Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 8 Giugno 2009 —

Esercizio 1 - ASD

- Giustificando la risposta, si risolva la seguente ricorrenza:
  $T(n) = 2T(\frac{n}{3}) + 3\sqrt{n^3}$
- Giustificando la risposta, si dica se la seguente affermazione è corretta.
  Per ogni funzione $f(n)$ asintoticamente positiva si ha $\Theta(\sqrt{n^3} + f(n)) = \Omega(\sqrt{n^3})$

Esercizio 2 - ASD

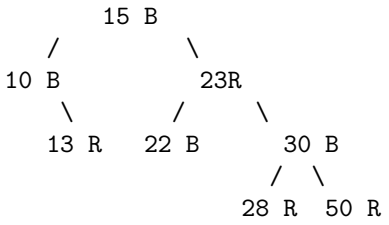
Chiamiamo *BST-foresta* una lista $F = T_1, T_2, ..., T_n$ di BST non vuoti a chiavi intere che soddisfa la seguente proprietà: per ogni i, $1 \leq i < n$, la chiave più grande memorizzata in T_i è minore o uguale alla chiave più piccola memorizzata in T_{i+1}.
Specificare la struttura dati BST-foresta (strutture dati usate, attributi dei nodi, operazioni, ...) e proporre un algoritmo per l'operazione di inserimento di una chiave k in una BST-foresta.

Esercizio 3 - ASD

Si descriva l'algoritmo per l'inserimento di un nuovo elemento in una coda di priorità Q, realizzata con un array che rappresenta un max-heap.

Esercizio 4 - ASD

1) Dire se il seguente albero binario gode della proprietà R/B. (Chiaramente si suppone che tutte le foglie-NIL lasciate implicite siano nere).



2) In caso di risposta positiva, disegnare l'albero che si ottiene dopo aver simulato l'inserimento della chiave 35.

Esercizio 1 (Laboratorio)

Una sequenza ordinata è una collezione in cui gli elementi compaiono in modo ordinato e sono ammesse più copie dello stesso elemento. Si vuole realizzare un package per rappresentare e gestire una sequenza ordinata di elementi di tipo integer. La struttura dati scelta per memorizzare la sequenza ordinata è la lista semplice circolare con sentinella. Ovviamente la lista viene mantenuta ordinata in senso crescente rispetto agli elementi della sequenza.
Data la classe:

```
package SO;
class NodoLista {
    int key;
    NodoLista next;

    NodoLista(int k) {
       key = k;
       next = null;
    }
}
```

si richiede di completare l'implementazione dei metodi *insert* e *oneOccurrence* della seguente classe. I due metodi devono avere complessità lineare rispetto al numero degli elementi in lista.

```
package SO;
public class SequenzaOrdinata {
    private NodoLista sentinel; // nodo sentinella
    private int count; // totale elementi in lista

    // post: costruisce una sequenza ordinata vuota
    public SequenzaOrdinata() {
        sentinel = new NodoLista(0); // valore convenzionale per la sentinella
        sentinel.next = sentinel;
        count = 0;
    }

    // post: inserisce l'elemento k nella sequenza ordinata e aggiorna count
    public void insert(int k) {...}

   // post: ritorna true se e solo se tutti gli elementi della sequenza ordinata
   //       occorrono una sola volta
   // NOTA: se la sequenza ordinata e' vuota ritorna true
   public boolean oneOccurrence() {...}

}
```

Esercizio 2 (Laboratorio)

Si consideri una tabella hash T[0...9] con gestione delle collisioni mediante liste di collisioni, ordinate in senso crescente rispetto alla chiave. La chiave della tabella hash è un numero reale $k \in (0, 1)$ e la funzione hash è $h(k) = \lfloor km \rfloor$, dove m è la dimensione della tabella hash. Ad esempio, per la tabella considerata, $h(0,57) = \lfloor 0,57 * 10 \rfloor = 5$. Partendo dalla tabella contenente solamente la chiave utilizzata come esempio (k = 0,57) si richiede di inserire in tabella le seguenti chiavi mostrando tutti i passaggi del procedimento:

- $0,123$
- $0,619$
- $0,24$
- $0,95$
- $0,5321$
- $0,67$

- $0,299$
- $0,91$
- $0,652$
- $0,74$

Qual è il fattore di carico della tabella?