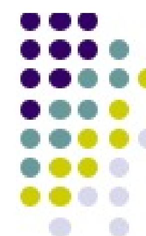
Alberi di ricerca binari

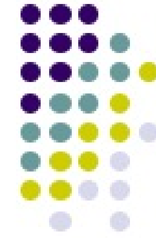


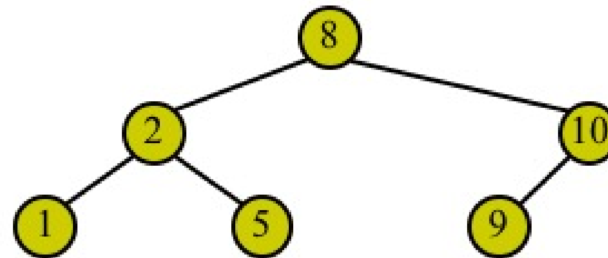
- Un albero di ricerca binario è un albero binario che memorizza in ciascun nodo una chiave in modo tale che
  - Se u, v e w sono tre nodi tali che u si trova nel sottoalbero sinistro di v e w si trova nel sottoalbero destro di v, allora

    u.dato.chiave < v.dato.chiave < w.dato.chiave

  (sulle chiavi è definita una relazione di ordine totale)

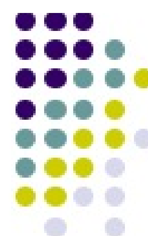
Esempio





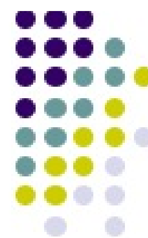
- Per semplicita` vengono mostrate solo le chiavi degli elementi

Visita inorder di un albero binario di ricerca



- Una visita inorder di un albero di ricerca binario visita le chiavi in ordine crescente
  - Possiamo ottenere la sequenza ordinata della chiavi

Algoritmo di ricerca

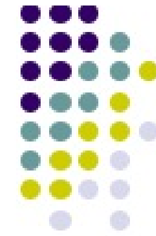


- Input: la chiave da cercare k e un nodo u dell’albero
- Output: l’elemento dell’albero con chiave k se k e\` presente nel dizionario; null altrimenti.

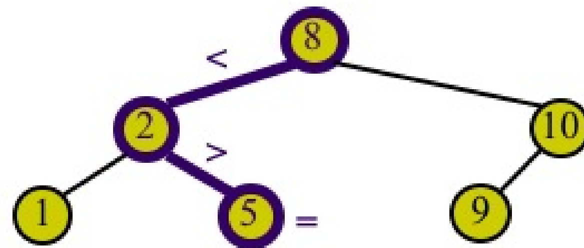
Comportamento dell’algoritmo di ricerca

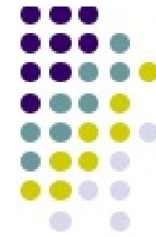
- Se la chiave k e\` uguale a quella dell’elemento di u l’algoritmo restiuisce l’elemento presente in u
- Se la chiave k e\` minore di quella dell’elemento di u la ricerca prosegue nel sottoalbero sinistroa di u
- Se la chiave k e\` maggiore di quella dell’elemento di u la ricerca prosegue nel sottoalbero destro di u
- Se u e\` null l’algoritmo resituisce null

Esempio



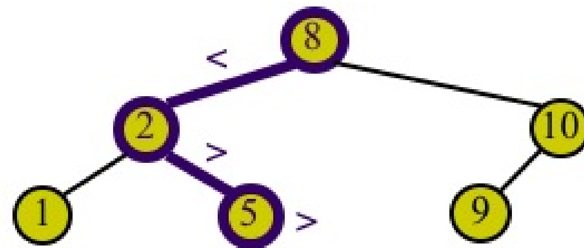
- Ricerca(radice,5)





Esempio

- Ricerca(root,6)



Ricerca con una chiave k

- Assumiamo l'esistenza di un descrittore albero con i campi:
  - `albero.radice` (=`null` per l'albero vuoto)
  - `albero.dimensione` (=0 per l'albero vuoto)

La funzione Ricerca prende in input solo la chiave da cercare e invoca una funzione Ricerca che, oltre a prendere in input la chiave, prende in input un nodo dell'albero.

```
1 Ricerca( k ):
2   RETURN RicercaInAlbero(albero.radice,k);
```

```
1 RicercaInAlbero( u, k ):
2   IF (u == null) RETURN null;
3   IF (k == u.dato.chiave) {
4     RETURN u.dato;
5   } ELSE IF (k < u.dato.chiave) {
6     RETURN RicercaInAlbero( u.sx, k );
7   } ELSE {
8     RETURN RicercaInAlbero( u.dx, k );
9   }
```

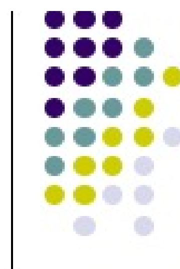
$O(h)$ tempo dove $h =$ altezza dell'albero

Algoritmo di Inserimento

- Input: un elemento e e un nodo u
- Se la chiave di e non e` presente nell'albero inserisce e nell'albero

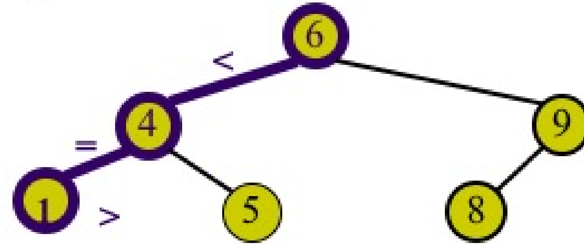
Comportamento dell'algoritmo di inserimento

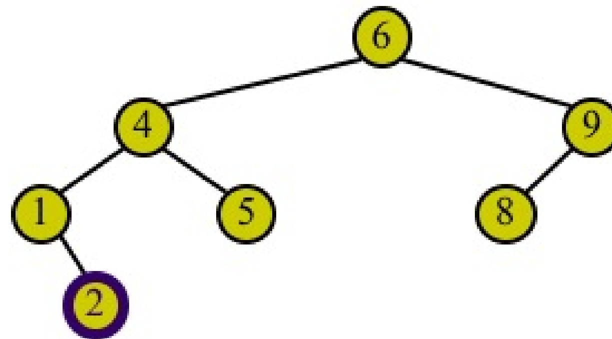
- Se la chiave k e` uguale a quella dell'elemento di u, la chiamata termina senza fare niente
- Se la chiave k e` minore di quella dell'elemento di u l'inserimento viene effettuato ricorsivamente nel sottoalbero sinistro di u
- Se la chiave k e` maggiore di quella dell'elemento di u l'inserimento viene effettuato ricorsivamente nel sottoalbero destro di u
- Se u e` null l'algoritmo crea un ndo foglia contenente e che diventa figlio del nodo su cui e` stato invocato precedentemente l'algoritmo .



Esempio

- Inseriamo 2





Inserimento di un elemento e

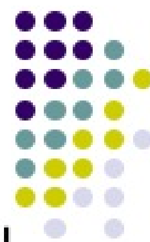
L'inserimento è simile alla ricerca ma quando l'algoritmo è invocato su null allora viene creato un nuovo nodo in cui viene inserito l'elemento *e*.

```
1  Inserisci( e ):
2    RETURN InserisciInAlbero(albero.radice,e);
```

```
1  InserisciInAlbero( u, e ):
2    IF (u == null) {
3      u = NuovoNodo();
4      u.dato = e;
5      u.sx = u.dx = null;
6    } ELSE IF (e.chiave < u.dato.chiave) {
7      u.sx = InserisciInAlbero( u.sx, e );
8    } ELSE IF (e.chiave > u.dato.chiave) {
9      u.dx = InserisciInAlbero( u.dx, e );
10   }
11   RETURN u;
```

$O(h)$ tempo dove $h =$ altezza dell'albero

Algoritmo di cancellazione

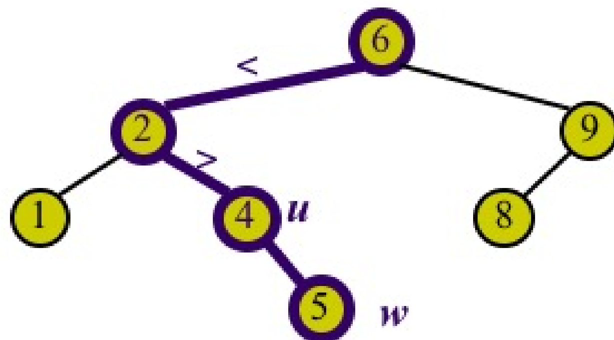


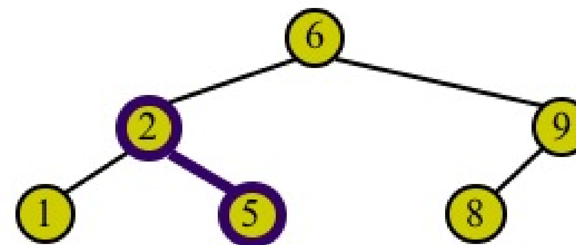
- Input: la chiave k da rimuovere e il nodo u radice del sottoalbero in cui si vuole rimuovere k

- L'algoritmo Cancella(u,k) funziona come segue::
  - Il nodo che contiene k ha al piu`un figlio
  - Il nodo che contiene k ha due figli

Algoritmo di cancellazione

- Caso 1: il nodo u da cancellare ha al piu` un figlio w
  - L'algoritmo di cancellazione rimuove il nodo u e lo rimpiazza con w
  - Esempio: rimoviamo la chiave 4

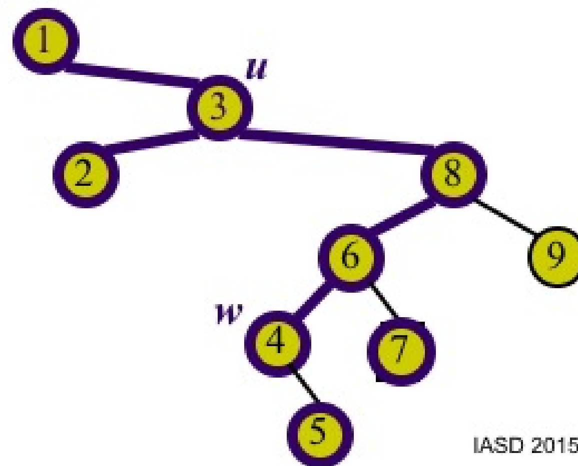


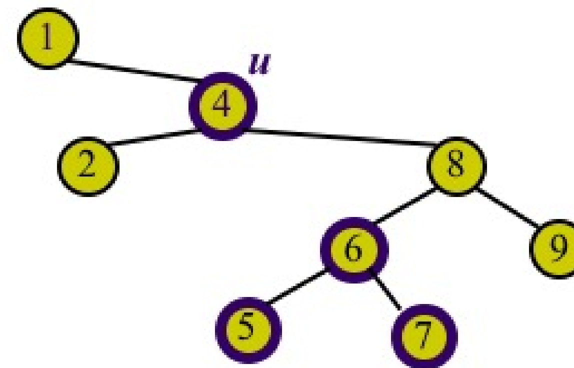


Algoritmo di cancellazione

- Caso 2: la chiave da cancellare è contenuta in un nodo u che ha due figli
  - Troviamo il nodo che contiene il successore dell'elemento da cancellare (è il nodo interno più a sinistra nel sottoalbero destro di w e quindi non ha figlio sinistro)
  - Copiamo l'elemento contenuto in w nel nodo u
  - Rimuoviamo w come nel caso 1
- Esempio: removiamo la chiave 3

Il tempo di esecuzione è O(altezza)





Cancellazione dell'elemento con chiave k in $O(h)$ tempo

Attenzione: il libro assume che i nodi non abbiano il campo padre
Caso 1 (linee 4-7): il nodo `u` è una foglia oppure ha un solo figlio
Caso 2 (linee 8-12): il nodo `u` ha due figli

```
Cancella( k ):
  IF (albero.radice!=null && albero.radice.dato.chiave == k) {
    IF (u.sx == null) {
      albero.radice = u.dx;
    } ELSE IF (u.dx == null) {
      albero.radice = u.sx;
    }
  }
  CancellaDaAlbero(albero.radice,k);
```

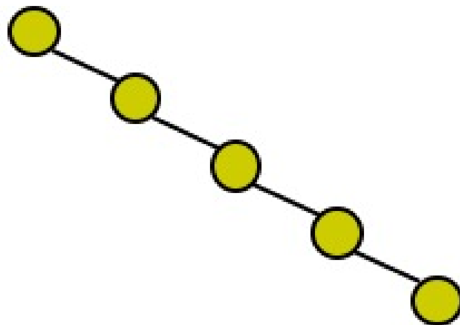
Cancellazione dell'elemento con chiave k in $O(h)$ tempo

```
CancellaDaAlbero( u, k ):
  IF (u != null) {
    IF (u.dato.chiave == k) {
      IF (u.sx == null) {
        u = u.dx;
      } ELSE IF (u.dx == null) {
        u = u.sx;
      } ELSE {
        w=MinimoSottoAlbero(u.dx);
        u.dato=w.dato;
        u.dx=CancellaDaAlbero(u.dx, w.dato.chiave);
      }
    } ELSE IF (k < u.dato.chiave) {
      u.sx = CancellaDaAlbero( u.sx, k );
    } ELSE IF (k > u.dato.chiave) {
      u.dx = CancellaDaAlbero( u.dx, k );
    }
  }
  RETURN u;
```

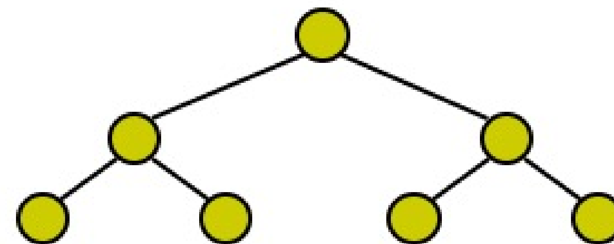
Complessità delle operazioni del dizionario implementato con albero binario di ricerca

- Consideriamo un dizionario con n entrate implementato con un albero binario di ricerca di altezza h
  - Lo spazio usato è $O(n)$
  - I metodi find, insert e remove impiegano tempo $O(h)$
- Nel caso pessimo $h = \Theta(n)$; nel caso ottimo h= $O(\log n)$

Caso pessimo



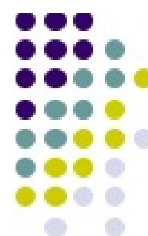
Caso ottimo



Caso pessimo $h = \Theta(n)$

- La forma dell'albero dipende dall'ordine d'inserimento delle chiavi
- Esempio: chiavi inserite in ordine crescente o decrescente
- Se le chiavi sono inserite in ordine casuale, l'albero risultante ha altezza logaritmica in media
- Vediamo come garantire altezza logaritmica al caso pessimo

Alberi AVL

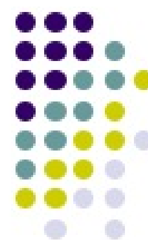


- Alberi AVL :
- Creati negli anni '60
- AVL: acronimo derivato dalle iniziali degli inventori: Adel'son-Velsky e Landis

- Proprieta` albero AVL:

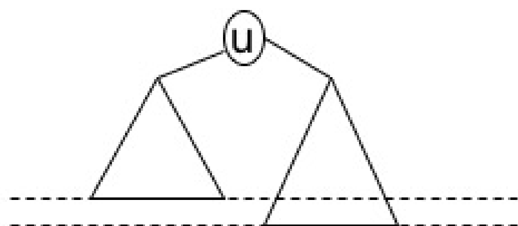
1. Albero binario di ricerca
2. Albero 1-bilanciato

Albero 1-bilanciato



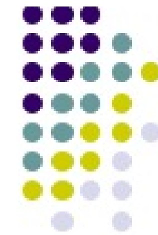
- h(u) = altezza del sottoalbero T(u) radicato in u
- h(null) = -1

- Un nodo u è 1-bilanciato se | h(u.sx) – h(u.dx) |≤ 1



- Un albero e` 1-bilanciato se tutti nodi dell'albero sono 1-bilanciati

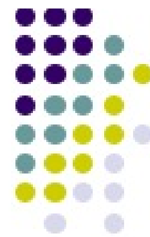
Altezza di un albero AVL



- Si puo` dimostrare che l'altezza di un albero AVL e` O(log n)

- Di conseguenza la ricerca di una richiede tempo O(log n)

Altezza h di un albero AVL: dimostrazione che h=O(log n)



La dimostrazione consiste nel

1. dimostrare che gli alberi AVL di altezza h hanno almeno c^h nodi per una certa costante c>1

- Cio` implica immediatamente h=O(log n)

- Per dimostrare il punto 1, consideriamo gli alberi AVL di altezza h con il piu` piccolo numero di nodi e dimostriamo che per essi vale la 1.

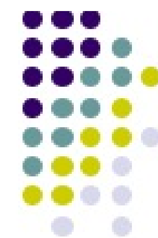
Altezza degli alberi AVL

- Nel seguito chiameremo *minimo* albero 1-bilanciato di altezza h, un albero 1-bilanciato di altezza h avente il più piccolo di nodi tra tutti gli alberi 1-bilanciati di altezza h.
- Prima di dimostrare che il numero di nodi di un albero 1-bilanciato è almeno c^h per una certa costante $c > 1$, dimostriamo la seguente affermazione.
- **Affermazione.** Un minimo albero 1-bilanciato di altezza $h \geq 1$ è formato dalla radice e da due sottoalberi della radice che sono minimi alberi 1-bilanciati di altezza $h-1$ e $h-2$ rispettivamente (per $h = 1$, ovviamente l'albero di altezza $h-2 = -1$ è vuoto).
  Motivazione.
  - La radice di un albero di altezza h deve avere almeno un figlio u di altezza $h-1$ altrimenti l'albero non avrebbe altezza h. L'albero avente come radice u deve essere a sua volta 1-bilanciato altrimenti l'intero albero non sarebbe 1-bilanciato. Ovviamente questo sottoalbero è un minimo albero 1-bilanciato di altezza $h-1$.
  - L'altro figlio della radice deve avere altezza almeno $h-2$ altrimenti la radice non sarebbe 1-bilanciata. Chiamiamo v questo figlio. Siccome l'albero che stiamo considerando contiene il più piccolo numero di nodi tra tutti gli alberi 1-bilanciati di altezza h, il nodo v deve essere radice dell'albero 1-bilanciato più piccolo tra tutti quelli della sua altezza e la sua altezza deve essere la più piccola possibile. Di conseguenza il sottoalbero radicato in v è un albero 1-bilanciato di altezza $h-2$.

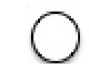
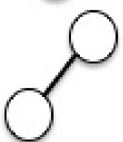
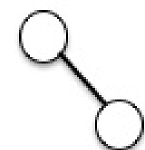
Altezza degli alberi AVL

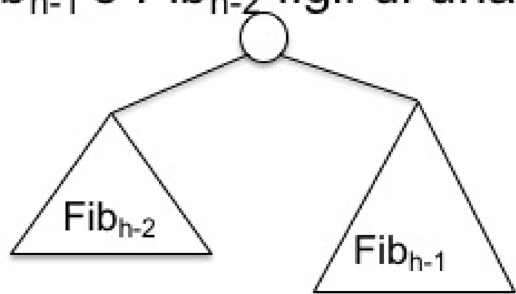
- Siamo pronti a dimostrare che il numero di nodi di un albero 1-bilanciato è almeno c^h per una costante $c > 1$.
- Sia n_h il numero di nodi di un minimo albero 1-bilanciato di altezza h.
- Vogliamo dimostrare per induzione che $n_h \geq (3/2)^h$.
  - Base dell'induzione: $h = 1$.
    Dall'affermazione nella slide precedente, si ha che per $h = 1$ il più piccolo albero 1-bilanciato ha due nodi. Si ha quindi $n_1 \geq 2 \geq 3/2$.
  - Passo induttivo: Supponiamo la disuguaglianza vera per ogni minimo albero 1-bilanciato di altezza a, con $1 \leq a \leq h-1$, e dimostriamo che la disuguaglianza è soddisfatta per il più piccolo albero 1-bilanciato di altezza h.
    Dall'affermazione nella slide precedente deduciamo che $n_h \geq n_{h-1} + n_{h-2} + 1$. Applicando l'ipotesi induttiva a n_{h-1} ed n_{h-2}, otteniamo $n_{h-1} \geq (3/2)^{h-1} + (3/2)^{h-2} + 1 = (3/2)^{h-2}(5/2) + 1 > (3/2)^{h-2}(9/4) = (3/2)^h$.

Alberi di Fibonacci

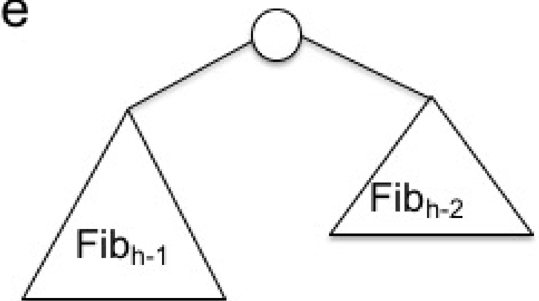


- Abbiamo dimostrato che un albero 1-bilanciato di altezza h e` formato da una radice a cui sono "attaccati" un minimo albero 1-bilanciato di altezza h-1 ed un minimo albero 1-bilanciato di altezza h-2
  - Un albero 1-bilanciato siffatto viene detto albero di Fibonacci
- Definizione di albero di Fibonacci di altezza h:

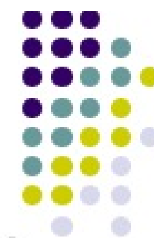
1. Per h=0, Fib_0 e` 
2. Per h=1 Fib_1 e`  oppure 
3. Per h>1, Fib_h e` ottenuto facendo diventare le radici di Fib_{h-1} e Fib_{h-2} figli di una nuova radice



oppure



Inserimento in AVL

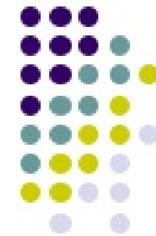


- Dopo la creazione della foglia f, cerca il suo **nodo critico** = minimo antenato u di f che non è più 1-bilanciato
- Ristruttura l’albero effettuando al piu` due rotazioni:
  - 0 o 1 rotazione su uno dei figli di u
  - 1 rotazione su u
- Aggiorna anche i campi altezza degli antenati di f
- Uscendo dalle chiamate ricorsive sugli antenati di f percorre tale cammino a ritroso

Inserimento di un elemento e in un AVL

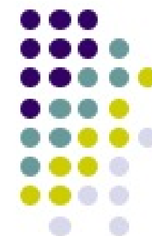
```
Inserisci( u, e ):
  IF (u == null) {
    RETURN f = NuovaFoglia( e );
  } ELSE IF (e.chiave < u.dato.chiave) {
    u.sx = Inserisci( u.sx, e );
    IF (Altezza(u.sx) - Altezza(u.dx) == 2) {
      IF (e.chiave>u.sx.dato.chiave) u.sx=RuotaAntiOraria(u.sx);
      u = RuotaOraria( u );
    }
  } ELSE IF (e.chiave > u.dato.chiave) {
    u.dx = Inserisci( u.dx, e );
    IF (Altezza(u.dx) - Altezza(u.sx) == 2) {
      IF (e.chiave < u.dx.dato.chiave) u.dx = RuotaOraria(u.dx);
      u = RuotaAntiOraria( u );
    }
  }
  u.altezza = max( Altezza(u.sx), Altezza(u.dx) ) + 1;
  RETURN u;
```

La funzione NuovaFoglia



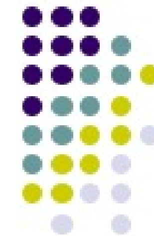
```
NuovaFoglia( e ):
  u = NuovoNodo();
  u.dato = e;
  u.altezza = 0;
  u.sx = u.dx = null;
  RETURN u;
```

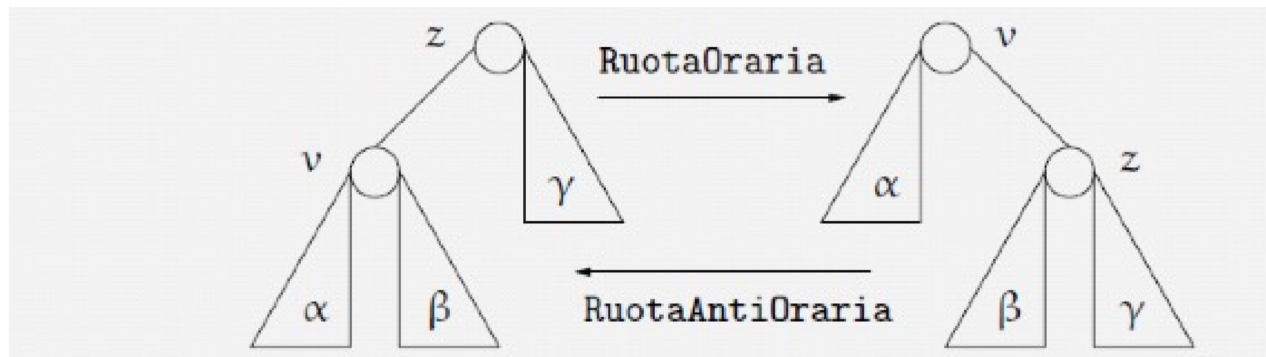
Inserimento di un elemento e in un AVL



```
Altezza( u ):
  IF (u == null) {
    RETURN -1;
  } ELSE {
    RETURN u.altezza;
  }
```

Rotazioni

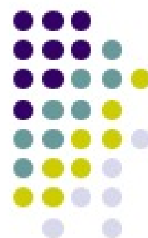





```
RuotaOraria( z ):
  v = z.sx;
  z.sx = v.dx;
  v.dx = z;
  z.altezza = max( Altezza(z.sx), Altezza(z.dx) ) + 1;
  v.altezza = max( Altezza(v.sx), Altezza(v.dx) ) + 1;
  RETURN v;
```

```
RuotaAntiOraria( v ):
  z = v.dx;
  v.dx = z.sx;
  z.sx = v;
  v.altezza = max( Altezza(v.sx), Altezza(v.dx) ) + 1;
  z.altezza = max( Altezza(z.sx), Altezza(z.dx) ) + 1;
  RETURN z;
```

Inserimento e cancellazione



- Inserimento richiede O(h) = O(log n) nel caso pessimo
- Cancellazione può essere realizzata in O(log n) nel caso pessimo
- Invece di cancellare di volta in volta i nodi, applichiamo la tecnica del *global rebuilding*:
  - invece di cancellare un nodo, lo marchiamo come cancellato logicamente
  - quando il numero di nodi marcati e` circa la metà della dimensione dell'albero, ricostruiamo completamente l'AVL
  - costo O(log n) per ogni reinserimento di un elemento non cancellato