Corso di laurea in Informatica
Introduzione agli Algoritmi
A.A. 2024/2025

Dizionari: Alberi AVL

Tiziana Calamoneri

SAPIENZA
UNIVERSITÀ DI ROMA

Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Bilanciamento dell'altezza (1)

Un ABR di altezza h contenente n nodi supporta le operazioni fondamentali dei dizionari in tempo $O(h)$, ma purtroppo non si può escludere che h sia $O(n)$, con conseguente degrado delle prestazioni.

Viceversa, tutte le operazioni restano efficienti se si riesce a garantire che l'altezza dell'albero sia piccola, in particolare sia $O(\log n)$. Un albero con altezza logaritmica è detto **bilanciato**.

Per trasformare un ABR in un ABR bilanciato esistono varie tecniche, dette di **bilanciamento**.

Bilanciamento dell'altezza (2)

Le tecniche di **bilanciamento** sono tutte basate sull'idea di riorganizzare la struttura dell'albero se essa, a seguito di un'operazione di inserimento o di eliminazione di un nodo, viola determinati requisiti.

In particolare, il requisito da controllare è che, per ciascun nodo dell'albero, l'altezza dei suoi due sottoalberi non sia "troppo differente".

Bilanciamento dell'altezza (3)

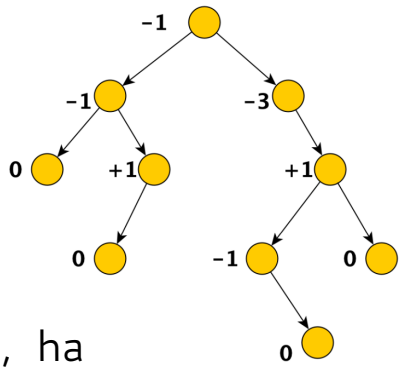
Ciò che rende non banali queste tecniche è che si vuole aggiungere agli ABR una proprietà (il bilanciamento) senza peggiorare il costo computazionale delle operazioni.

In letteratura vi sono vari approcci. Noi ne descriveremo uno.

Alberi AVL (1)

In un albero binario, il **fattore di bilanciamento** (fdb) di un nodo è dato dalla differenza tra l'altezza del suo sottoalbero sinistro e l'altezza del suo sottoalbero destro:

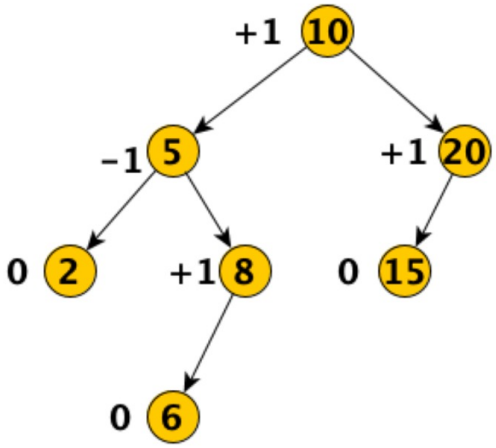
fdb(nodo)=
altezza(figlio_sinistro)-
altezza(figlio_destro).



Una foglia, in questo contesto, ha fdb pari a 0.

Alberi AVL (2)

Def. Un albero AVL è un albero binario di ricerca in cui il fdb di ciascuno dei suoi nodi sia compreso tra -1 ed 1.



Alberi AVL (3)

Teorema. Gli alberi AVL sono bilanciati.
Dim. Sia N(h) il numero minimo di nodi che può avere un albero AVL di altezza h.

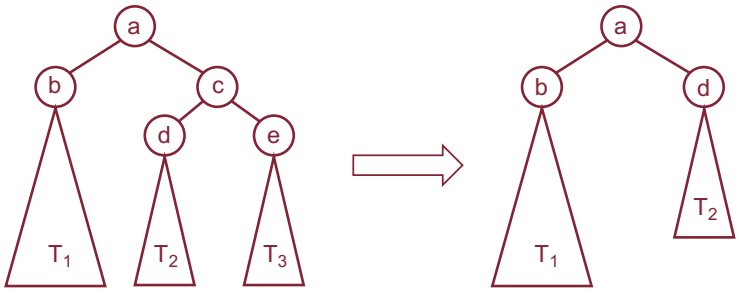
Calcoliamo N(h):

- N(0)=1

- N(1)=2

Alberi AVL (4)

…
Per h>2, i due sottoalberi della radice dell'albero di altezza h con il numero minimo di nodi non possono avere la stessa altezza, altrimenti si potrebbe ridurre il numero di nodi:



Alberi AVL (5)

…

Per calcolare il numero minimo di nodi in un albero AVL di altezza h dobbiamo, quindi, massimizzare la differenza di altezza tra i due sottoalberi mantenendo le proprietà degli AVL. Perciò l'albero con il numero minimo di nodi ha:

- un sottoalbero di altezza h-1;
- l'altro di altezza h - 2;
- ciò avviene ricorsivamente in tutti i sottoalberi.

Alberi AVL (6)

…

Possiamo quindi esprimere N(h) in modo ricorsivo:

- $N(h)=N(h-1)+N(h-2)+1$ per $h \geq 2$
- $N(0)=1$; $N(1)=2$.

Risolvendo, ad esempio con il metodo iterativo, troviamo:

$$N(h) = 2^{h/2+1}-1$$

Poiché $n \geq N(h)$:

$h \leq 2\log n$ per tutti gli AVL. CVD

Operazioni su alberi AVL (1)

Il teorema precedente garantisce che le operazioni di:

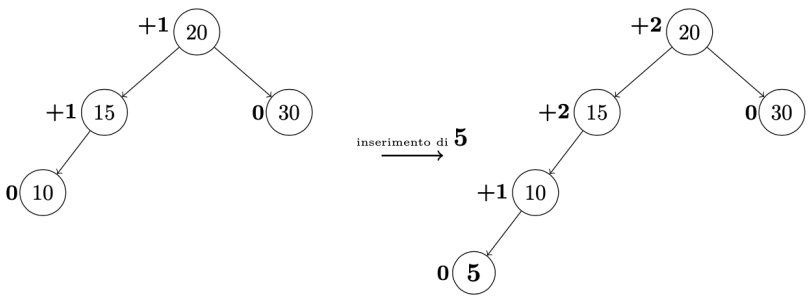
- ricerca di una chiave
- ricerca del massimo o del minimo
- ricerca del predecessore o del successore

siano identiche a quelle per gli ABR ma che siano tutte eseguite con un costo computazionale *O(log n)*.

Operazioni su alberi AVL (2)

Questo non vale automaticamente per le operazioni di inserimento e cancellazione, perché vanno mantenute le proprietà di bilanciamento dell'albero AVL!!

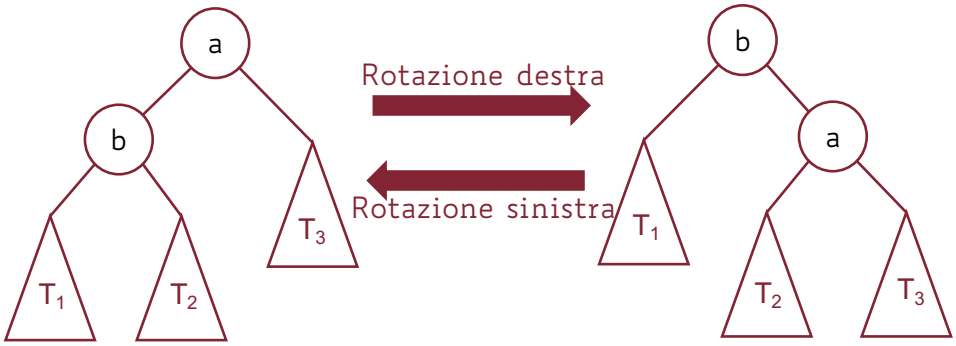
Nota. dopo un inserimento o una cancellazione, la struttura dell'albero può dover essere riaggiustata:



Rotazioni (1)

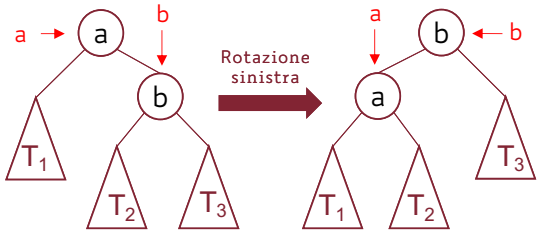
Le **rotazioni** permettono di ripristinare in tempo $O(\log n)$ le proprietà dell'albero AVL dopo un inserimento o una cancellazione.

Le rotazioni possono essere **destre** o **sinistre** e sono operazioni locali che non modificano l'ordinamento delle chiavi secondo la visita in-ordine.



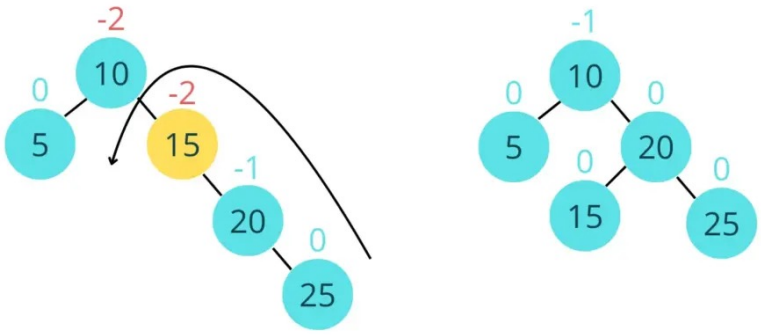
Rotazioni (2)

```
def RBA_rotaz_sinistra (p,a): #a=punt. al nodo su cui si ruota
   b = a->right
   a->right = b->left
   if a->right ≠ None:
      a->right->parent = a
   b->left = a
   b->parent = a->parent
   if a->parent==None:
      p = b
   else:
        if a==a->parent->left:
          a->parent->left = b
        else:
          a->parent->right = b
   a->parent = b
    return p
```



Costo computazionale: $\Theta(1)$. (La rotazione destra è analoga)

Rotazioni (3)

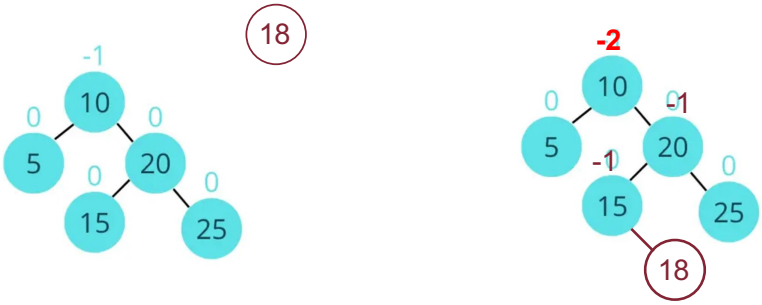


Nota: la visita in-order prima e dopo la rotazione produce lo stesso elenco di chiavi!

Inserimento (1)

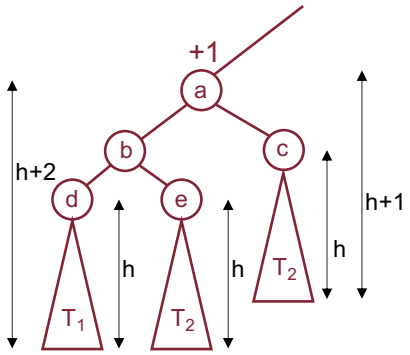
Passo Preliminare: Si inserisce l'elemento seguendo le regole dell'inserimento in un ABR, modificando il fdb ove necessario (solo sul cammino tra la radice ed il nodo appena inserito!)

Esempio:



Inserimento (2)

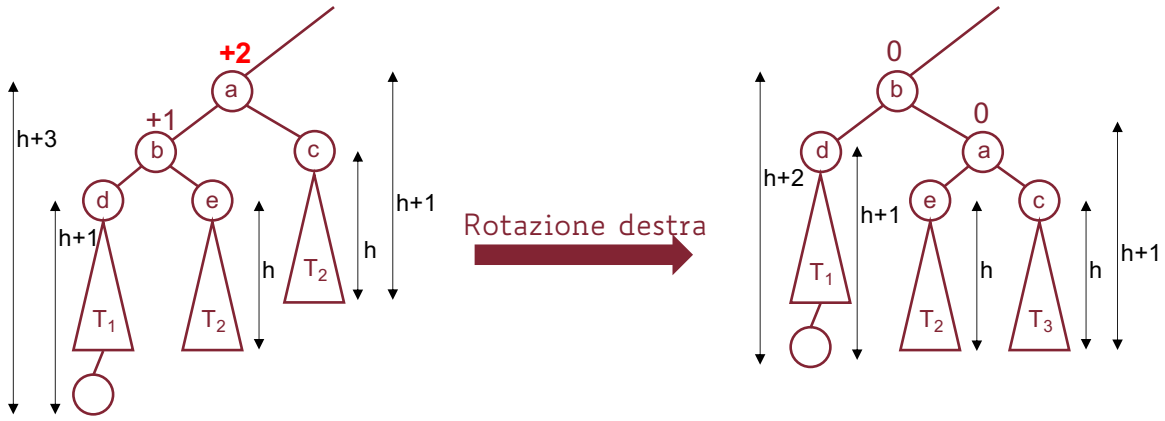
- Se l'inserimento causa uno sbilanciamento, sia a il nodo più in basso in cui questo si osserva.
- sia +2 il suo fdb



N.B.1. Il sottoalbero radicato in b ha un'altezza maggiore di T_2
N.B.2. T_1 e T_2 hanno la stessa altezza, altrimenti o a non si sbilancerebbe o si sbilancerebbe anche b, contro l'ipotesi che a sia il nodo sbilanciato più basso.

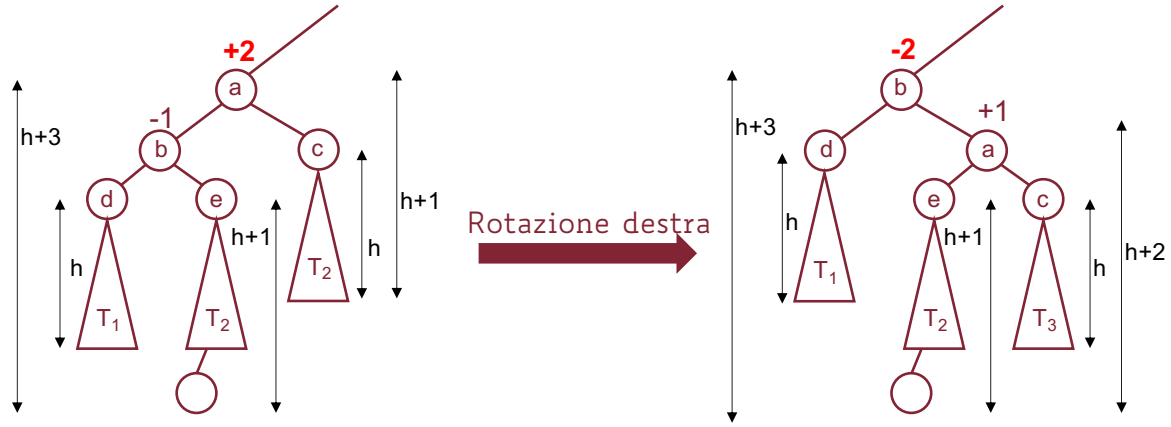
Inserimento (3)

caso 1. Il nodo che produce lo sbilanciamento viene inserito in T_1



Inserimento (4)

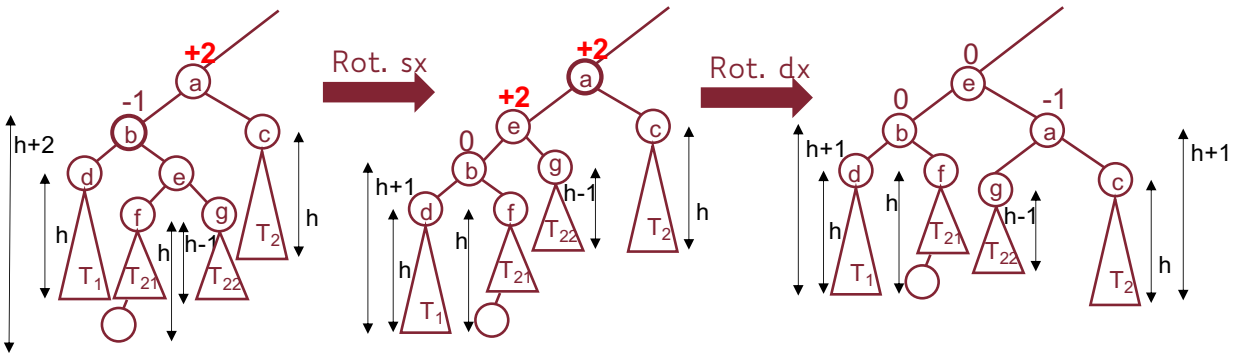
caso 2. Il nodo che produce lo sbilanciamento viene inserito in T_2



una rotazione non è sufficiente!

Inserimento (5)

caso 2. Si risolve con due rotazioni in sequenza:



Inserimento (6)

Riepilogo:

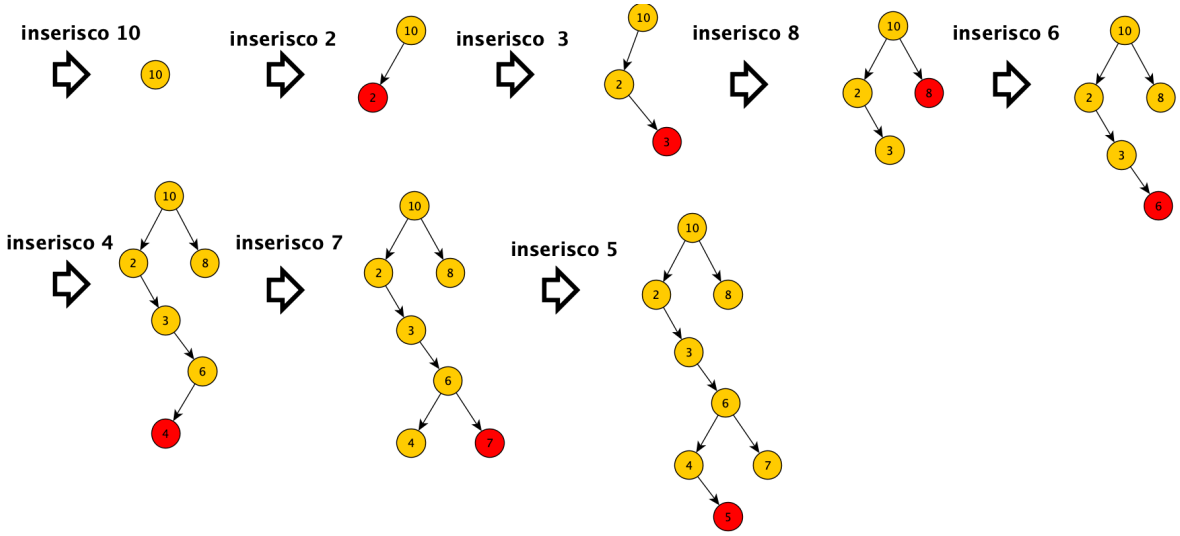
Per inserire un nuovo nodo si eseguono i seguenti passi:

1. Creazione nodo e suo inserimento:
   costo O(h)=O(log n)
2. Ricalcolo dei fattori di bilanciamento:
   costo O(h)=O(log n)
3. Verifica dell'eventuale sbilanciamento e ribilanciamento:
   costo O(1)

Totale: O(log n)

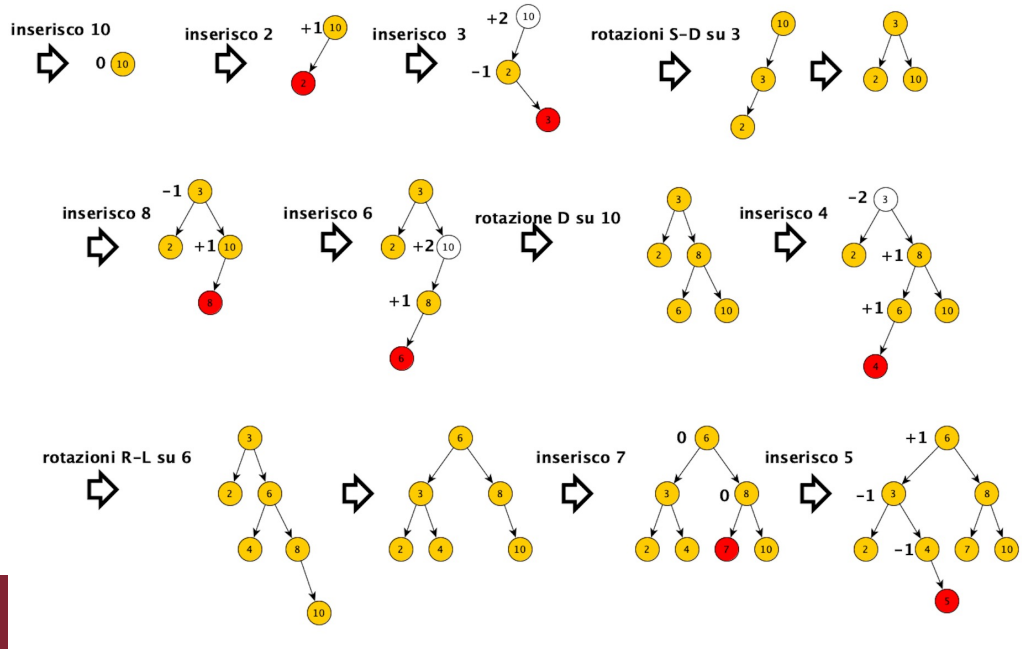
Esercizio svolto 1

Esercizio. Costruire un ABR, a partire dall'albero vuoto, inserendo successivamente le chiavi 10, 2, 3, 8, 6, 4, 7 e 5 in quest'ordine.



Esercizio svolto 2

Esercizio. Costruire un AVL, a partire dall'albero vuoto, inserendo successivamente le chiavi 10, 2, 3, 8, 6, 4, 7 e 5 in quest'ordine.



Cancellazione (1)

Anche la cancellazione in un albero AVL segue le stesse regole che in un ABR, ma è seguita da una fase di ribilanciamento necessaria per correggere eventuali sbilanciamenti causati dalla rimozione del nodo:

1. Rimozione della chiave
2. Ricalcolo dei fattori di bilanciamento
3. Verifica e ribilanciamento
4. Propagazione del ribilanciamento

Cancellazione (2)

1. Rimozione della chiave
   Lo sappiamo già fare:
   - Si cerca il nodo con chiave x - $O(h)$
   - Se il nodo da eliminare è una foglia, lo si rimuove direttamente - $\Theta(1)$
   - Se il nodo ha un solo figlio, lo si sostituisce con il figlio stesso - $\Theta(1)$
   - Se il nodo ha due figli, lo si sostituisce con il suo successore in ordine (il nodo con il valore minimo nel sottoalbero destro) e si elimina il successore - $O(h)$

Cancellazione (3)

2. Ricalcolo dei fattori di bilanciamento
   Dopo la rimozione, si aggiornano i fattori di bilanciamento risalendo il cammino dai nodi interessati, vale a dire il cammino che va dal padre del nodo eliminato alla radice dell'albero – $O(h)$

Cancellazione (4)

3. Verifica e ribilanciamento

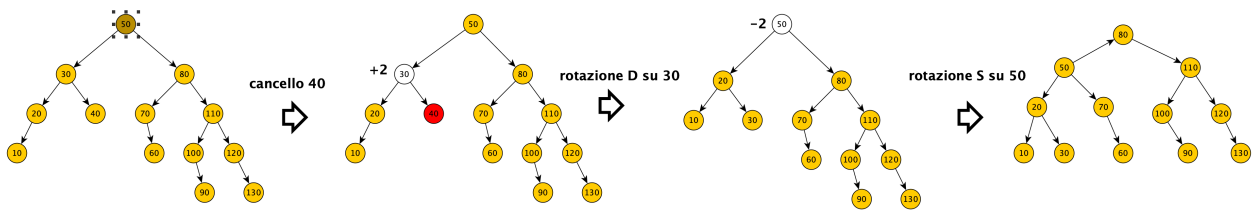
   Durante la risalita, se un nodo *a* diventa critico, ovvero con un fattore di bilanciamento pari a ±2, è necessario eseguire un ribilanciamento del sottoalbero radicato in *a*.

   I casi possibili sono simili a quelli dell'inserimento… – $\Theta(1)$

Cancellazione (5)

4. Propagazione del ribilanciamento

   A differenza dell'inserimento, dopo il ribilanciamento locale, l'altezza dell'albero può ridursi di un'unità. Pertanto, potrebbe essere necessario propagare il ribilanciamento verso i nodi antenati lungo il cammino fino alla radice – $O(h)$



Corso di laurea in Informatica
Introduzione agli Algoritmi
A.A. 2024/2025

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi (1)

- Scrivere lo pseudocodice di una funzione che ordina un array A:
  1. inserendo i suoi elementi in un ABR;
  2. ricopiando su A gli elementi incontrati eseguendo la visita inorder sull'ABR.

  Valutarne il costo computazionale. Se l'ABR è un AVL, cambia qualcosa? se si, cosa?

Esercizi (2)

- Fornire una dimostrazione alternativa del fatto che il numero di nodi presenti in un albero AVL di altezza h è limitato inferiormente da $2^{h/2+1}- 1$, utilizzando questa volta il metodo di sostituzione.
- Progettare un algoritmo per costruire un AVL a partire da un array ordinato di numeri interi. Se il numero di chiavi nell'array è n, l'algoritmo deve avere costo computazionale O(n).

Esercizi (3)

Costruire un AVL inserendo le seguenti chiavi: 4, 3, 1, 7, 6, 5, 9, 8.

Successivamente, cancellare dall'albero la radice.

Eseguire le operazioni nell'ordine specificato e mostrare dettagliatamente:

- le rotazioni effettuate (specificando se sono semplici o doppie);
- i fattori di bilanciamento aggiornati a ogni passo.

Esercizi (4)

- Progettare un algoritmo che, dato un albero di ricerca, verifichi se esso soddisfa le proprietà di un AVL. In particolare, controllare che, per ogni nodo, la differenza in altezza tra i sottoalberi sinistro e destro sia al massimo pari a 1. L'algoritmo deve avere costo O(n).

Esercizi (5)

- Progettare un algoritmo per unire due alberi AVL in uno solo, mantenendo la proprietà di bilanciamento. L'algoritmo deve avere costo $O(n_1 + n_2)$, dove n_1 ed n_2 sono il numero di nodi del primo e del secondo albero.
- Scrivere una funzione che restituisca tutte le chiavi memorizzate in un AVL che siano in un dato intervallo [a,b]. Il costo della funzione deve essere O(logn+m), dove m è il numero di chiavi restituite.