Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 21 Febbraio 2007 —

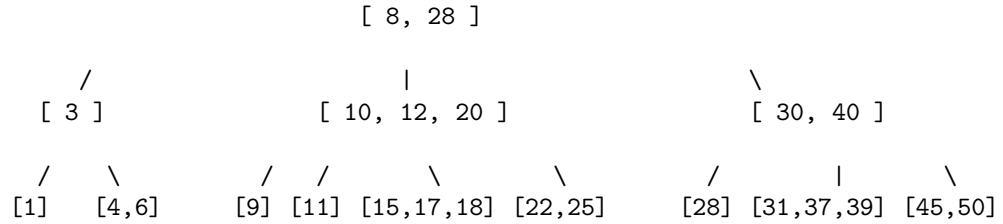
Esercizio 1 (ASD)

Si risolvano le seguenti ricorrenze, giustificando la risposta.

- $T(n) = 5T(\frac{n}{2}) + n^2 + 3n$
- $T(n) = 2T(\frac{n}{3}) + T(\frac{n}{4}) + n$

Esercizio 2 (ASD)

- Quale è la complessità di una operazione di inserimento di una chiave in un B-albero con n chiavi, in funzione di n? Giustificare la risposta.
- Si consideri il seguente albero 2-3-4 e lo si trasformi in un albero R/B.



Esercizio 3 (ASD)

Si consideri la struttura dati albero generale i cui nodi hanno gli attributi: `key`, `fratello`, `figlio` e `padre` e si sviluppi un algoritmo per verificare se è soddisfatta la seguente proprietà speciale:

`key`[`padre`[x]] > `key`[x], per ogni nodo x diverso dalla radice

Esercizio 4 (ASD)

Sia L una lista semplice (con attributi `key` e `next` e sentinella `sent`[L]) di numeri interi e $k > 0$ un numero intero. Sia inoltre new_node(c,x) una funzione che restituisce un nuovo nodo con campo key uguale a c e campo next uguale ad x. Scrivere un algoritmo che trasforma L inserendo dopo ogni k nodi un nuovo nodo che ha come chiave la somma dei k nodi precedenti.
Esempio. Assumiamo k=5. L= (3,2,1,1,5,3,4,1,2,6,1,8) viene trasformata in L'= (3,2,1,1,5,12,3,4,1,2,6,16,1,8).
Dimostrare la correttezza dell'algoritmo.

Esercizio 5 (LAB)

Sia data la seguente interfaccia, dove per tutti i metodi con parametro di tipo `Node` assumiamo la precondizione p != null, e per tutti i metodi che restituiscono un `Node`, utilizziamo il valore null quando il metodo sarebbe indefinito (ad esempio, per `parent()` applicato alla radice, o `root()` applicato ad un albero vuoto).

```
interface BinTree  {
      Node root();               // la radice dell'albero
      Node parent(Node p);       // padre di p nell'albero
      Node left(Node p);         // figlio sinistro di p nell'albero
      Node right(Node p);        // figlio destro di p nell'albero
      boolean internal(Node p);  // true sse p e' un nodo interno
      boolean leaf(Node p);      // true sse p e' una foglia
  int key(Node p);           // la chiave contenuta nel nodo p
}
```

Definite, in Java, l'implementazione del metodo `prec()` descritto dalla seguente specifica.

```
     /**
      * PRE:  T != null e' un BinTree quasi completo, p un Nodo di T
      *
      * POST: restituisce un riferimento al Node che precede p in una visita in
      *       ampiezza di T che attraversi i livelli da sinistra a  destra.
      *       Eccezione se p non ha precedenti in T
      */
      public static Node prec(BinTree T, Node p) throws NoSuchElementException
```

L'implementazione deve garantire una **complessità** $O(log\ n)$, dove n è il numero di nodi dell'albero.

Esercizio 6 (LAB)

Sia data la seguente definizione parziale della classe `List` che realizza una lista semplice.

```
class List {
      /**
       *  NOTAZIONE:
       *   - next^i[head] = il ListItem raggiunto seguendo "i" riferimenti
   *     next a partire da head.
       *   - next^0[head] = head
   *
   *  INVARIANTE DI RAPPRESENTAZIONE
       *  (a) size >= 0
       *  (b) size > 0  => next^i[head] != null (0 <= i <= size-1)
       *                   tail = next^(size-1)[head], next[tail] = null
       *  (c) size = 0 <=> head = tail = null
   *
       *  FUNZIONE DI ASTRAZIONE
       *  this  = []                    se size = 0
       *        = [x_0.... x_{size-1}] dove xi = item[next^i[head]]
   */

      private class ListItem { int item;  ListItem next; }
      private ListItem head, tail;
      private int size;

      public void reverse() {
         // POST: trasforma this invertendone l'ordine degli elemeni, ovvero
     //       se this = [x1,x2,...,xk], this_post = [xk,...,x2,x1]
      }
      // costruttori, altri metodi, ... etc
}
```

Fornite l'implementazione del metodo `reverse()`. L'implementazione deve garantire una **complessità** $\Theta(n)$, dove n è la dimensione della lista, ed utilizzare spazio costante, ovvero modificare la lista sul posto, senza creare una nuova lista.