Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Proff. Ghezzi, Lanzi, Matera e Morzenti
Appello del 18 Luglio 2005
**Recupero II Parte**

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

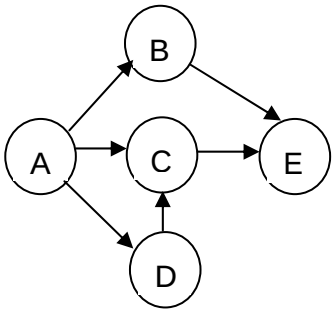
COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

Spazio riservato ai docenti

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |

**Esercizio 1.** Dato il seguente grafo orientato:




a. Si elenchino i suoi *ordinamenti topologici*.

**Risposta:**
ABDCE
ADCBE
ADBCE

b. Si dica, motivando la risposta, in che modo l'aggiunta di un arco da **C** ad **A** modifica il numero di ordinamenti topologici precedentemente individuati.

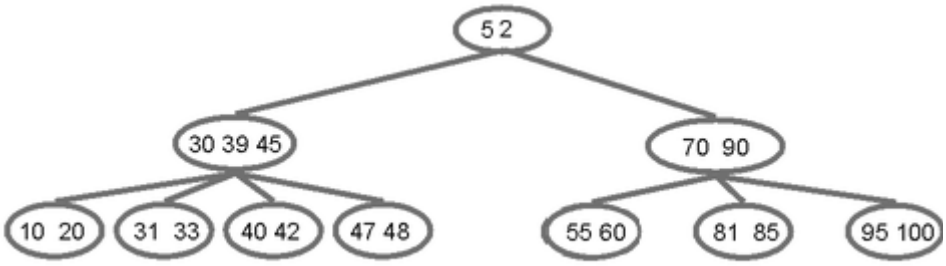
**Risposta:**
L'aggiunta di un arco da **C** ad **A** introduce un ciclo; quindi non esistono più ordinamenti topologici.

c. Si dica, motivando la risposta, se esiste un attraversamento *depth-first* che corrisponde a uno degli ordinamenti topologici individuati al punto a.

**Risposta:**
**E** deve necessariamente essere l'ultimo nodo visitato nell'ordinamento topologico. **E** è infatti l'unico nodo a non avere archi uscenti.
Poiché nessun attraversamento di tipo depth-first ha **E** alla fine, il depth-first non corrisponde in nessun caso a un ordinamento topologico.

**Esercizio 2.**

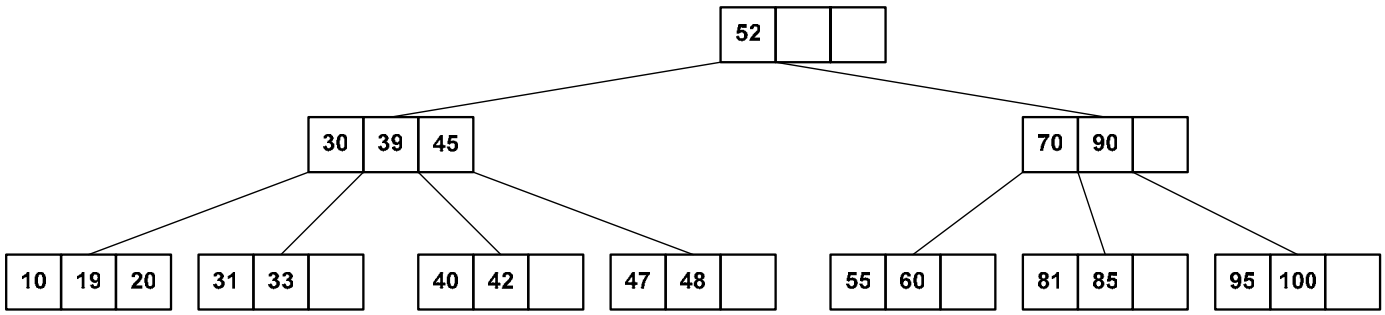
**Quesito 1.**



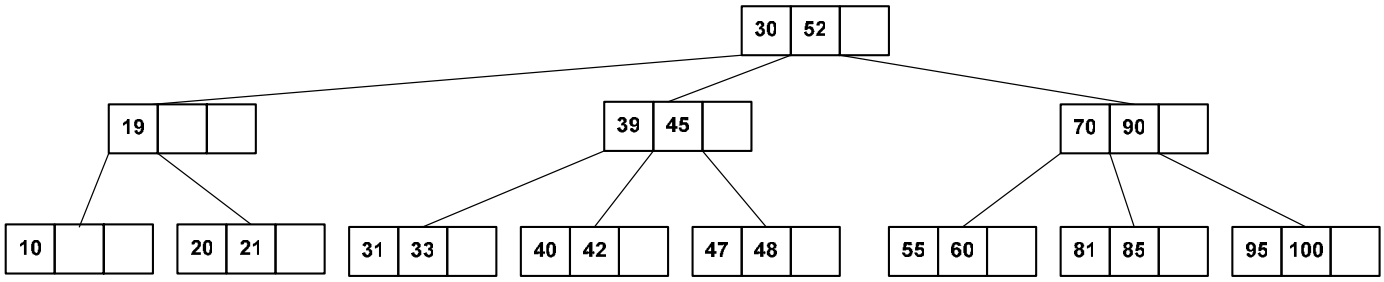

Si consideri il precedente B-albero di ordine 4 e si mostri il risultato dopo ciascuna delle seguenti operazioni:

1. Inserisci 19
2. Inserisci 21
3. Inserisci 23
4. Elimina 90
5. Elimina 30
6. Elimina 81

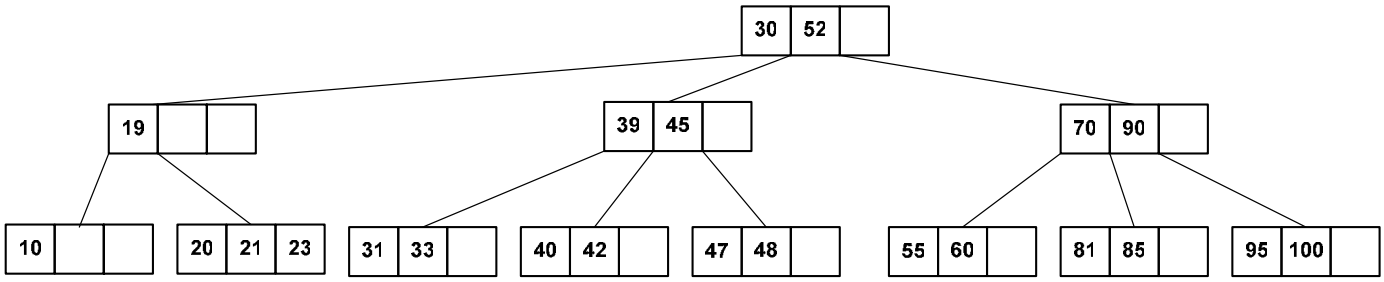
Inserisce il 19 nella foglia.




Inserendo il 21 la foglia deve essere suddivisa, e l'elemento centrale fra 10, 19, 20 e 21 deve essere spostato verso l'alto. In questo caso, non essendo un unico elemento centrale ma due, come convenzione spostiamo verso l'alto il 19. Ma anche il nodo interno deve essere suddiviso e l'elemento centrale deve essere spostato verso l'alto. Seguendo la precedente convenzione, fra 19, 30, 39 e 45, spostiamo verso l'alto l'elemento 30.

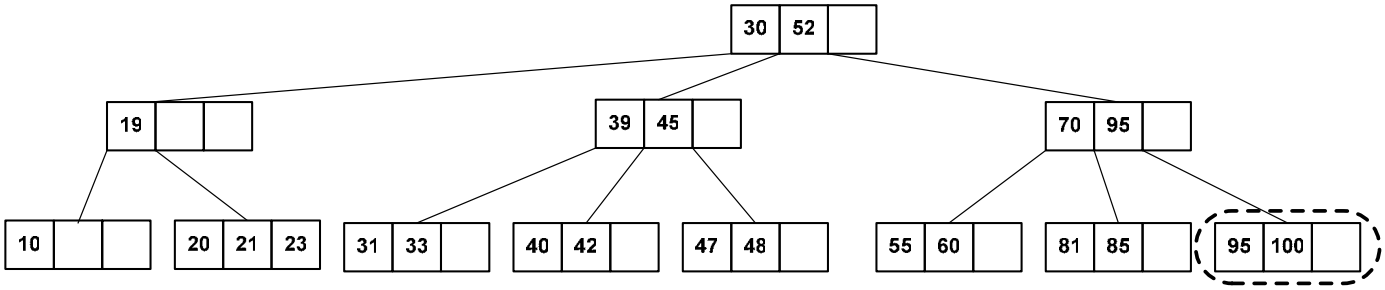



Inserire il 23 non comporta problemi, la foglia ha spazio.

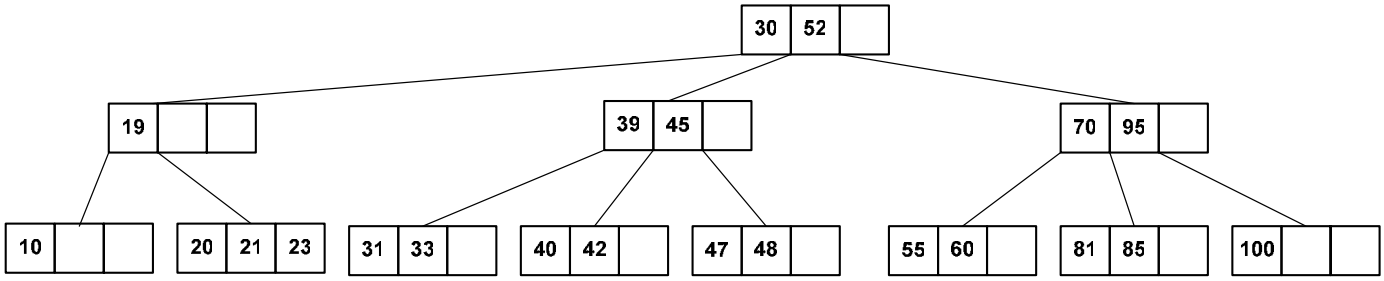


Per eliminare il 90 che e' un elemento interno e non sulla foglia, (1) si trova il successore all'interno del B-albero, in questo caso il 95, (2) si sostituisce il 90 con il 95 nel nodo interno, e infine (3) si cancella il successore (ovvero il 95) dalla foglia.

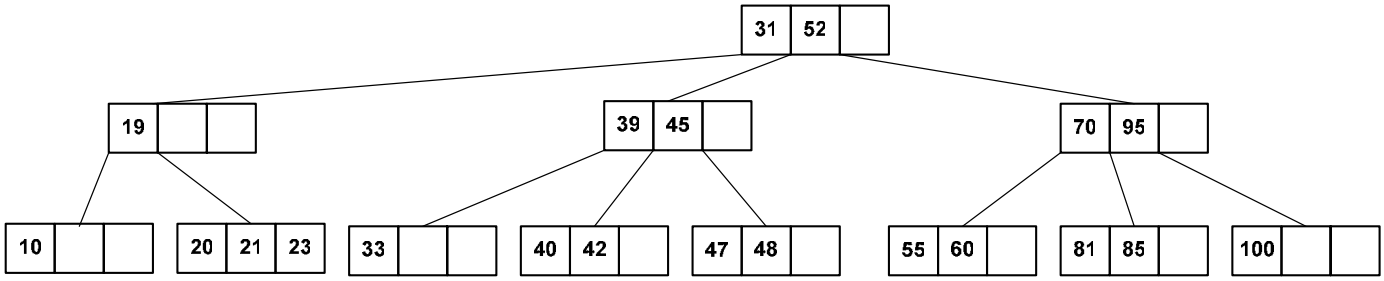
Quindi, rispetto all'albero precedente, eliminare il 90 dall'albero e' equivalente a eliminare il valore 95 dalla foglia evidenziata con un rettangolo tratteggiato del B-albero sottostante.



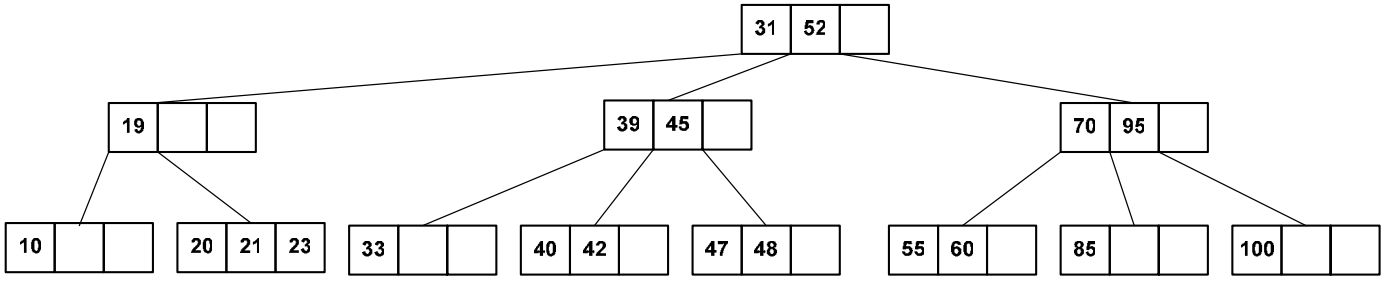

Eliminando il 95 dalla foglia, l'albero diventa:




Per eliminare il 30 si ripete il ragionamento di prima: (1) si trova il valore immediatamente maggiore di 30 presente nell'albero (in questo caso il 31), (2) si sostituisce il valore da cancellare con il suo successore (il 30 nella radice viene sostituito con 31), e infine il valore 31 nella foglia viene eliminato.




L'elemento 81 e' presente in una foglia di due elementi e quindi puo' essere cancellato direttamente.




**Quesito 2.**
Si supponga di voler implementare il B-albero come una struttura dati in memoria centrale, e che

**Esercizio 2 (continua).**

**Quesito 2.**
Si supponga di voler implementare il B-albero come una struttura dati in memoria centrale, e che pertanto il nodo dell'albero in questione sia definito da una classe Java di nome *Btree4*. Si definiscano gli attributi necessari per memorizzarne i valori contenuti nei nodi (ipotizzando che essi siano semplicemente degli interi, come in figura) e i riferimenti ad altri nodi.

```
class Btree4 {









}
```

**Risposta:**

```
class Btree4 {
private int lkey;
private int ckey;
private int rkey;
private int numKeys;
private Btree4 left;
private Btree4 centerLeft;
private Btree4 centerRight;
private Btree4 right;
}
```

**Esercizio 2 (continua).**

**Quesito 3.**

a. Si codifichi un'operazione *sumElements*, esportata dalla classe `Btree4`, che calcola la somma di tutti gli interi contenuti nell'albero.

```
class Btree4 {
…… // le definizioni inserite in risposta al quesito 2

int sumElements() {









}
}
```

b. Qual è la complessità dell'operazione di somma in funzione del numero di nodi dell'albero?
c. E in funzione del numero totale di interi?

**Risposta**

```
int sumElements() {
int s;

s = lkey + ckey + rkey;
if (left != null) s = s + left.sumElements();
if (centerLeft != null) s = s + centerLeft.sumElements();
if (centerRight != null) s = s + centerRight.sumElements();
if (right != null) s = s + right.sumElements();
return s;
}
```

La complessità dell'operazione di somma è lineare nel numero dei nodi, e anche nel numero dei valori memorizzati, perché ogni nodo contiene al più 4 valori.

**Esercizio 3.**

**Quesito 1.**
Dato un albero contenente *n* nodi e avente profondità *p*, considerati due suoi nodi distinti *u* e *v*, si definisce *minimo antenato comune* di *u* e *v* il nodo, tra gli antenati comuni a entrambi, che si trova più in basso, cioè a distanza massima dalla radice (si noti che un nodo è considerato antenato di sé stesso).

a. Si definisca un algoritmo efficiente per trovare il minimo antenato comune di due nodi nel caso di *albero di ricerca binario* (binary search tree) e lo si codifichi implementando il seguente metodo statico:

```
static BinNode minimoAntenatoComune (BinNode r, Elem el1, Elem el2)
```

dove il parametro *r* rappresenta la radice dell'albero, mentre *el1* ed *el2* i valori memorizzati nei due nodi di cui si cerca il minimo antenato comune.
Si assuma che i valori memorizzati nell'albero siano tutti distinti. Si assuma inoltre che *el1* ed *el2* siano effettivamente memorizzati in due nodi dell'albero.

**Risposta**

```
static BinNode minimoAntenatoComune (BinNode r, Elem el1, Elem el2){
  while ((r.key()>el1 && r.key()>el2) || (r.key()<el1 && r.key()<el2))
    if (r.key()>el1 && r.key()>el2)
      r = r.left();
    else if (r.key()<el1 && r.key()<el2)
      r = r.right();
  return r;
}
```

b. Si fornisca una sintetica ma convincente spiegazione della correttezza dell'algoritmo.

**Risposta**
Chiaramente, il minimo antenato comune dei due nodi è la radice del più piccolo sottoalbero che li include entrambi; inoltre, se uno dei due nodi è antenato dell'altro esso è anche il minimo antenato comune e, se nessuno dei due nodi è antenato dell'altro, allora il minimo antenato comune è il nodo più alto (più vicino alla radice) tra quelli che contengono un valore intermedio tra i due valori dati e perciò i due nodi *u* e *v* appartengono uno al suo sottoalbero sinistro e l'altro al suo sottoalbero destro. Quindi per trovare il minimo antenato comune si può percorrere un cammino che parte dalla radice e continua a scendere fintantoché si incontrano valori maggiori di entrambi el1 ed el2 (allora si deve continuare nel sottoalbero sinistro) o minori di entrambi (allora bisogna proseguire nel sottoalbero destro). Questa discesa termina quando il valore nel nodo che si è raggiunto è uguale a uno del due o è compreso tra di essi, il che rende falsa la condizione del *while*: in entrambi i casi, per quanto detto in precedenza, tale nodo è il minimo antenato comune.

c. Si valuti, motivando la risposta, la complessità dell'algoritmo nel caso pessimo, sia per un albero bilanciato sia per un albero non bilanciato.

**Risposta**
Poiché l'algoritmo scende di un livello a ogni ripetizione del corpo del ciclo while, la complessità nel caso pessimo è $\Theta(p)$, che nel caso di albero bilanciato è $\Theta(\log n)$, mentre nel caso di albero non bilanciato è $\Theta(n)$.

**Esercizio 3 (continua).**

**Quesito 2.**
Si consideri ora lo stesso problema per il caso di un albero binario non ordinato.

a. Si tratteggi informalmente ma nel modo più chiaro e preciso possibile un algoritmo che risolva tale problema.

**Risposta**
Poiché l'albero non è ordinato occorre cercare i due nodi contenenti i valori *el1* ed *el2*, durante la ricerca si costruiscono i due cammini che dalla radice portano ai due nodi. Il nodo minimo antenato comune è l'ultimo del prefisso comune a tali due cammini.

b. Si valuti, motivando la risposta, la complessità di questo secondo algoritmo nel caso pessimo per un albero bilanciato e per un albero non bilanciato.

**Risposta**
Essendo l'albero non ordinato la ricerca dei due nodi richiede in ogni caso (albero bilanciato o no) un tempo $\Theta(n)$; inoltre la costruzione dei cammini non aggiunge operazioni con costo di ordine superiore e il calcolo del prefisso comune è lineare nella lunghezza dei cammini. Quindi la complessità è $\Theta(n)$ indipendentemente dalla forma dell'albero, $\Theta(p)$ se l'albero non è bilanciato e $\Theta(2^p)$ se è bilanciato.