Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Proff. Campi, Ghezzi, Matera e Morzenti
Seconda prova in itinere
4 Luglio 2006

COGNOME E NOME (IN STAMPATELLO)

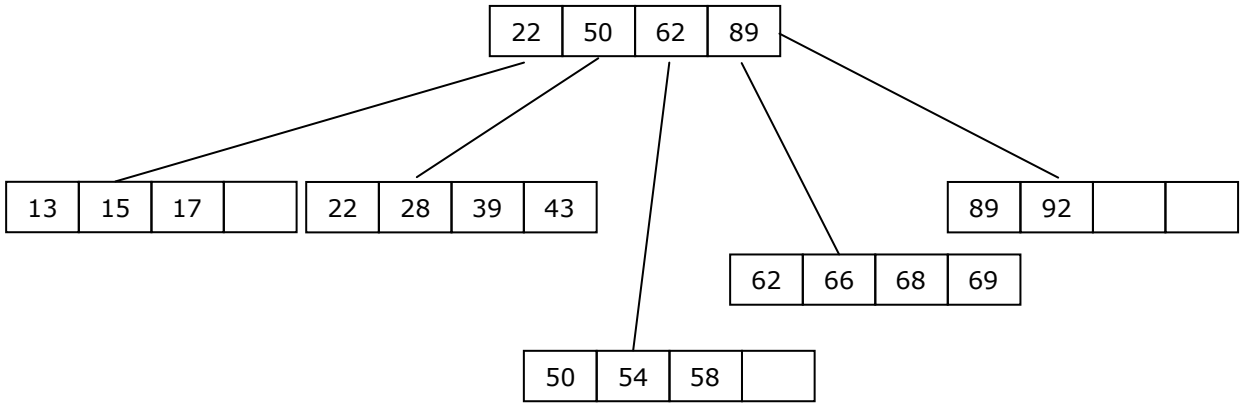
MATRICOLA

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1: B+-tree

Sia dato il seguente B+-tree di ordine 5.

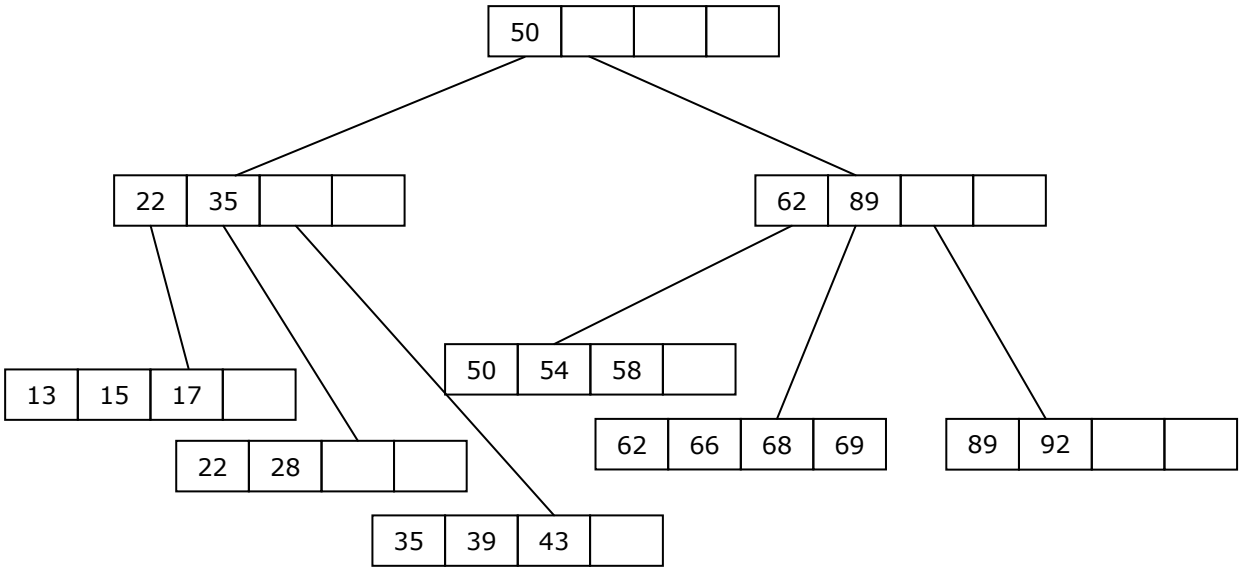


Si mostri il risultato delle seguenti operazioni:

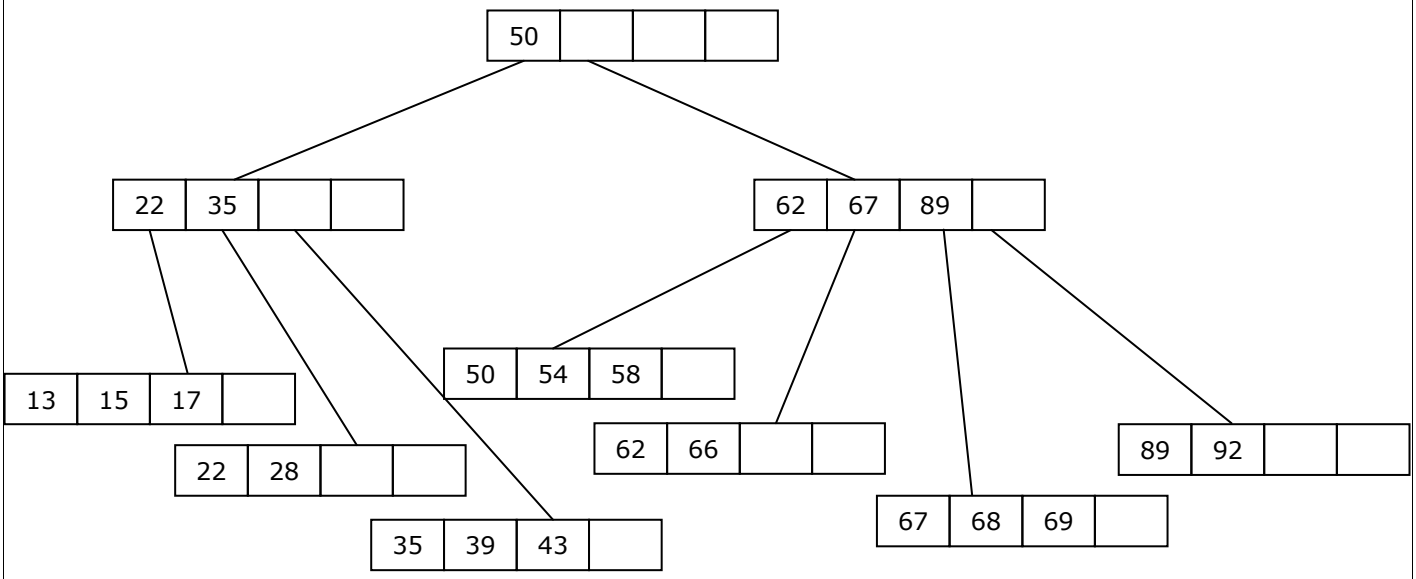
- Inserisci 35
- Inserisci 67
- Cancella 92
- Cancella 89
- Cancella 22 (ridistiribuendo in caso di necessità verso destra)
- Cancella 43

Soluzione

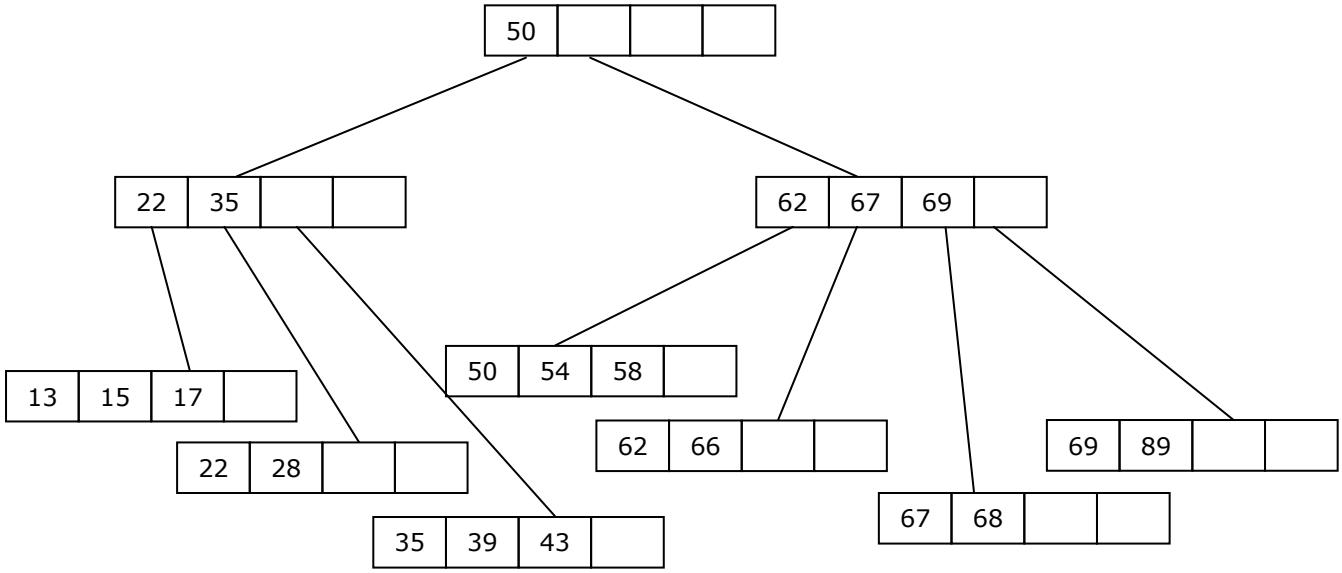
- Inserisci 35



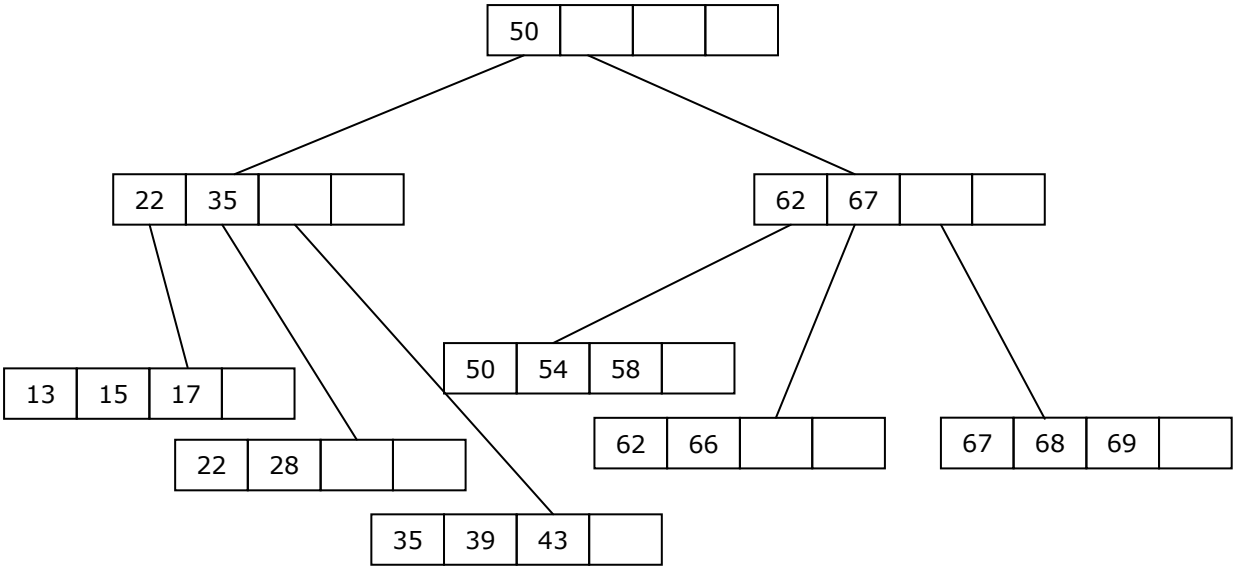
- Inserisci 67



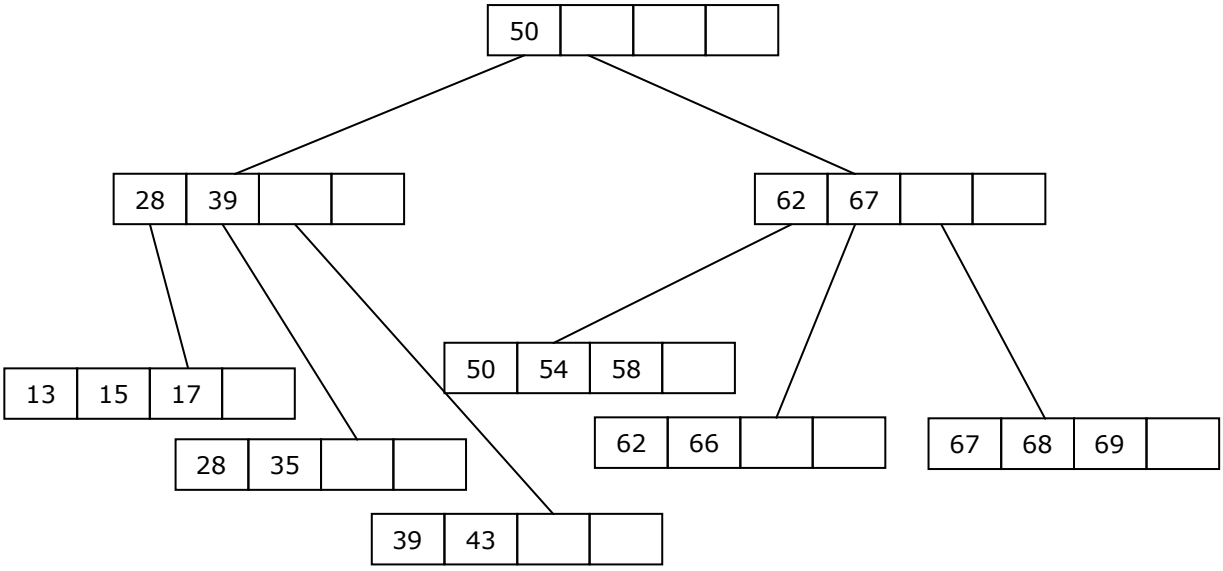
- Cancella 92



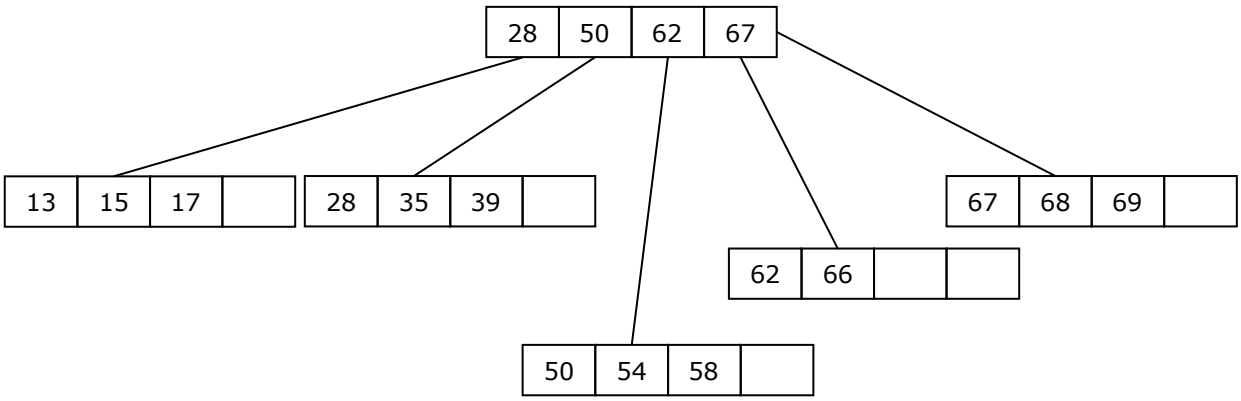
Cancella 89



Cancella 22
(ridistribuendo in caso di necessità verso destra)



Cancella 43



Esercizio 2: Tabelle di Hash.

Quesito 1.
Si consideri una tabella di hash con M=11 (e quindi indici da 0 a 10), chiave intera positiva, funzione di hash definita come segue

h(K) = "somma delle cifre che compongono la rappresentazione decimale della chiave K" MOD 11

e probing quadratico.

Mostrare il risultato dell'inserzione dei seguenti numeri, nell'ordine indicato

12, 86, 68, 59, 95, 21

riportando anche, per ogni elemento inserito, la sequenza di probe.
Commentare il tutto indicando il fenomeno che si verifica, la sua causa e il possibile rimedio.

K	Indice della posizione di inserimento	Sequenza di probe
12		
86		
68		
59		
95		
21		

Soluzione

K	Indice della posizione di inserimento	Sequenza di probe
12	3	3
86	4	3, 4
68	7	3, 4, 7
59	1	3, 4, 7, 1
95	8	3, 4, 7, 1, 8
21	6	3, 4, 7, 1, 8, 6

h(K) vale 3 per tutti i valori inseriti. Si verifica quindi il noto fenomeno del *clustering secondario*, causato dal fatto che il probing quadratico utilizza una funzione di probe che non dipende dal valore della chiave K ma solo da quello della *home position* h(K). Per evitare il clustering secondario occorre utilizzare l'hashing doppio, in cui la funzione di probe dipenda anche dal valore della chiave.

Esercizio 2 (continua)

Quesito 2.
Sia dato uno schema di hashing in cui:

- A partire da una chiave alfabetica, si genera una chiave numerica binaria concatenando la codifica ASCII su 7 bit dei caratteri della chiave.
- Il numero binario così ottenuto è dato in input alla funzione di hash h(k) = k mod 128, utilizzata per indicizzare una tabella di 128 posizioni.

Si discuta la bontà di tale schema di hashing, o alternativamente si descrivano brevemente schemi migliori per la conversione di chiavi alfabetiche.

Soluzione
La funzione h(k)= k mod 128 agisce sui 7 bit di ordine più basso della chiave numerica. Questo significa che l'hashing dipenderà esclusivamente dall'ultimo carattere della chiave alfabetica. Per ottenere una buona distribuzione delle chiavi calcolate da h, la funzione deve invece dipendere il più possibile da tutti i caratteri della chiave.
Soluzioni più appropriate sono:
- sommare i valori delle codifiche ASCII dei caratteri della chiave alfabetica;
- considerare i 7 bit intermedi.

Esercizio 3.

Si consideri il seguente problema: dato un array contenente un insieme di *n* numeri interi tutti distinti e disposti in un ordine non precisato, costruire in albero di ricerca binario che contenga tutti i numeri e sia *bilanciato* (cioè abbia profondità minima).

Quesito 1:
Si dimostri che tale problema possiede un *lowerbound* pari a $\Omega(n \cdot \log n)$.

Soluzione
Se esistesse, per il problema in questione, un algoritmo con complessità inferiore a $\Theta(n \cdot \log n)$ (e. assumiamo, almeno lineare) allora a partire da esso sarebbe possibile definire un algoritmo con la stessa complessita` per il problema dell'ordinamento. Infatti, a partire dall'albero di ricerca binario è possibile produrre una sequenza ordinata del contenuto dell'albero mediante una semplice visita dell'albero in ordine simmetrico da sinistra a destra, che ha costo lineare.

Quesito 2:
Facendo riferimento all'interfaccia `BinNode` e alle sue implementazioni utilizzate a lezione e sul libro di testo, si codifichi un metodo statico

`BinNode array2BalancedBST(int [] a)`

che, ricevendo come parametro l'array, restituisce al chiamante un riferimento alla radice di un albero binario di ricerca bilanciato che ne contiene tutti gli elementi.
[**Suggerimento:** ordinare l'array di partenza e utilizzare un metodo ausiliario, basato sulla strategia divide et impera che, a partire da un segmento dell'array individuato da una coppia di indici, produce un albero di ricerca binario bilanciato che contiene gli elementi di quel segmento...]. Si scelga l'algoritmo in modo che abbia complessità asintotica minimale.

Soluzione

```
BinNode array2BalancedBST(int [] a){
      sort(a);
      return buildHelp(a, 0, a.length-1);
}

BinNode buildHelp (int [] a, int l, int r){
      if (l>r) return null;
            elsif (l==r) return new LeafNode(a[l]);
            else {
                  int mid = (l+r)/2;
                  return new IntlNode ( a[mid],
                                        helpBuild(a, 0, mid-1),
                                        helpBuild(a, mid+1, r) );
                  }
}
```

Esercizio 3 (continua)

Quesito 3:
Si valuti la complessità asintotica del metodo codificato al punto 2.

Soluzione
La complessità dell'esecuzione di `buildHelp` è data dalla soluzione della seguente equazione alle ricorrenze

$$T(n) = 2 \cdot T(n/2) + C \quad \text{per } n>1$$
$$T(1) = K$$

che notoriamente ha soluzione lineare, quindi la complessità della costruzione dell'albero è determinata da quella dell'operazione iniziale di ordinamento, cioè $\Theta(n \cdot \log n)$.

Quesito 4:
Con riferimento ai quesiti 1-3 precedenti, si stabilisca se il problema enunciato nella premessa si può considerare chiuso.

Soluzione
L'algoritmo definito al punto 2 ha complessità $\Theta(n \cdot \log n)$ e al punto 1 si è mostrato che tale complessità costituisce un lowerbound per il problema; questo è pertanto chiuso.