Capitolo 5

Alberi binari di ricerca

Definizione 5.1. Un *albero binario di ricerca* è un albero binario che soddisfa le seguenti proprietà:

- ogni nodo v contiene un elemento *elem(v)* cui è associata una chiave *chiave(v)* presa da un dominio totalmente ordinato;
- le chiavi nel sottoalbero sinistro di v sono minori o uguali a *chiave(v)*;
- le chiavi nel sottoalbero destro di v sono maggiori o uguali a *chiave(v)*.

Un albero binario di ricerca è descritto dal seguente schema generale:
Dati: un albero binario di ricerca di altezza h e n nodi, ciascuno contenente coppie (*elem*, *chiave*).
Operazioni[1]

> `search(`*chiave k*`)` $\rightarrow$ *elem*
> Partendo dalla radice, ricerca un elemento con chiave k, usando la proprietà di ricerca per decidere quale sottoalbero esplorare (*complessità* $O(h)$).
>
> `insert(`*elem e, chiave k*`)` $\rightarrow$ *void*
> Crea un nuovo nodo v contenente la coppia (e, k) e lo aggiunge all'albero in posizione opportuna, in modo da mantenere la proprietà di ricerca (*complessità* $O(h)$).
>
> `delete(`*elem e*`)` $\rightarrow$ *void*
> Se il nodo v contenente l'elemento e ha al più un figlio, elimina v collegando il figlio all'eventuale padre, altrimenti scambia il nodo v con il suo predecessore ed elimina il predecessore (*complessità* $O(h)$).

5.1 Alberi AVL

Consideriamo un albero binario di ricerca con n nodi. Nel caso peggiore, la sua altezza può essere proporzionale ad n. Se invece l'albero fosse completo (o completo fino al penultimo livello) avrebbe altezza proporzionale a $\log n$. In questo caso le prestazioni delle operazioni risulterebbero molto più efficienti.

Garantire la (quasi) completezza dell'albero sarebbe molto dispendioso. Sono però stati introdotti diversi tipi di *alberi bilanciati* nei quali le operazioni di inserimento/ricerca/cancellazione risultano efficienti. Noi vedremo gli *alberi AVL*.

Definizione 5.2. Un albero è *bilanciato in altezza* se le altezze dei sottoalberi sinistro e destro di ogni nodo differiscono al più di un'unità.

Definizione 5.3. Il *fattore di bilanciamento* $\beta(v)$ di un nodo v è la differenza tra l'altezza del sottoalbero sinistro e quella del sottoalbero destro di v:

$$\beta(v) = \mathit{altezza}(\mathit{sottoalbero\_sx}(v)) - \mathit{altezza}(\mathit{sottoalbero\_dx}(v)) \tag{5.1}$$

[1] Vi rimando al libro di testo [1] per l'implementazione di queste ed altre operazioni negli alberi binari di ricerca.

Un albero AVL è un albero bilanciato in altezza che soddisfa alla proprietà:

$$| \beta(v) | \leq 1 \text{ per ogni nodo } v$$

Per l'efficienza delle operazioni di inserimento/cancellazione, ogni nodo v avrà i campi *chiave(v)*, *figlio_ dx(v)* e *figlio_ sx(v)*, tipici degli alberi binari, e il campo *altezza(v)* che memorizza l'altezza dell'albero radicato in v. Per mantenere l'albero bilanciato a seguito di cancellazioni ed inserimenti, occorre applicare delle *rotazioni* ogni qualvolta risulti necessario.

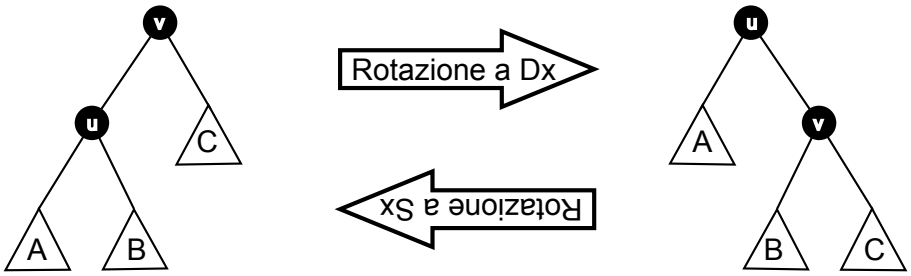


Figura 5.1: Rotazioni base

5.1.1 Ribilanciamento tramite rotazioni

Sia ora T albero AVL. Il fattore di sbilanciamento di ognuno dei suoi nodi è al massimo 1. Se cancelliamo o inseriamo un nodo in T possiamo aumentare il fattore di sbilanciamento al massimo di uno. Dobbiamo quindi effettuare le operazioni di rotazione su nodi sbilanciati il cui fattore di sbilanciamento è *esattamente* 2 in valore assoluto. In pratica si operano due rotazioni, che variano a seconda delle situazioni. Possiamo distinguere possono diversi casi:

Sinistra - sinistra: si esegue quando un nodo ha coefficiente di bilanciamento +2 ed il figlio destro un coefficiente pari a 0 o a +1 (in Figura 5.2 il nodo sbilanciato è quello con chiave 3).

Destra - destra: si esegue quando un nodo ha coefficiente di bilanciamento -2 ed il figlio sinistro un coefficiente pari a 0 o -1 (in Figura 5.2 il nodo sbilanciato è quello con chiave 6).

Sinistra - destra: si esegue quando un nodo ha coefficiente di bilanciamento -2 e il figlio sinistro un coefficiente pari a +1 (in Figura 5.2 il nodo sbilanciato è quello con chiave 3).

Destra - sinistra: si esegue quando un nodo ha coefficiente di bilanciamento +2 e il figlio destro un coefficiente pari a -1 (in Figura 5.2 il nodo sbilanciato è quello con chiave 6).

Proprietà 5.1. Una rotazione SS, SD, DS o DD, applicata ad un nodo v con fattore di bilanciamento ± 2, fa decrescere di 1 l'altezza del sottoalbero radicato in v prima della rotazione.

L'operazione di ricerca si svolge esattamente come in un albero binario di ricerca. Le operazioni di cancellazione e inserimento vengono modificate come segue:

Inserimento

- si crea un nuovo nodo e lo si inserisce nell'albero con lo stesso procedimento usato per i BST;
- si ricalcolano i fattori di bilanciamento che sono mutati in seguito all'inserimento (solo i fattori di bilanciamento dei nodi nel cammino tra la radice e il nuovo elemento possono mutare e possono essere facilmente calcolati risalendo nel cammino dalla foglia verso la radice);
- se nel cammino compare un nodo con fattore di bilanciamento pari a ± 2, occorre ribilanciare mediante rotazioni (si può dimostrare che è sufficiente una sola rotazione per bilanciare l'albero).

Cancellazione

- si cancella il nodo con il medesimo procedimento usato nei BST;
- si ricalcolano i fattori di bilanciamento mutati in seguito all'operazione (solo i nodi nel cammino tra la radice e il padre del nodo eliminato possono aver subito una modifica del fattore);
- per ogni nodo con fattore di bilanciamento pari a ± 2, procedendo dal basso verso l'alto, si opera una rotazione (può essere necessario eseguire più rotazioni, in questo caso).

5.1.2 Alberi di Fibonacci

Definizione 5.4. Un *albero di Fibonacci* di altezza h è un albero AVL di altezza h con il *minimo* numero di nodi possibile.

Un albero di Fibonacci di altezza h può essere costruito unendo, tramite l'aggiunta di una radice, un albero di Fibonacci di altezza $h-1$ e uno di altezza $h-2$; è facile verificare che ogni nodo interno di un albero di questo tipo ha fattore di bilanciamento pari a $+1$ (o -1, dipende dalla costruzione). Studiamo ora il rapporto tra numero di nodi ed altezza di un albero di Fibonacci.

Lemma 5.1. *Sia T_h un albero di Fibonacci di altezza h e sia n_h il numero dei suoi nodi; risulta $h = \Theta(\log n_h)$*

Dimostrazione. Per dimostrare la tesi, proviamo prima che $n_h = F_{h+3} - 1$, per induzione su h.

Passo base: si ha $h = 0$ e $n_0 = 1 = 2 - 1 = F_3 - 1$.

Ipotesi induttiva: supponiamo $n_k = F_{k+3} - 1$ per $k < h$.

Passo induttivo: sia $h > 0$; si ha:

$$n_h = 1 + n_{h-1} + n_{h-2} = 1 + F_{h+2} - 1 + F_{h+1} - 1 = F_{h+3} - 1$$

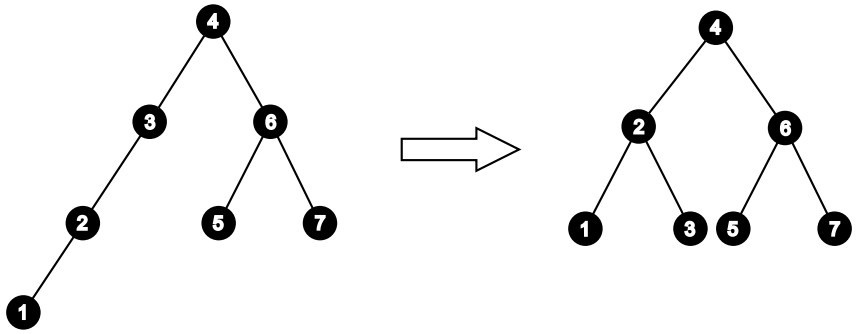
Poiché $F_h = \Omega(2^{h/2})$ e $F_h = O(2^h)$, otteniamo $n_h = \Omega(2^{h/2})$ e $n_h = O(2^h)$, da cui $2^{h/2} = O(n_h)$ e $2^h = \Omega n_h$. Quindi $h = O(\log_{\sqrt{2}} n_h)$ e $h = \Omega(\log_2 n_h)$. Possiamo concludere che $h = \Theta(\log n_h)$. □

Corollario 5.1. *Un albero AVL con n nodi ha altezza $O(\log n)$.*

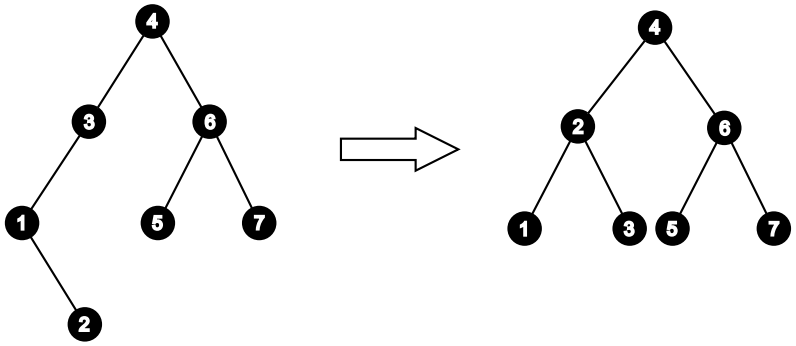
Dimostrazione. Sia T un generico albero AVL con altezza h ed n nodi. Consideriamo l'albero di Fibonacci T_h di altezza h. Sia n_h il numero dei nodi di T_h. Per definizione $n_h \leq n$. Grazie al Lemma 5.1 sappiamo $h = (\log n_h)$, quindi anche $h = (\log n)$. □

Teorema 5.1. *In un albero AVL di n nodi, le operazioni di inserimento, cancellazione e ricerca impiegano tempo $(\log n)$.*

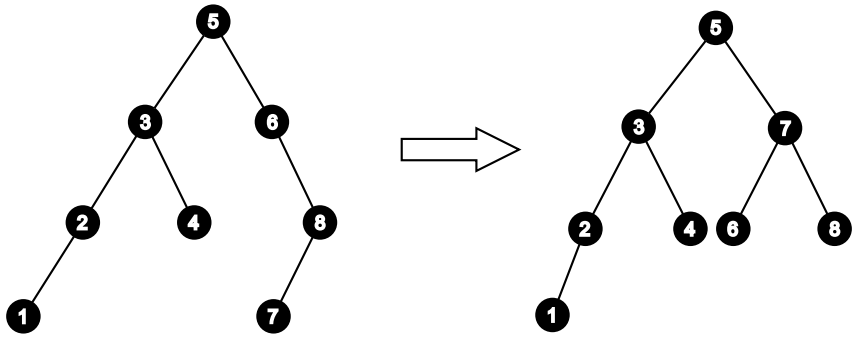
Dimostrazione. Sfruttiamo il Corollario 5.1 ricordando che in un albero binario di ricerca di altezza h le operazioni di inserimento, cancellazione e ricerca richiedono tempo $O(h)$. □



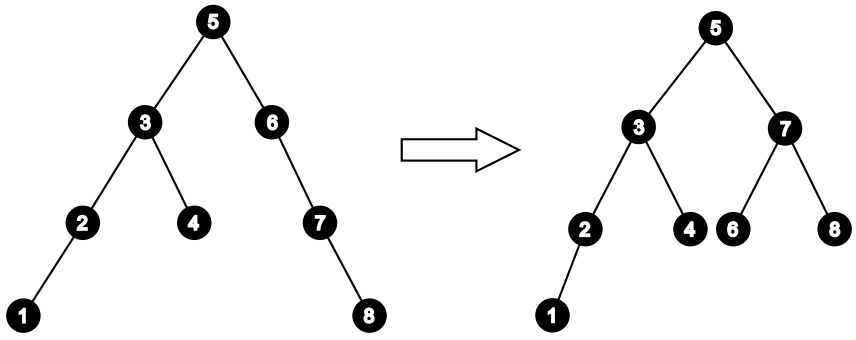
(a) Rotazione SS



(b) Rotazione SD



(c) Rotazione DS



(d) Rotazione DD

Figura 5.2: Ribilanciamento con rotazioni