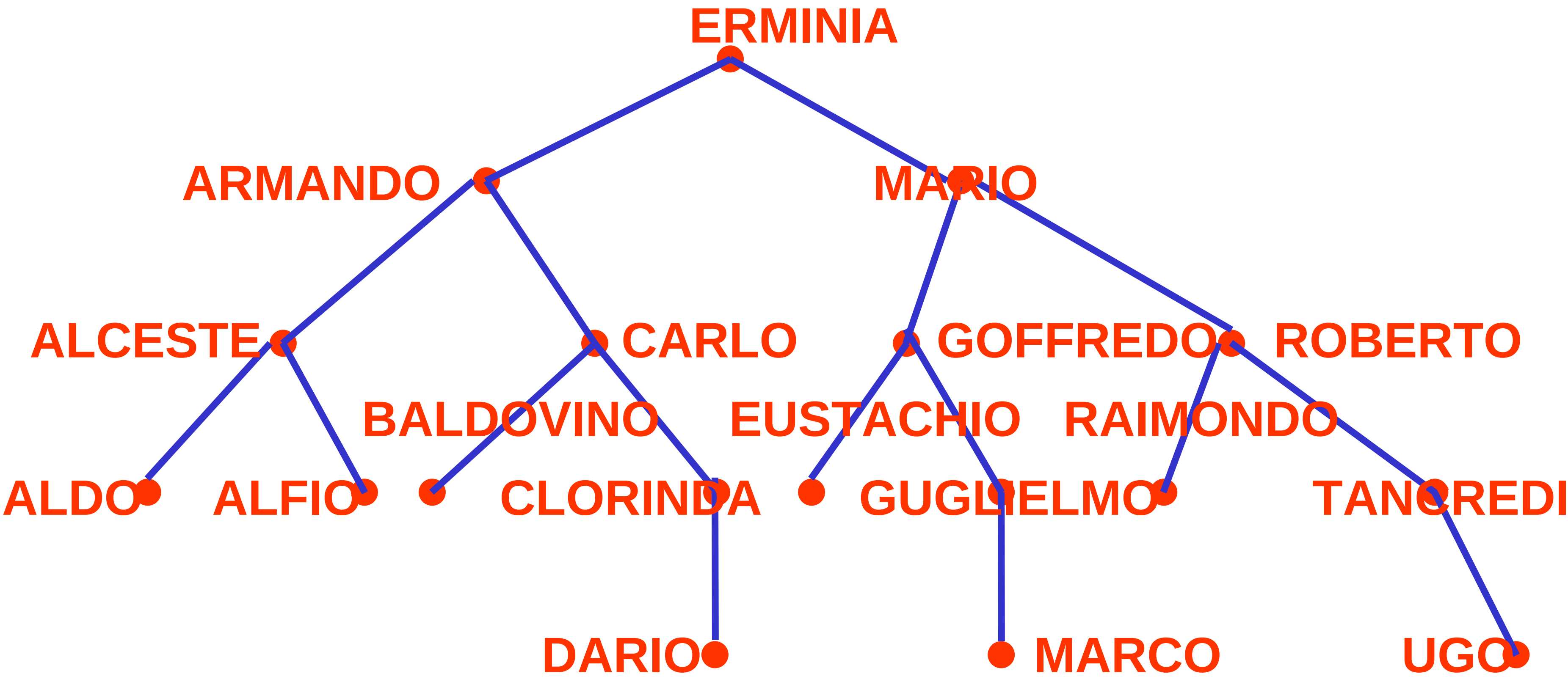


POSSIAMO CONSIDERARE LE CHIAVI RICAVATE DAI NOMI TRASFORMANDO LA RAPPRESENTAZIONE BINARIA IN NUMERO

RICERCA BINARIA (IN UNA TABELLA)

```
RICERCA_BINARIA(A:tabella;
                K: chiave; SUCCESSO: boolean)

  MAX ← N
  MIN ← 1
  SUCCESSO ← false
  while (MAX ≥ MIN) do
    MED ← (MAX - MIN)/2
    if (A[MED].ATTR_CHIAVE=K) then
      SUCCESSO ← true
    else
      if (A[MED].ATTR_CHIAVE>K) then
        MAX ← MED-1
      else
        MIN ← MED+1
```



IL PROCEDIMENTO DI RICERCA BINARIA DI UN NOME IN UNA TABELLA PUO' ESSERE VISUALIZZATO MEDIANTE UN ALBERO BINARIO.

albero binario di ricerca

E' un albero binario nel quale ogni nodo v contiene un elemento elem(v) cui è associata una chiave key(v) presa da un dominio totalmente ordinato.

Inoltre soddisfa le seguenti proprietà

per ogni nodo

1. **tutte** le chiavi nel sottoalbero sinistro di v sono $\leq$ key (v)
2. tutti le chiavi nel sottoalbero destro di v sono $\geq$ key(v)

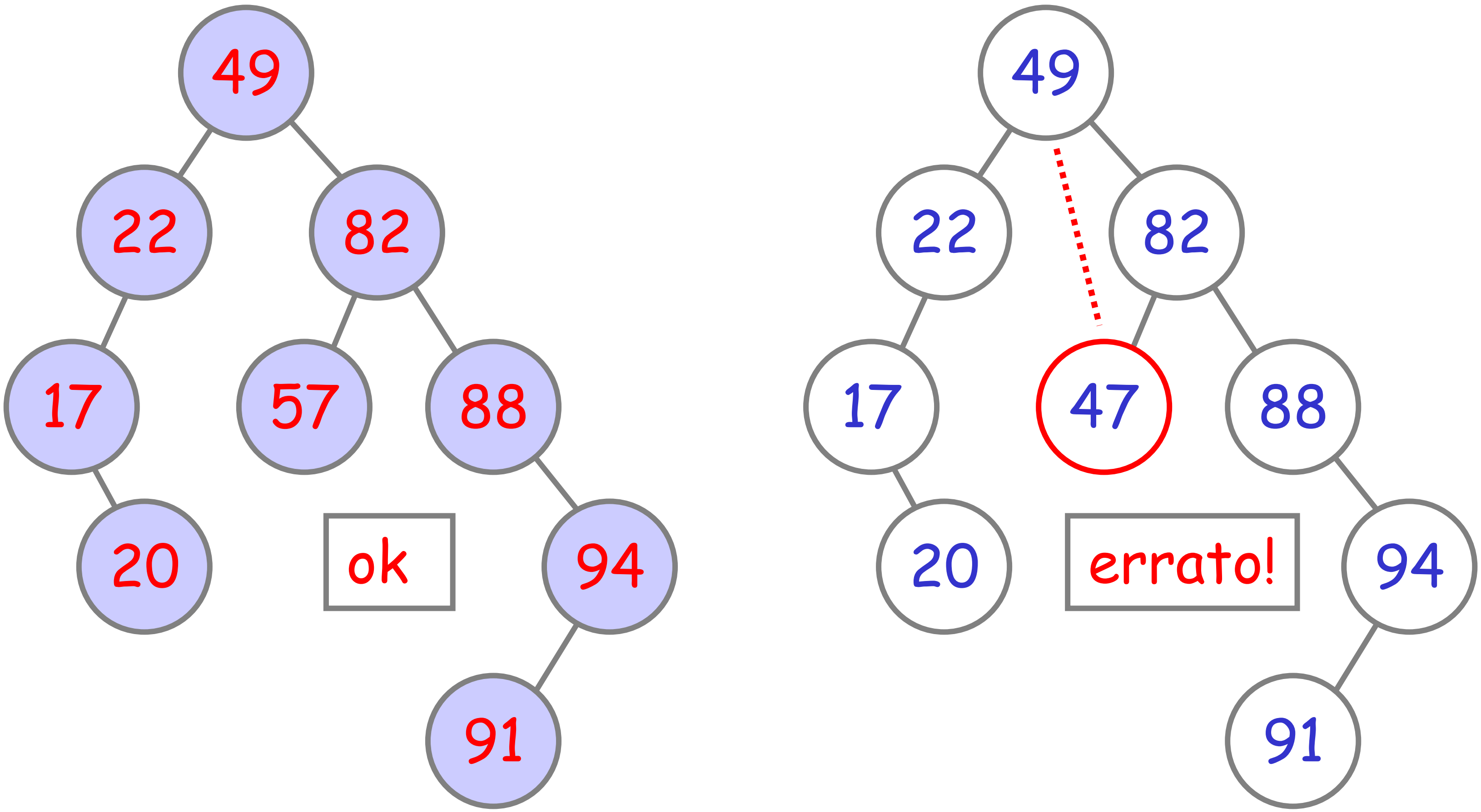
albero binario di ricerca

E’ indicato spesso come BST (binary search tree)

Le proprietà 1 e 2, note come *proprietà di ricerca*, hanno lo scopo di agevolare la ricerca di un elemento e garantiscono che, visitando l’albero binario di ricerca in ordine simmetrico, si ottengono chiavi in ordine non decrescente.

Grazie a tali proprietà per ricercare un elemento è sufficiente confrontare l’elemento da ricercare con quello contenuto nella radice e, nel caso siano diversi, riapplicare il procedimento al sottoalbero sinistro o destro.

albero binario di ricerca



D’ora in poi, per semplicità, useremo come chiave l’elemento stesso

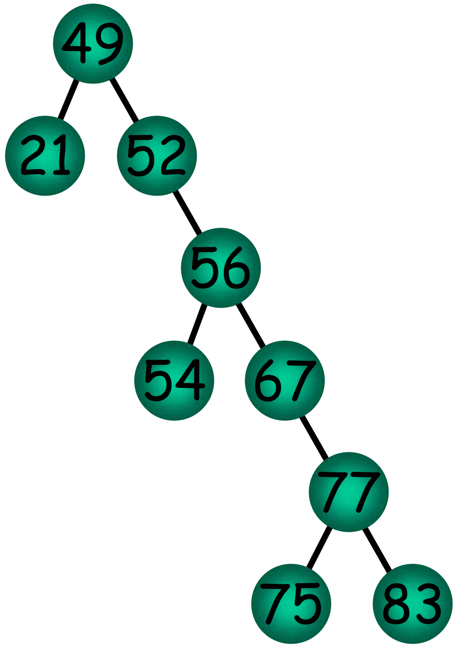
Operatori utili in un BST

Oltre ai normali operatori su albero binario è utile la operazione di verifica dell’appartenenza definita per l’insieme A

APPARTIENE (x,A)=b

che si basa su una discesa lungo un percorso radice-foglia.

Ovviamente sarebbe preferibile avere alberi *popolosi* e *simmetrici*, ma non sempre è possibile, specie dopo ripetuti aggiornamenti.

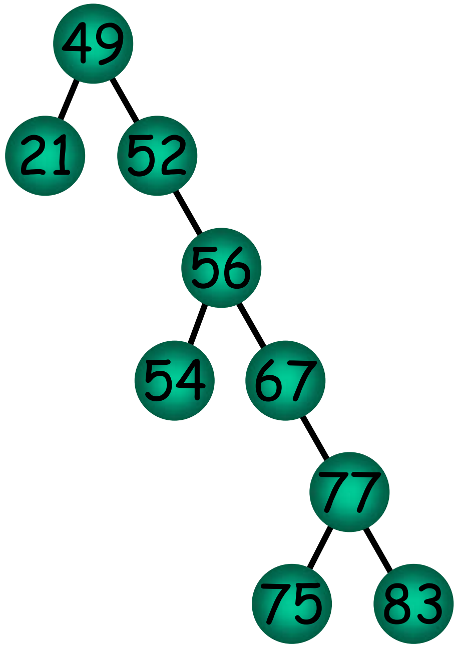


istanze problematiche: alberi molto profondi e sbilanciati...

costo della ricerca in un BST

BST di *n* nodi

- caso peggiore: O(*n*)
- caso medio: dipende dalla distribuzione
- caso migliore: O(1) (poco interessante)



istanze problematiche: alberi molto profondi e sbilanciati...

L’operatore MIN (o MAX)

Per migliorare la gestione di un albero ordinato è utile disporre dell’operatore MIN (0 MAX) che ha la funzione di estrarre l’elemento minimo (o massimo).

Basta effettuare una ricerca in cui si segue sempre il figlio sinistro (destro) fino ad arrivare ad un nodo che non ha figlio sinistro (destro)

Se h è il livello massimo delle foglie l’operatore MIN è di ordine O(h)

Determinazione di MAX e MIN

- La chiave minima dovrà essere nel sottoalbero sinistro della radice
- Pertanto per determinare l'elemento minimo è sufficiente discendere tutti i nodi da figlio sinistro in figlio sinistro fino ad arrivare alla foglia …e così via
- Analogamente la chiave massima in un albero binario dovrà trovarsi nel sottoalbero destro della radice
- Si procede come sopra discendendo per i figli destri fino ad arrivare alla foglia….

Minimo e massimo

```
Tree-Minimum(x)
While not sinistrovuoto[x,T]
do      x ← figliosinistro[x,T]
return x

Tree-Maximum(x)
while not destrovuoto[x,T]
do      x ← figliodestro[x,T]
return x
```

Inserire e cancellare elementi

Per inserire e cancellare elementi in un albero **BST** occorre effettuare una ricerca in modo simile a quanto visto per APPARTIENE per localizzare il nodo contenente l'elemento da eliminare (*in caso di cancellazione*) o il sottoalbero vuoto al posto del quale va sostituita una foglia contenente il nuovo elemento (*in caso di inserimento*).

inserimento in un BST

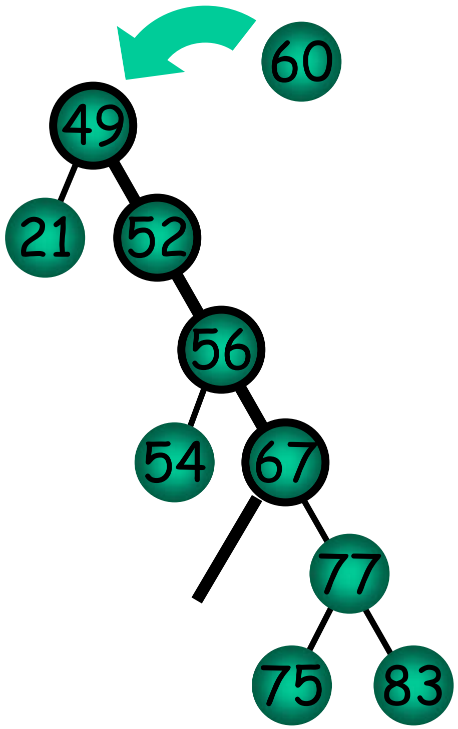
il nuovo nodo *u* viene sempre inserito come foglia

- fase 1: cerca il nodo genitore *v*
- fase 2: inserisci *u* come figlio di *v*

inserimento in un BST/2

- fase 1: termina quando si raggiunge un nodo privo del figlio in cui avrebbe avuto senso continuare la ricerca

- fase 2: si limita a collegare una nuova foglia



inserimento in un BST/3

caso peggiore

- costo fase 1: O(*n*)
- costo fase 2: O(1)
- costo totale: O(*n*)

caso medio (distrib. unif.)

- costo fase 1: O(lg *n*)
- costo fase 2: O(1)
- costo totale: O(lg *n*)

L'altezza attesa di un BST ottenuto tramite n inserimenti random è logaritmica

cancellazione da un BST

di solito un po' più difficile dell'inserimento perché l'elemento *x* da eliminare non necessariamente è un nodo foglia u.

tre casi:

- cancellazione di una foglia
- cancellazione di un nodo con un solo figlio
- cancellazione di un nodo con due figli

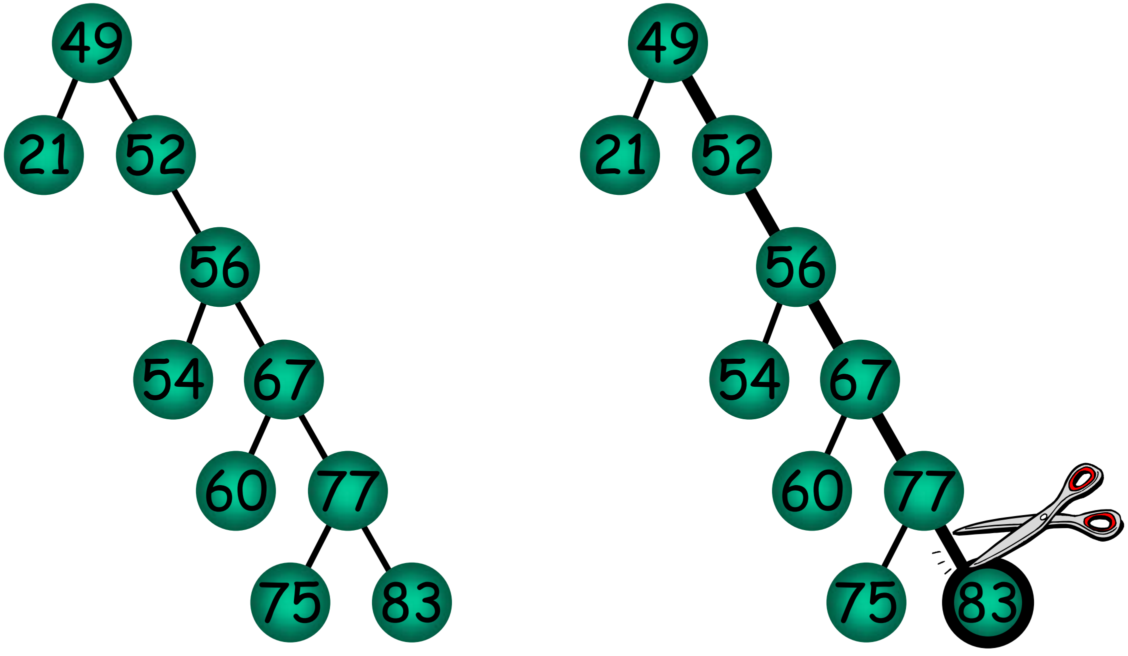
cancellazione: foglia

- basta individuare il nodo padre e distaccare la foglia

- individuare il genitore significa sostanzialmente effettuare una ricerca (come nella fase 1 dell'inserimento)

esempio 1

cancellazione di 83

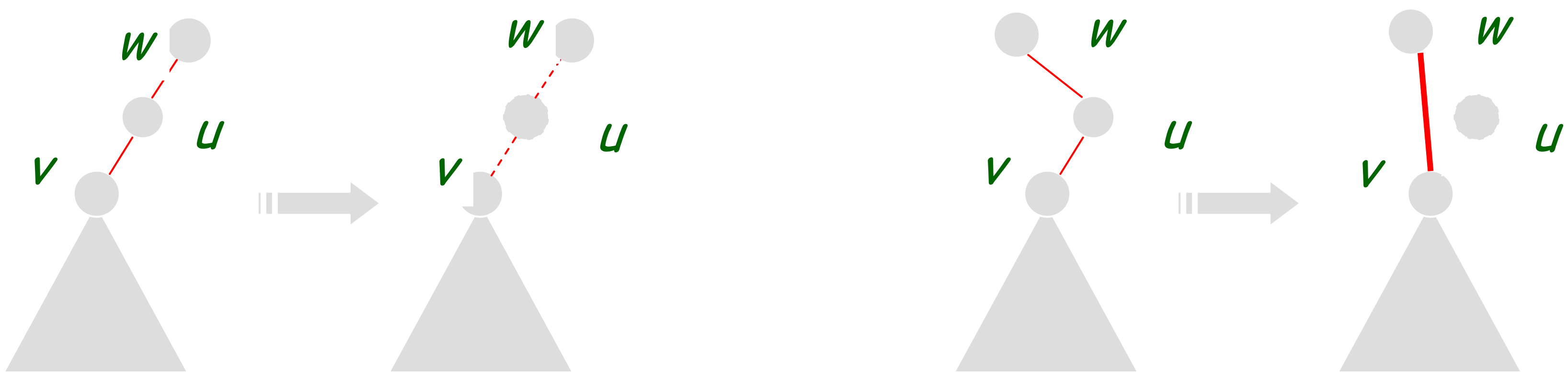


cancellazione: nodo con un solo figlio

u = nodo da cancellare

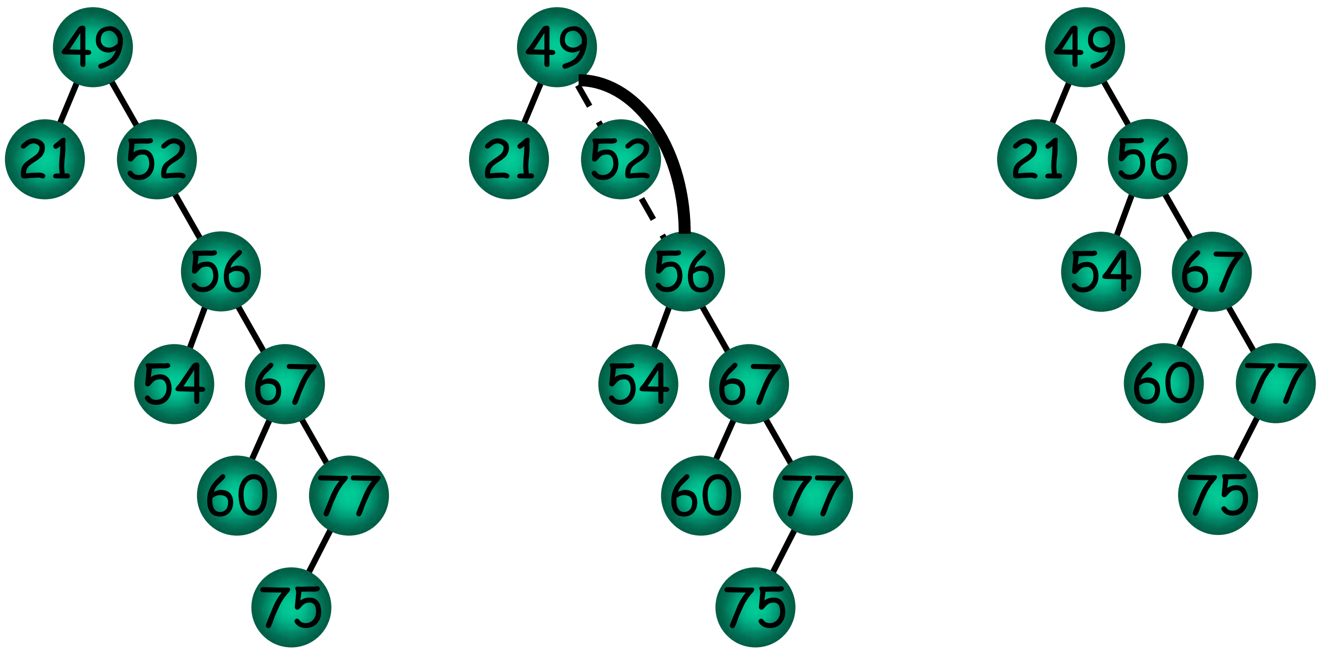
v = unico figlio di u

- individuare *w* = genitore di *u*
  - se *u* è radice, *v* diviene la nuova radice
- se esiste *w*, sostituire al collegamento (*w*,*u*) il collegamento (*w*,*v*)



esempio 2

cancellazione di 52



cancellazione: nodo con 2 figli

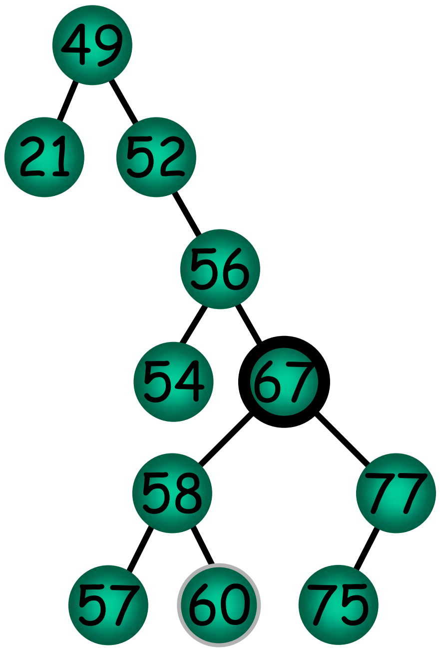
Se **u** ha due figli ci si può ricondurre al primo o al secondo caso sostituendo l'elemento da cancellare **x** con il più piccolo elemento **y** tale che **y >x** .

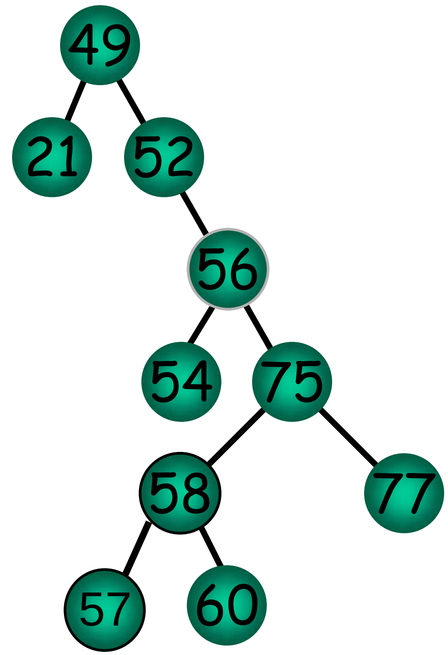
Disponendo di **MIN** l'operazione è semplice: si scende sempre a sinistra nel sottoalbero radicato nel figlio destro di **u** fino a trovare il nodo **v** da eliminare nella struttura.

L'elemento **y** prende il posto di **x** nel nodo **u** mentre viene rimosso il nodo **v**

esempio 3

Cancellazione di 67





introduzione al bilanciamento

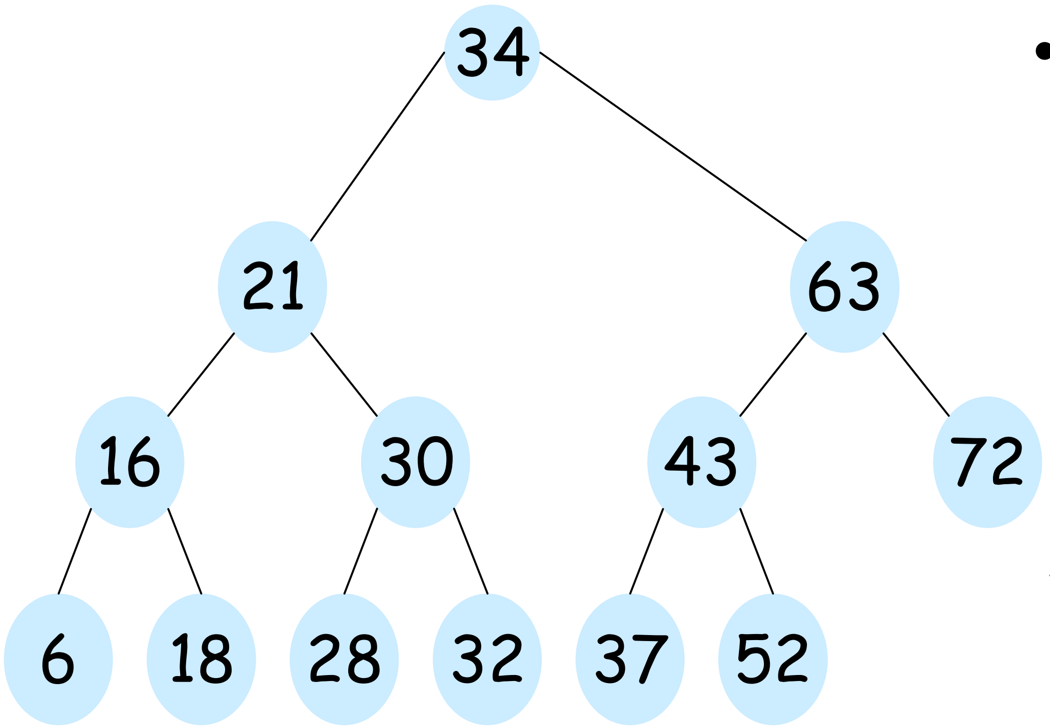
nozione intuitiva di bilanciamento (*applicabile sia ad alberi binari che n-ari*)

- **tutti i rami di un albero hanno approssimativamente la stessa lunghezza**
- **ciascun nodo interno ha "molti" figli**

caso ideale per un albero n-ario

- **ciascun nodo ha 0 o *n* figli**
- **la lunghezza di due rami qualsiasi differisce di al più una unità**

ATTENZIONE! ripetuti inserimenti/eliminazioni possono distruggere il bilanciamento
–degrado delle prestazioni



- un albero binario perfettamente bilanciato di n nodi ha altezza

$$\lfloor \lg_2 n \rfloor + 1$$

se ogni nodo ha 0 o 2 figli

$n_f = n_i + 1$

n_f = # foglie

n_i = # nodi interni

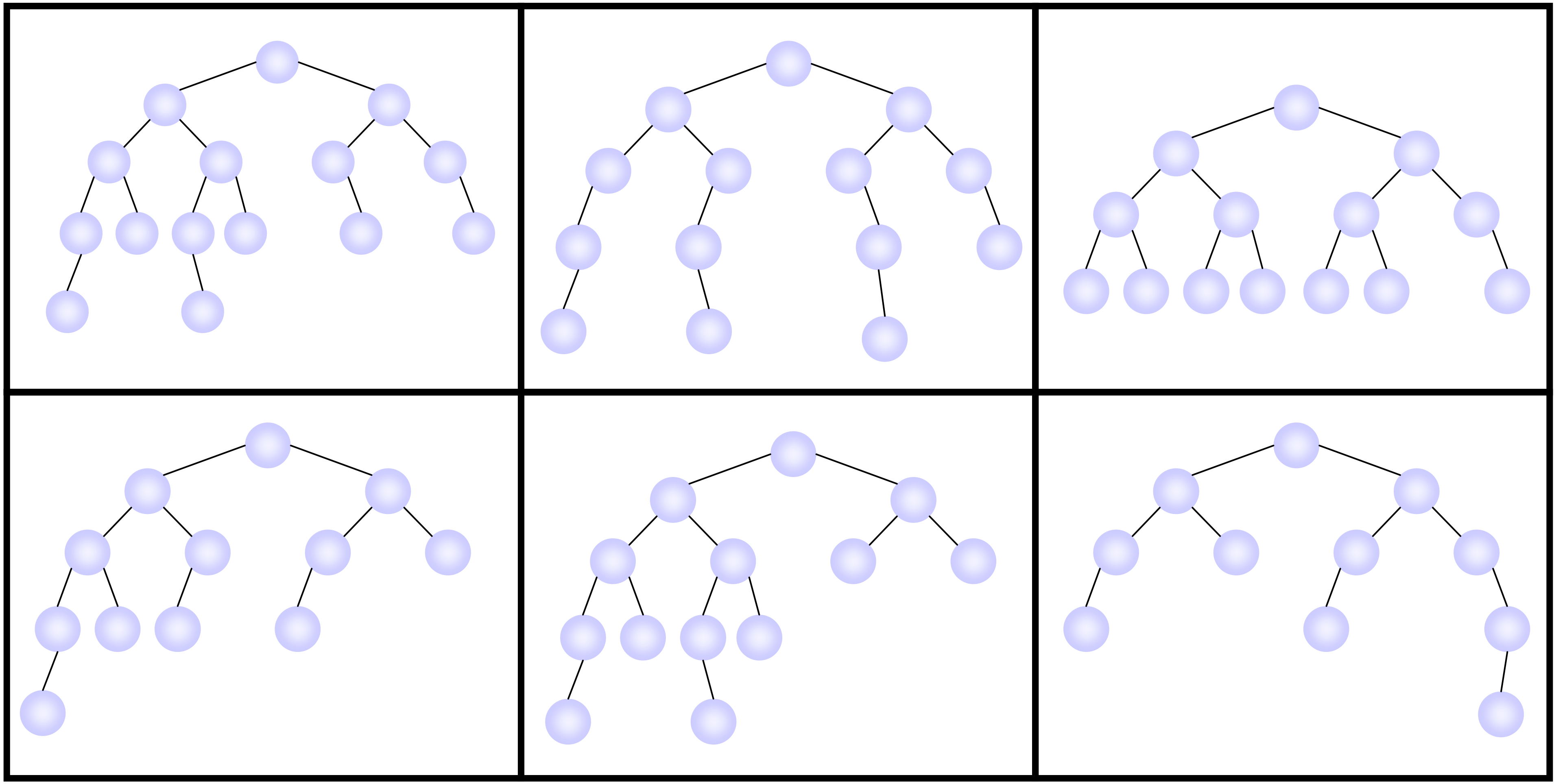
$n = n_f + n_i$

le foglie sono circa il 50% dei nodi

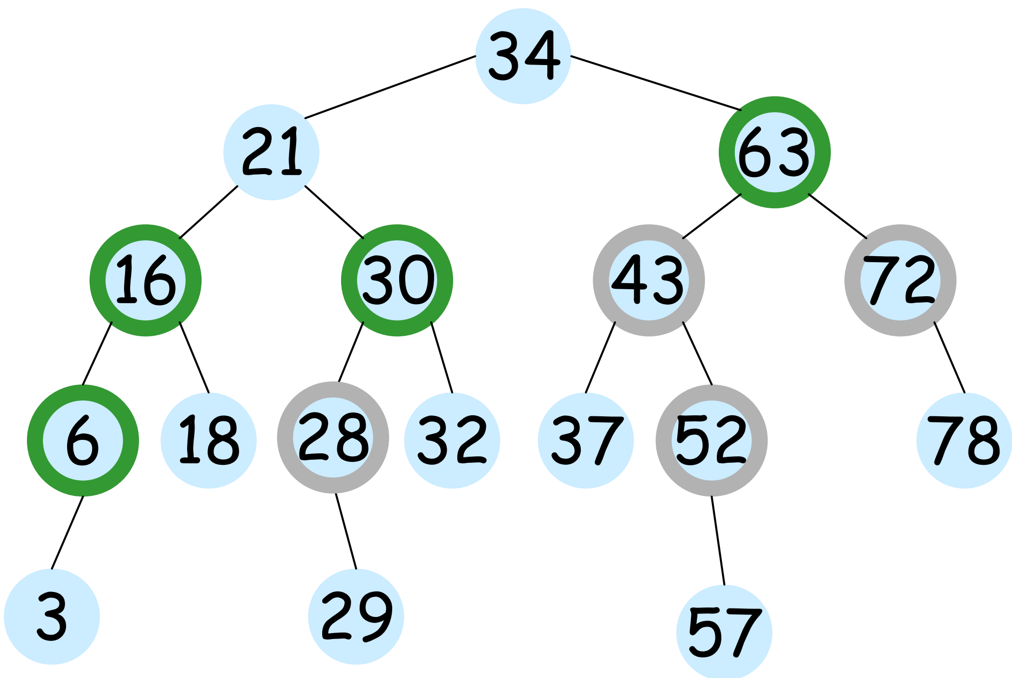
bilanciamento in altezza

- un albero è *bilanciato in altezza* se le altezze dei sottoalberi sinistro e destro <u>di ogni nodo</u> differiscono di al più un'unità
- gli alberi bilanciati in altezza sono detti *alberi AVL* da **A**del'son-**V**el'skii & **L**andis, primi proponenti

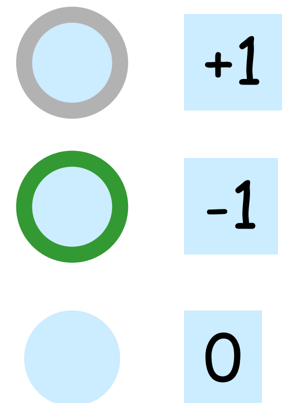
alberi AVL?



fattore di bilanciamento

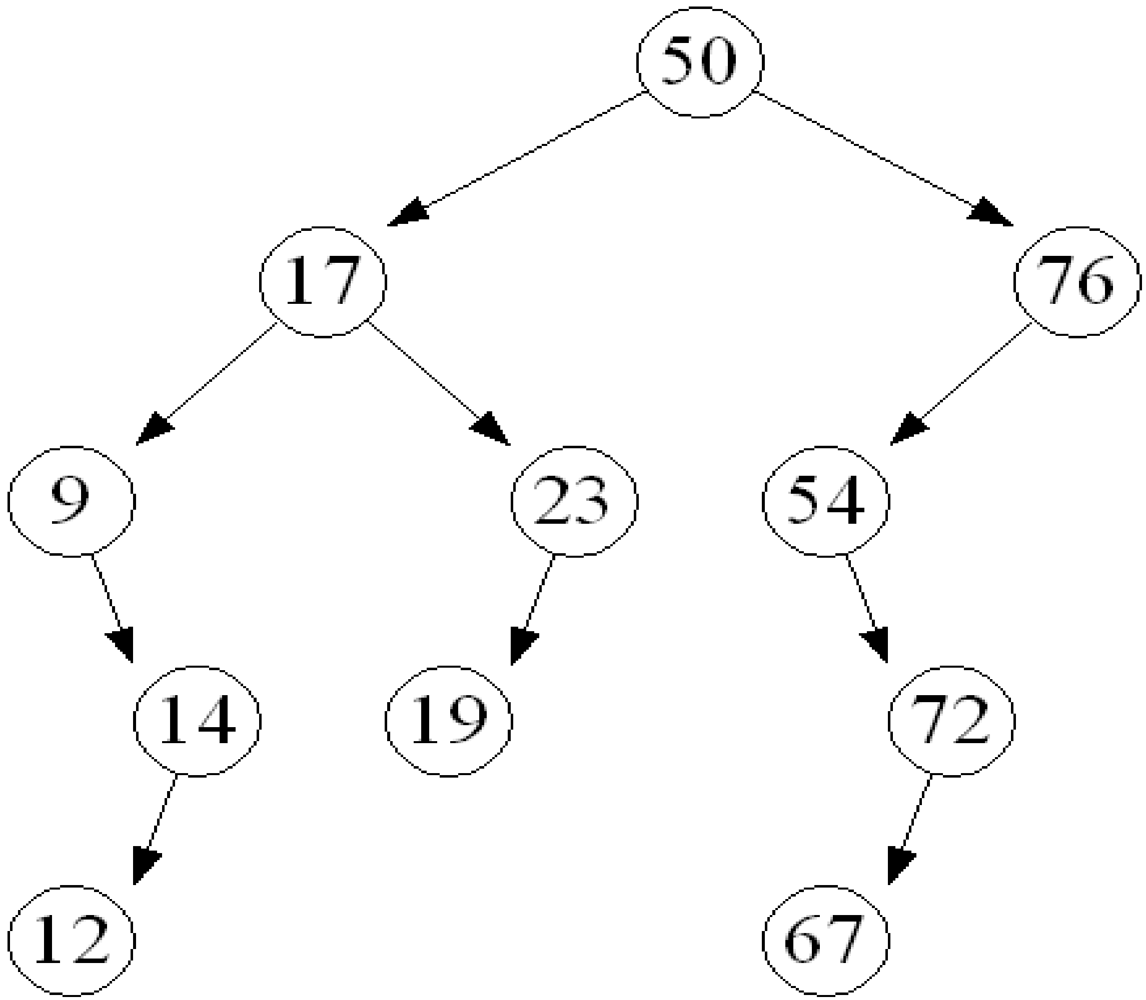


fattore di bilanciamento (FDB):
altezza sottoalbero dx – altezza sottoalbero sx

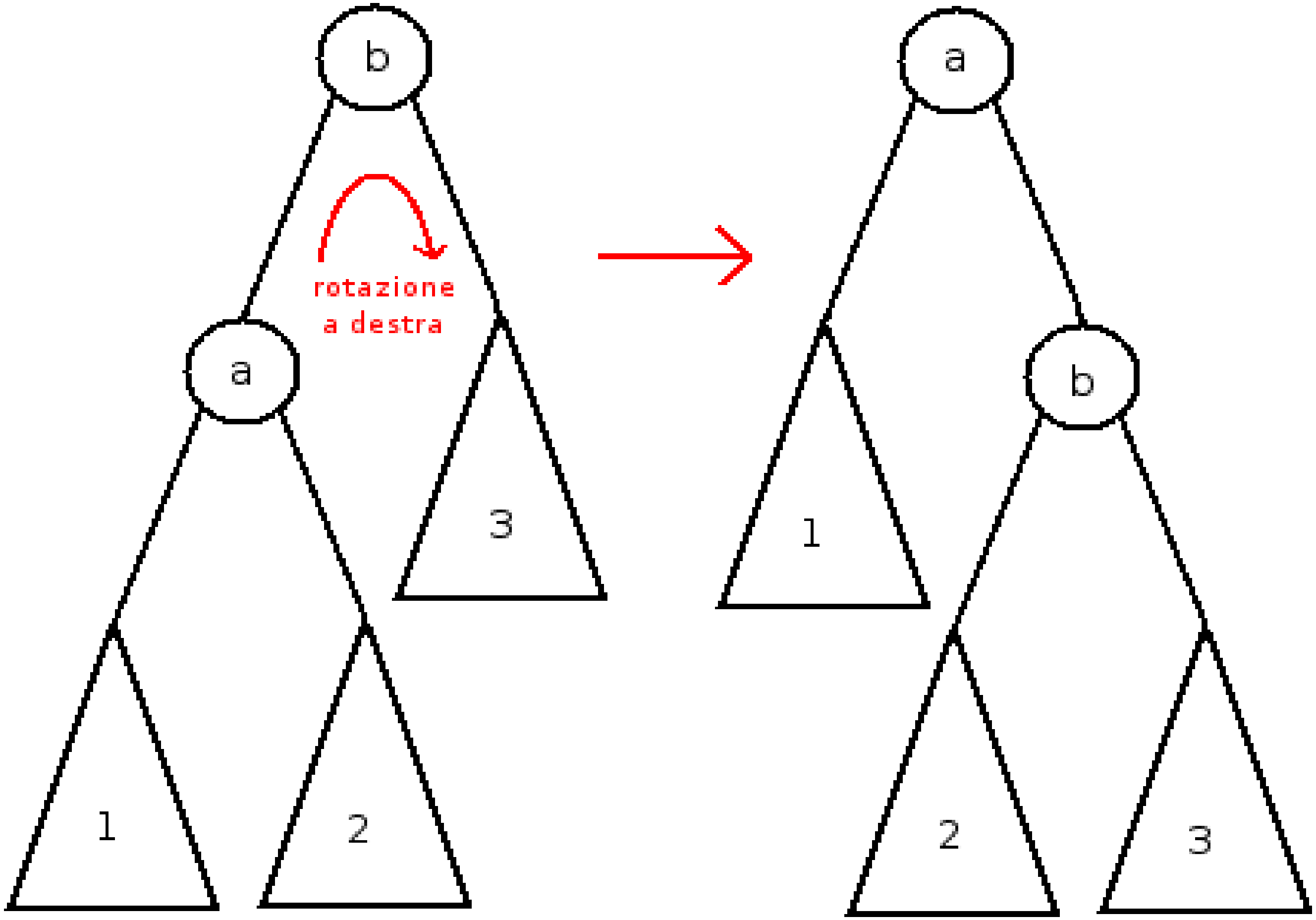


in un albero bilanciato in altezza
$|FDB| \leq 1$, per ogni nodo

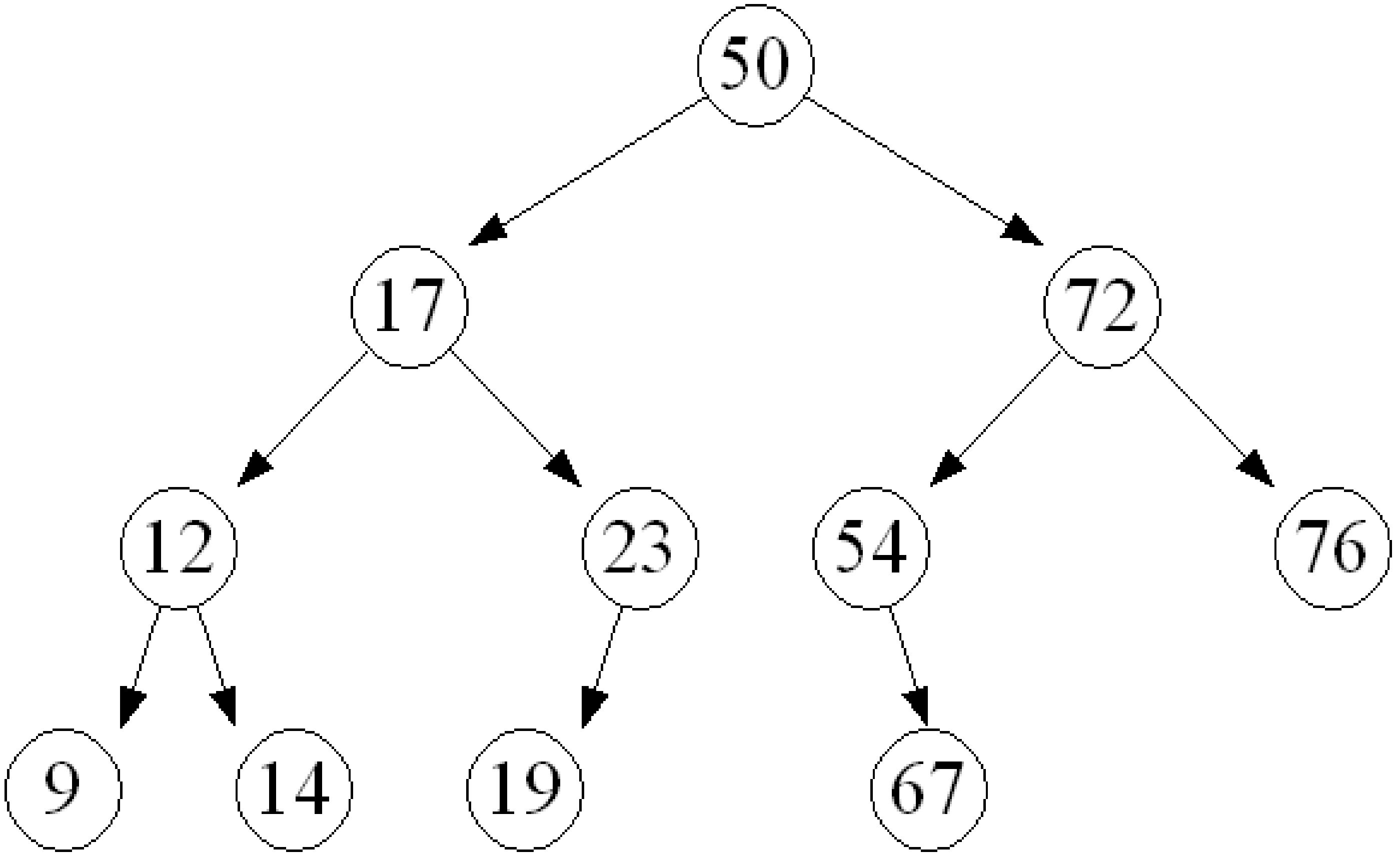
Se un albero è sbilanciato si può intervenire con degli algoritmi di bilanciamento



Esempio di un albero **non-AVL**: il nodo contenente il numero 9 ha un coefficiente di bilanciamento +2, quello contenente il numero 76 di -3 e quello contenente il numero 54 di +2



Un nodo con il coefficiente di bilanciamento diverso da 1, 0 o -1 è considerato sbilanciato e viene ribilanciato grazie alle rotazioni a destra o a sinistra. Questo è un esempio di rotazione a destra; la rotazione a sinistra è simmetrica



Lo stesso albero, adesso **AVL** (dopo essere stato bilanciato: rotazione a sinistra sul nodo contenente il numero 9 e una rotazione a destra sul nodo contenente il numero 54, seguita da una rotazione a destra sul nodo contenente il numero 72)

Rotazione a sinistra

Operazioni

1. far diventare B figlio destro di x
2. far diventare x figlio sinistro di y
3. far diventare y figlio di p, il vecchio padre di x