Algoritmi e Strutture Dati

Corso di Laurea in Ingegneria dell'Informazione
Sapienza Università di Roma – sede di Latina

Fabio Patrizi

Dipartimento di Ingegneria Informatica, Automatica e Gestionale (DIAG)
Sapienza Università di Roma – Italy
www.diag.uniroma1.it/~patrizi
patrizi@diag.uniroma1.it

Alberi binari di ricerca

Il tipo astratto $Dizionario$

Riprendiamo il tipo di dato astratto $Dizionario$

tipo $Dizionario$:
dati: insieme finito $S \subseteq Chiave \times Elemento$ ($Chiave$ è totalmente ordinato)
operazioni:

- $insert(Chiave\ c, Elemento\ e)$:
  Se $\neg\exists e'.(c, e') \in S$, aggiunge (c, e) ad S
- $delete(Chiave\ c)$:
  Se $\exists e'.(c, e') \in S$, rimuove (c, e') da S
- $search(Chiave\ c) \rightarrow E$:
  Se $\exists e.(c, e) \in S$, restituisce e, altrimenti restituisce $null$

Rappresentazione	$insert$	$delete$	$search$
Collegata	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Indicizzata (non ordinata)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Indicizzata ordinata	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$

Esistono implementazioni più efficienti?

Alberi binari di ricerca (BST)

Definition

Un *Albero Binario di Ricerca (Binary Search Tree, BST)*, è un albero binario tale che:

1. Ogni nodo v contiene una chiave, indicata con $chiave(v)$, proveniente da un dominio $Chiave$ totalmente ordinato, ed un elemento, indicato con $elem(v)$, proveniente da un insieme $Elem$
2. Per ogni nodo v e ogni nodo w del sottoalbero sinistro di v, si ha $chiave(w) \leq chiave(v)$
3. Per ogni nodo v e ogni nodo w del sottoalbero destro di v, si ha $chiave(w) > chiave(v)$

Le proprietà **??**. e **??**. sono dette "proprietà di ricerca"

SAPIENZA UNIVERSITÀ DI ROMA

Ricerca di un elemento in un BST

Grazie alle proprietà di ricerca possiamo eseguire la ricerca in maniera concettualmente analoga a $BinarySearch$

$Algoritmo\ search(chiave\ k) \rightarrow Elem$

$v \leftarrow$ radice dell'albero;
while $(v \neq null)$ **do**
    if $(chiave(v) == k)$ **then return** $elem(v)$;
    if $(chiave(v) \geq k)$ **then**
        $v \leftarrow$ figlio sinistro di v;
    else
        $v \leftarrow$ figlio destro di v;
return null;

Ricerca di un elemento in un BST

Theorem

L'algoritmo $search$ per la ricerca di un elemento in un BST ha costo temporale $\mathcal{O}(h) = \mathcal{O}(n)$, dove h è l'altezza del BST.

Proof.

Ad ogni iterazione v si sposta in basso di un livello. Poiché sono presenti h livelli, possono esserci al più h iterazioni. Inoltre, nel caso peggiore, ogni nodo ha un solo figlio, pertanto $h = n$. □

Inserimento di un elemento in un BST

- Nuovi nodi vengono inseriti come foglie
- Si procede cercando l'elemento da inserire, fino a raggiungere il nodo che ne dovrebbe essere padre
- Il nodo raggiunto diventa padre del nuovo elemento

Algoritmo $insert(chiave\ k, Elem\ e)$

Crea un nuovo nodo w con $chiave(w) = k$ e $elem(w) = e$;
if (L'albero è vuoto) **then**
    rendi w radice dell'albero;
    return ;
$v \leftarrow$ radice dell'albero
if $(k == chiave(v))$ **then** return null;
if $(k < chiave(v))$ **then**
    if (v non ha figlio sx) **then** rendi w figlio sx di v;
    else esegui $insert(k, e)$ sul sottoalbero sx di v;
else
    if (v non ha figlio dx) **then** rendi w figlio dx di v;
    else esegui $insert(k, e)$ sul sottoalbero dx di v;

Inserimento di un elemento in un BST

Theorem

L'algoritmo $insert$ per l'inserimento di un elemento in un BST ha costo temporale $\mathcal{O}(h) = \mathcal{O}(n)$, dove h è l'altezza del BST.

Proof.

Ogni chiamata ricorsiva viene effettuata su un albero con dimensione pari a quella dell'albero di input, ridotta di 1. Sono sufficienti pertanto h chiamate, ciascuna di costo costante. □

SAPIENZA UNIVERSITÀ DI ROMA

Cancellazione di un elemento da un BST

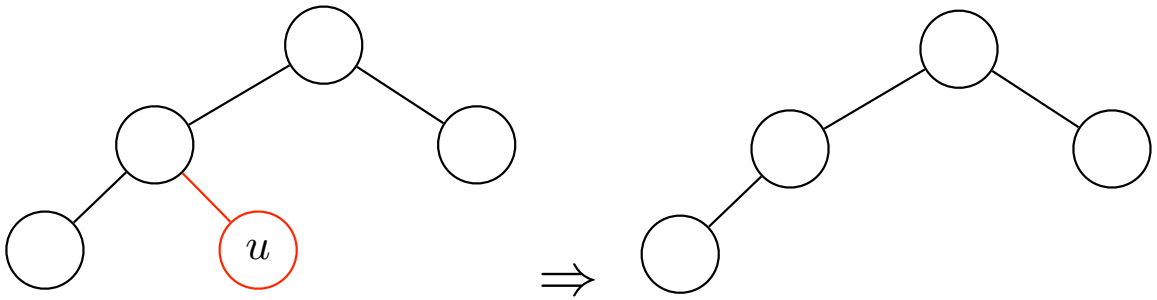
Sia u il nodo da eliminare, abbiamo tre casi:

1. Se u è una foglia, si procede alla cancellazione
2. Se u ha un solo figlio, si connette il padre di u al figlio di u
3. Se u ha entrambi i figli, si procede come segue:
   - si individua il nodo v con chiave massima del sottoalbero sx
   - si copiano $chiave(v)$ ed $elem(v)$ in u
   - si elimina v (applicando i casi **??** o **??**)

SAPIENZA UNIVERSITÀ DI ROMA

Cancellazione di un elemento da un BST: Caso 1

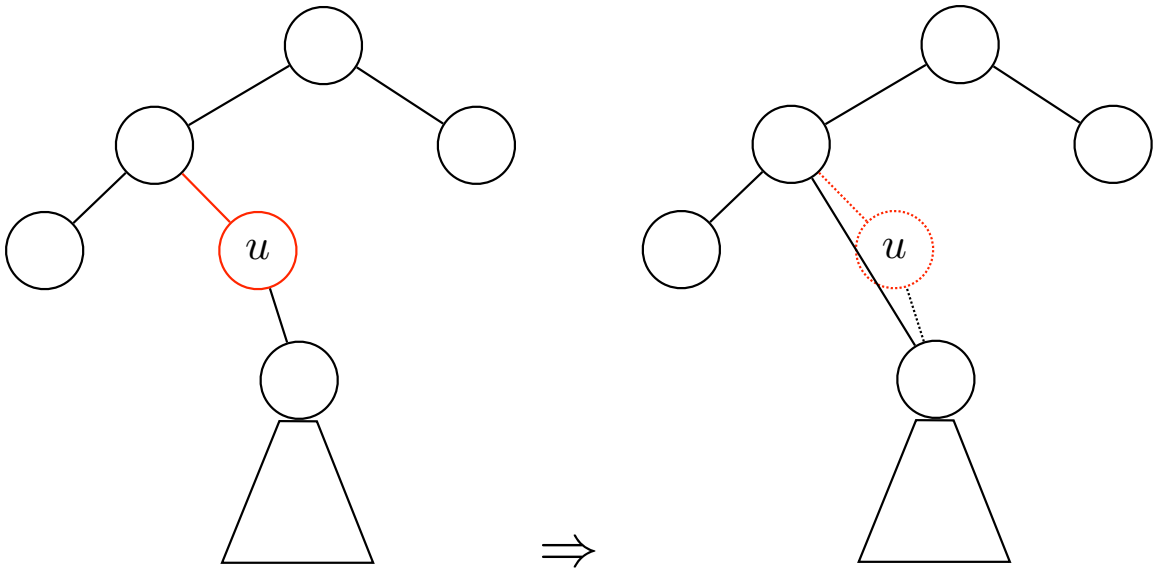
Il nodo u da eliminare è una foglia



È immediato vedere che la cancellazione non altera le proprietà di ricerca, in quanto l'ordinamento relativo tra i rimanenti nodi è invariato

Cancellazione di un elemento da un BST: Caso 2

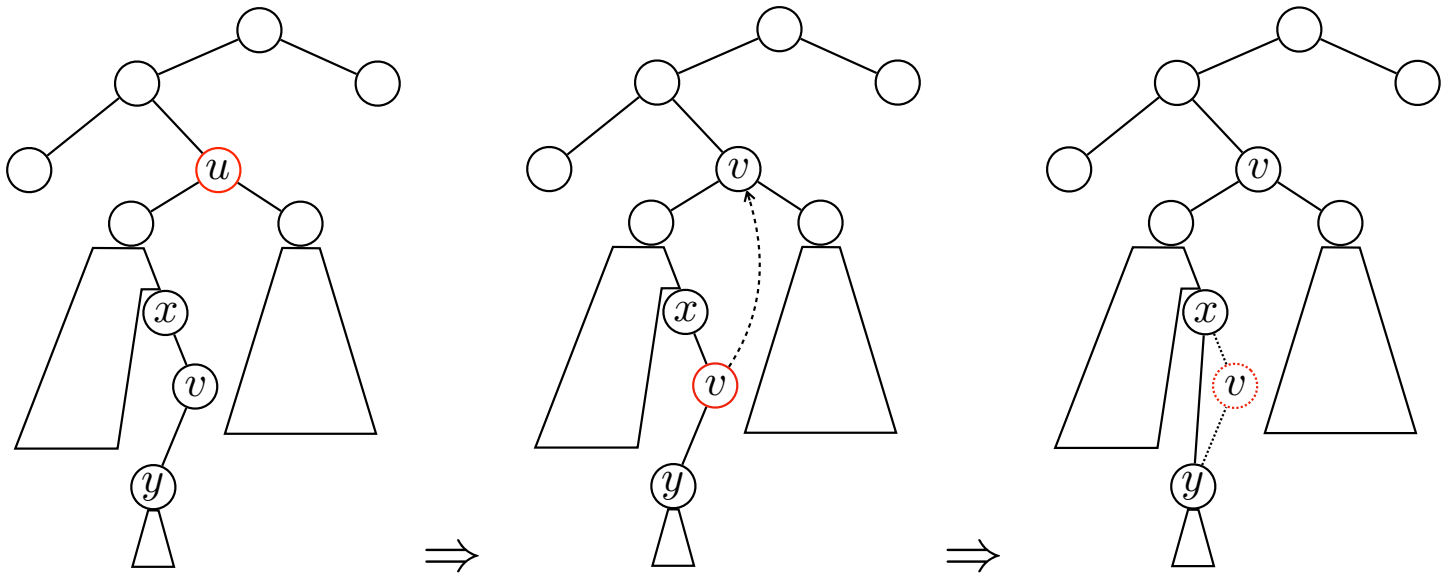
Il nodo u da eliminare ha un solo figlio



Anche in questo caso, l'ordinamento relativo tra i nodi rimanenti dopo la cancellazione è invariato

Cancellazione di un elemento da un BST: Caso 3

Il nodo u da eliminare ha entrambi i figli



Si noti che v ha chiave minore di u ma maggiore di quella di tutti i nodi del sottoalbero sx di u. Pertanto lo spostamento di v al posto di u non compromette le proprietà di ricerca.

Cancellazione di un elemento da un BST

Assumiamo che ci sia un solo nodo con chiave k

Algoritmo delete(chiave k)

individua il nodo u con chiave k;
if (u è una foglia) **then**
    elimina u dall'albero;
    return ;
if (u ha un solo figlio w) **then**
    individua il padre z di u e rendi z padre di w, al posto di u;
    return ;
individua il nodo v con chiave massima tra i nodi del sottoalbero sx di u;
assegna al nodo u $chiave(v)$ ed $elem(v)$;
rimuovi il nodo v applicando uno dei casi precedenti;

SAPIENZA UNIVERSITÀ DI ROMA

Cancellazione di un elemento da un BST

Theorem

L'algoritmo $delete$ per la cancellazione di un elemento in un BST ha costo temporale $\mathcal{O}(h) = \mathcal{O}(n)$, dove h è l'altezza del BST.

Proof.

- Individua nodo u con chiave k (e memorizzane nodo padre): $\mathcal{O}(h)$
- Elimina foglia (a partire da nodo padre): $O(1)$
- Elimina nodo con un solo figlio (a partire dal nodo padre): $O(1)$
- Individua nodo v con chiave max nel sottoalbero sx di u: $\mathcal{O}(h)$
- Assegna campi al nodo u: $\mathcal{O}(1)$
- Rimuovi nodo v: $\mathcal{O}(h)$

Complessivamente: numero costante di operazioni di costo $\mathcal{O}(h)$. □

SAPIENZA UNIVERSITÀ DI ROMA

Implementazione Dizionario tramite BST

- Nell'implementazione di un Dizionario tramite BST, tutte le operazioni hanno costo $\mathcal{O}(h)$, ovvero $\mathcal{O}(n)$, potendo essere il BST arrangiato arbitrariamente
- Sembrerebbe pertanto che un'implementazione indicizzata ed ordinata possa addirittura essere più conveniente, in quanto tutte le operazioni hanno costo $\mathcal{O}(n)$ eccetto search, che ha costo $\mathcal{O}(\log n)$
- Se riuscissimo a limitare l'altezza dell'albero che ospita il Dizionario, ad esempio garantendo $h = \mathcal{O}(\log n)$ potremmo però ottenere un'implementazione significativamente più efficiente

SAPIENZA UNIVERSITÀ DI ROMA

Alberi AVL (Adelson-Velsky, Landis)

Fattore di bilanciamento di un nodo

Definition

Dato un albero $T = (N, A)$ ed un nodo $n \in N$, si definisce *fattore di bilanciamento di* n il valore

$$\beta(n) = altezza(sx(n)) - altezza(dx(n)),$$

ovvero la differenza tra l'altezza del sottoalbero sx del nodo n e l'altezza del sottoalbero dx del nodo n

Definition

Un albero $T = (N, A)$ è detto *bilanciato* se per ogni nodo $n \in N$, si ha:

$$|\beta(n)| \leq 1$$

Definition

Chiamiamo *albero AVL* un albero binario di ricerca bilanciato

Altezza di un albero AVL

Theorem

Un albero AVL con n nodi ha altezza $h = \mathcal{O}(\log n)$.

Per dimostrare questo risultato, dimostriamo innanzitutto il seguente Lemma:

Lemma

Il numero minimo di nodi $N(h)$ per costruire un albero AVL di altezza h, soddisfa la seguente relazione di ricorrenza:

$$N(h) = \begin{cases} h, \text{ se } h = 0, 1 \\ 1 + N(h-1) + N(h-2), \text{ se } h \geq 2 \end{cases}$$

Altezza di un albero AVL

Per $h = 0, 1$, è immediato vedere che $N(0) = 0$ ed $N(1) = 1$, in quanto l'albero vuoto e l'albero con la sola radice sono entrambi alberi AVL.

Per $h \geq 2$, indichiamo con T_h un generico albero di altezza h contenente $N(h)$ nodi.

Poiché T_h contiene il minimo numero di nodi con cui si può costruire un albero AVL di altezza h, esso sarà necessariamente costituito dalla radice r con sottoalberi gli alberi AVL con numero minimo di nodi T_{h-1} e T_{h-2}.

Da ciò segue il Lemma: $N(h) = 1 + N(h-1) + N(h-2)$.

Altezza di un albero AVL

- Dal Lemma precedente, essendo $N(h-2) < N(h-1)$, segue che: $N(h) = 1 + N(h-1) + N(h-2) > N(h-1) + N(h-2) > 2N(h-2)$, ovvero:
$$N(h) > 2N(h-2)$$
- Svolgendo per iterazione (si noti che $N(2) = 2$):
$$N(h) > 2N(h-2) > \ldots > 2^i \cdot N(h-2i) > 2^{h/2-1} \cdot N(2) = 2^{h/2}$$
- Passando ai logaritmi: $\log_2 N(h) > h/2$, ovvero $h < 2\log_2 N(h)$
- Poiché dalla definizione di $N(h)$, per un qualsiasi albero AVL di altezza h contenente n nodi, si ha $n \geq N(h)$, allora: $h < 2\log_2 N(h) \leq n$
- da cui la tesi:
$$h = \mathcal{O}(\log n)$$

Operazioni su alberi AVL

Gli alberi AVL hanno altezza $h = \mathcal{O}(\log n)$ quindi le operazioni $insert$, $search$ e $delete$ descritte per i BST hanno costo $T(n) = \mathcal{O}(\log n)$

Tuttavia, partendo da un albero AVL ed eseguendo un inserimento o una cancellazione potremmo ottenere un albero sbilanciato

Come possiamo garantire che inserimenti e cancellazioni preservino il bilanciamento?

SAPIENZA UNIVERSITÀ DI ROMA

Rotazioni

Una *rotazione* è un'operazione che permette di riconfigurare i nodi di un BST, modificandone i fattori di bilanciamento

Dopo aver eseguito un inserimento o una cancellazione da un albero AVL, se questo risulta sbilanciato, il bilanciamento dei suoi nodi può essere ripristinato tramite opportune *rotazioni*

Osservazione: se una cancellazione o inserimento sbilanciano un nodo precedentemente bilanciato, tale nodo avrà fattore di bilanciamento pari, in modulo, a 2

Rotazioni

Per bilanciare un nodo occorre individuare la *causa dello sbilanciamento*

- Sia v un nodo tale che $|\beta(v)| = 2$
- I sottoalberi di v differiscono in altezza per un valore pari a 2 (quindi almeno uno dei suoi due sottoalberi ha altezza pari a 2)
- Distinguiamo 4 casi:

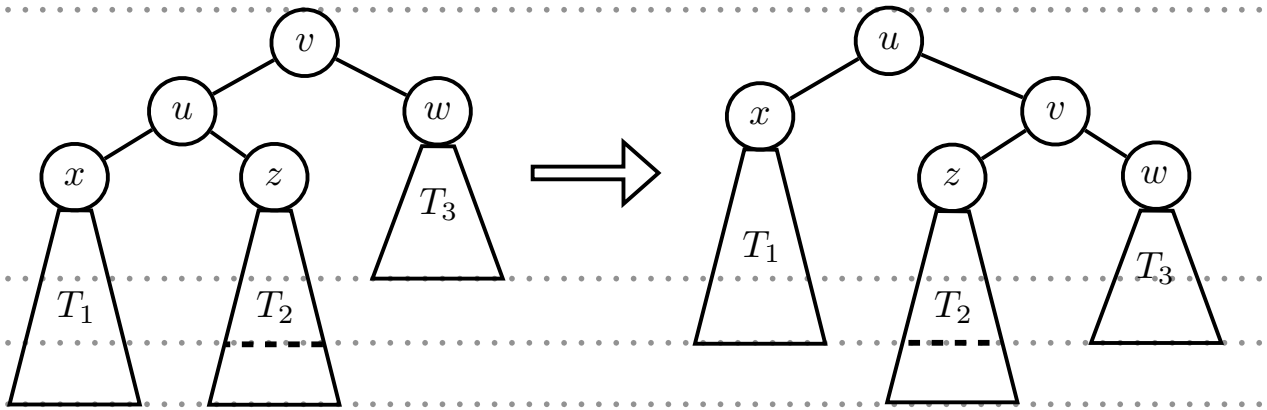
(SS) il sottoalbero sx di $sx(v)$ contiene una foglia a massima profondità
(DD) il sottoalbero dx di $dx(v)$ contiene una foglia a massima profondità
(SD) il sottoalbero dx di $sx(v)$ contiene una foglia a massima profondità
(DS) il sottoalbero sx di $dx(v)$ contiene una foglia a massima profondità

SAPIENZA UNIVERSITÀ DI ROMA

Rotazione semplice

La *rotazione semplice* permette di bilanciare un nodo nei casi SS e DD

Per il caso SS applichiamo una *rotazione semplice a destra* con perno il nodo sbilanciato (il caso DD è speculare).



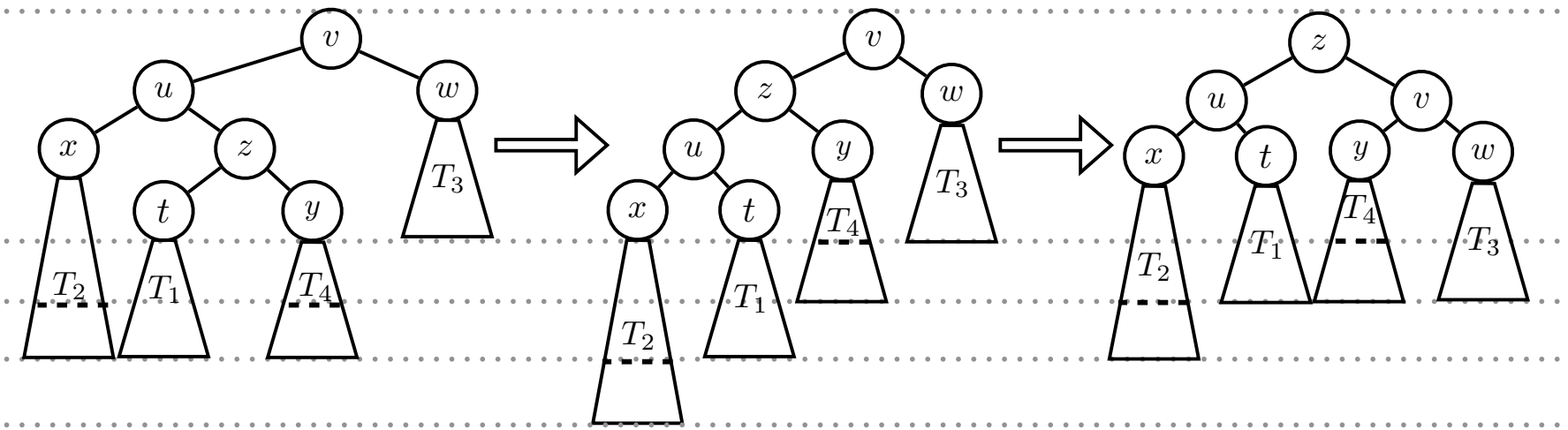
- Il nodo v risulta bilanciato
- Le proprietà di ricerca sono preservate
- Se $h(T_2) < h(T_1)$ (es., dopo inserimento), la rotazione riduce l'altezza dell'albero di 1

Rotazione doppia

La *rotazione doppia* permette di bilanciare un nodo nei casi SD e DS

Per il caso SD (il caso DS è speculare):

- applichiamo una *rotazione semplice a sinistra* con perno il figlio sx di v
- applichiamo una *rotazione semplice a destra* con perno il nodo v



- Il nodo v risulta bilanciato
- Le proprietà di ricerca sono preservate
- Se $h(T_2) \leq h(T_1)$ (es. dopo inserimento), la rotazione doppia riduce l'altezza dell'albero di 1

Inserimento

Per l'inserimento procediamo come segue:

- Inseriamo un nuovo nodo come in un BST
- Calcoliamo i fattori di bilanciamento nel cammino dalla radice al nuovo nodo
- Se esiste qualche nodo sbilanciato lungo il cammino, è sufficiente bilanciare il nodo più profondo v per bilanciare l'intero albero

Per l'ultimo punto, è sufficiente ricordare che il bilanciamento riduce di 1 l'altezza dell'albero con radice v (oltre a cambiarne la radice). Se inizialmente l'altezza è h, dopo l'inserimento diventerà $h + 1$ ma nuovamente h, a seguito del bilanciamento. Pertanto, al termine del bilanciamento, il fattore di bilanciamento degli antenati di v non sarà cambiato.

Cancellazione

Per la cancellazione procediamo come segue:

- Cancelliamo il nodo come in un BST
- Calcoliamo i fattori di bilanciamento nel cammino dalla radice al nuovo nodo. Si noti che solo i nodi lungo tale cammino possono cambiare fattore di bilanciamento
- Se esiste qualche nodo sbilanciato lungo il cammino, occorre bilanciare i nodi, partendo dal nodo più profondo v tra quelli sbilanciati

Per l'ultimo punto, si noti che la cancellazione non modifica l'altezza dell'albero con radice in v (in quanto abbassa il sottoalbero più basso). Pertanto, se inizialmente la sua altezza è h, dopo la cancellazione sarà ancora h, ma potrebbe diventare $h-1$ (nel caso in cui $h(T_2) \leq h(T_1)$) a seguito del bilanciamento. In questo caso, il fattore di bilanciamento degli antenati di v sarà cambiato e ciò potrebbe causare uno sbilanciamento.

SAPIENZA UNIVERSITÀ DI ROMA

Costo delle operazioni in un albero AVL

Theorem

Le operazioni di inserimento, cancellazione e ricerca in un albero AVL hanno costo $\mathcal{O}(\log n)$.

Proof.

Per la ricerca, il risultato è conseguenza del fatto che la ricerca in un BST ha costo $\mathcal{O}(h)$ e che in un albero AVL $h = \mathcal{O}(\log n)$.

Per l'inserimento, basta osservare che l'inserimento di un nuovo nodo come foglia ha costo $\mathcal{O}(h) = \mathcal{O}(\log n)$ ed il bilanciamento ha costo costante.

Per la cancellazione, osserviamo che la cancellazione come in un BST ha costo $\mathcal{O}(h) = \mathcal{O}(\log n)$ e che questa deve essere seguita da $\mathcal{O}(h) = \mathcal{O}(\log n)$ bilanciamenti, ciascuno di costo costante. Si ricordi infatti che occorre bilanciare solo i nodi lungo il cammino radice-v, che sono, al più, $h = \mathcal{O}(\log n)$.

□

Confronto tra le implementazioni di *Dizionario*

Rappresentazione	*insert*	*delete*	*search*
Collegata	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Indicizzata (non ordinata)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Indicizzata ordinata	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
BST	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Albero AVL	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

SAPIENZA UNIVERSITÀ DI ROMA