Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 5 Giugno 2006 —

Esercizio 1 (ASD)

Considerata la ricorrenza:

$$T(n) = 10T(\frac{n}{3}) + 3n^2 + n$$

si richiede di:

- risolverla utilizzando il teorema principale;
- dire se $T(n) = O(n^2)$, giustificando la risposta.

Esercizio 2 (ASD)

Si consideri l'operazione `successore` che dato un nodo x in un albero binario di ricerca T restituisce il nodo di T che segue x in una visita in in-ordine di T. Nell'ipotesi che `successore`$[x]$ sia diverso da `NIL`, dire quali delle seguenti affermazioni sono vere e quali false, giustificando la risposta.

1. `key`$[x] >$ `key`[successore$[x]$].
2. Il nodo `successore`$[x]$ si trova nel sottoalbero destro di x.
3. Il nodo `successore`$[x]$ si trova nel sottoalbero sinistro di x.
4. Esiste un cammino dalla radice di T ad una foglia che passa per entrambi i nodi x e `successore`$[x]$.

Esercizio 3 (ASD)

Si consideri la struttura dati albero generale i cui nodi hanno gli attributi: `key`, `fratello`, `figlio` e `padre` e soddisfano la seguente proprietà speciale:

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

Si sviluppi un algoritmo in pseudo-codice che dato un nodo x di T lo elimina e restituisce un albero generale che soddisfa ancora la proprietà speciale.

Esercizio 4 (ASD + Laboratorio)

Si vuole realizzare un'implementazione del tipo di dato Dizionario mediante una tabella hash ad indirizzamento aperto, in cui le collisioni sono risolte mediante scansione lineare.
Data la classe:

```
package Esercizio4;
class Coppia {
    Comparable key;          // chiave
    Object elem;             // elemento

    // post: costruisce un nuovo elemento con chiave key e valore ob
    Coppia(Comparable k, Object ob) {
        key = k;
        elem = ob;
    }
}
```

che rappresenta una coppia (chiave, elemento) da memorizzare nel dizionario, si richiede di:

1. completare l'implementazione della seguente classe *DizHashAperto*, ipotizzando che il dizionario non ammetta chiavi duplicate:

```
package Esercizio4;
public class DizHashAperto {
    private static final int DEFSIZE = 113;     // capacita' array
    private Coppia[] diz;                       // tabella hash
    private int count;                          // totale coppie nel dizionario

    //post: costruisce un dizionario vuoto
    public DizHashAperto() { diz = new Coppia[DEFSIZE]; count = 0; }

    // pre: key diverso da null
    // post: ritorna l'indice dell'array associato a key
    private int hash(Comparable key) { return (key.hashCode() % DEFSIZE); }

   // pre: key, ob diversi da null
   // post: aggiunge la coppia (key,ob) nel dizionario. Ritorna true
   //       se l'operazione e' andata a buon fine; false altrimenti
    public boolean insert(Comparable key, Object ob) {...} // COMPLETARE!
}
```

   Si osservi che la funzione hash è implementata dal metodo privato *hash*.

   N.B: è possibile definire eventuali metodi privati di supporto.

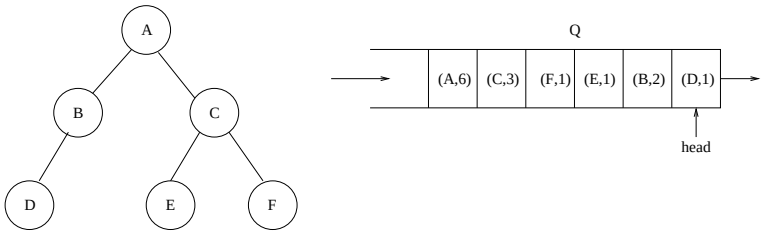
2. discutere gli eventuali problemi nella gestione delle collisioni che il metodo di scansione lineare può comportare.

Esercizio 5 (Laboratorio)

Si richiede di implementare *usando la ricorsione* il seguente metodo per la classe *BinaryTree* del package *BinTrees*, relativo agli alberi binari:

```
// post: ritorna una coda contenente, per ciascun nodo n dell'albero, una stringa che rappresenta
//       la coppia (key(n), num-nodi(n)) dove key(n) e' il valore della chiave di n e num-nodi(n)
//       e' il numero di nodi del sottoalbero radicato in n, compreso n stesso.
//       Se l'albero e' vuoto ritorna null.
public QueueCollegata sottoalberi() {...}
```

Si richiede che la coda restituita dal metodo *sottoalberi*, rappresenti una visita in post-ordine delle chiavi dell'albero. Esempio:



Nel foglio allegato sono descritte tutte le classi utili allo svolgimento dell'esercizio. In particolare, la classe *QueueCollegata* fa parte del package *Queues* visto a lezione.
N.B: è possibile utilizzare eventuali metodi privati di supporto.

Esercizio 6 (ASD)

1. Sia L una lista semplice (con attributi `key` e `next`) con sentinella (`sent`[L]) di numeri interi. Scrivere un algoritmo che trasforma L eliminando tutti gli elementi che contengono una chiave maggiore di quella dell'elemento immediatamente successivo nella lista di partenza. Esempi: la lista 7,5,6,3,2 si trasforma in 5,2 e la lista 3,4,6,1,5 si trasforma in 3,4,1,5.
2. Dimostrare la correttezza dell'algoritmo proposto.

```
***************************** classe BTNode.java ***************************
package BinTrees;
class BTNode {

    Object key;       // valore associato al nodo
    BTNode parent;    // padre del nodo
    BTNode left;      // figlio sinistro del nodo
    BTNode right;     // figlio destro del nodo


    // post: ritorna un albero di un solo nodo, con valore value e sottoalberi
    //       sinistro e destro vuoti
    BTNode(Object ob) {
        key = ob;
        parent = left = right = null;
    }


    // post: ritorna un albero contenente value e i sottoalberi specificati
    BTNode(Object ob,
           BTNode left,
           BTNode right,
           BTNode parent) {
        key = ob;
        this.parent = parent;
        setLeft(left);
        setRight(right);
    }
    ....
    ....
}
***************************** classe BinaryTree.java ***************************
package BinTrees;
import java.util.Iterator;
import Queues.*;
public class BinaryTree implements BT {
    private BTNode root;        // la radice dell'albero
    private BTNode cursor;      // puntatore al nodo corrente
    private int count;          // numero nodi dell'albero


    // post: crea un albero binario vuoto
    public BinaryTree() {
        root = null;
        cursor = null;
        count = 0;
    }
    ....
    ....
}
***************************** classe QueueCollegata.java ***************************
package Queues;
public class QueueCollegata implements Queue {
    private QueueRecord head;    // puntatore al primo elemento in coda
    private QueueRecord tail;    // puntatore all'ultimo elemento della coda
    private int count;           // numero di elementi in coda

    // post: costruisce una coda vuota
    public QueueCollegata() {
        head = null;
        tail = null;
        count = 0;
    }

    // post: ritorna il numero di elementi nella coda
    public int size() {...}

    // post: ritorna true sse la coda e' vuota
    public boolean isEmpty() {...}

    // post: svuota la coda
    public void clear() {...}

    // pre: coda non vuota
    // post: ritorna il valore del primo elemento della coda
    public Object front() {...}

    // pre: value non nullo
    // post: inserisce value in coda
    public void enqueue(Object ob) {...}

    // pre: coda non vuota
    // post: ritorna e rimuove l'elemento il primo elemento in coda
    public Object dequeue() {...}
}
```