Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 1 Settembre 2006 —

Esercizio 1 (ASD)

Considerata la ricorrenza:

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

si richiede di:

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

Esercizio 2 (ASD)

1. Sia A un array di $n > 10$ elementi che contiene un max-heap. Per ciascuna delle seguenti affermazioni dire se essa è necessariamente vera oppure no. Giustificare la risposta.
   - A[1] $\geq$ A[3].
   - A[2] $\geq$ A[6].
   - A[1] $\leq$ A[5].
   - A[5] $\leq$ A[2].
2. Sia T un BST (albero binario di ricerca) che contiene $n > 10$ chiavi. Si descriva un algoritmo efficiente (scrivere lo pseudo codice) per trasferire le chiavi memorizzate in T in un array A che rappresenta un max-heap.

Esercizio 3 (ASD)

Considerare la seguente procedura e determinare la sua complessità asintotica in funzione dell'input n.

```
proc(n) ::=
    m <- n*n
    while m > 0 do
        h <- 2*m + n
        for i = 1 to h do
            B[i] <- 0
        h <- 2*n
        for i = 1 to h do
            C[i] <- 0
        m <- m-1
```

Esercizio 4 (ASD)

Scrivere lo pseudocodice per una funzione che, dato un albero binario di ricerca T contenente chiavi distinte, determina se la radice di T contiene l'elemento *mediano superiore* dell'insieme delle chiavi contenute in T.
DEF: l'elemento mediano superiore di un insieme di n elementi è l'elemento che si trova in posizione $\lceil n/2 \rceil$ nella sequenza ordinata degli elementi dell'insieme. Esempio: il mediano dell'insieme $\{5, 1, 10, 2, 4\}$ è 4 mentre il mediano dell'insieme $\{9, -1, 5, 7, 8, 2\}$ è 5.

Esercizio 5 (Laboratorio)

Implementare il seguente metodo per la classe *BinaryTree* del package *BinTrees* visto a lezione:

```
    // post: ritorna una lista concatenata di tipo SLList contenente le chiavi di tutti nodi
    //       dell'albero che sono nonni di almeno un nipote.
    //       La lista e' vuota se non esiste alcun nodo nell'albero con questa proprieta'.
public SLList nonni() {...}
```

Se necessario, è possibile definire metodi privati di supporto.

Esercizio 6 (Laboratorio)

Si vuole realizzare l'implementazione di una *lista di liste* in cui i nodi della lista principale sono di tipo:

```
package Esercizio6;
class NodoLL {
    Object  key;       // chiave
    NodoL   headL;     // riferimento alla testa della lista associata al nodo
    NodoLL  next;      // riferimento al prossimo nodo

    NodoLL(Object ob) { key = ob; next = null; headL = null; }
}
```

mentre i nodi delle liste associate alla lista principale sono di tipo:

```
package Esercizio6;
class NodoL {
    Object key;    // chiave
    NodoL  next;   // riferimento al prossimo nodo

    NodoL(Object ob) { key = ob; next = null; }
}
```

Si richiede di:

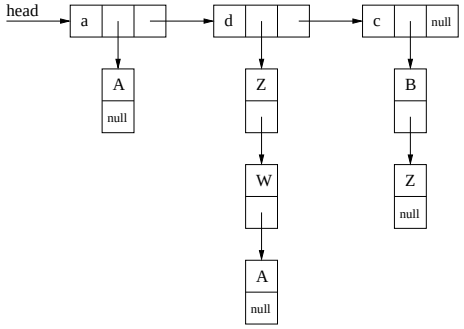
1. completare l'implementazione della seguente classe *ListaDiListe*:

   ```
   package Esercizio6;
   public class ListaDiListe {
       private NodoLL head = null;       // riferimento alla lista di liste

       // pre: ob1 e ob2 non nulli
       // post: ricerca l'oggetto ob1 nella lista principale e:
       //       - se ob1 e' presente inserisce ob2 in coda alla lista associata al nodo di ob1
       //       - se ob1 non e' presente inserisce un nuovo record con chiave ob1 in coda alla
       //         lista principale e un nuovo nodo con chiave ob2 nella lista associata al nodo di ob1
       public void insert(Object ob1, Object ob2) {...}
   }
   ```

2. [**Facoltativo**] scrivere gli invarianti del metodo *insert* relativi ai cicli di posizionamento nella lista principale e nella lista associata.

Un esempio di lista di liste costruita dalla classe *ListaDiListe* è il seguente:



```
****************************** interfaccia List.java ***************************
package BasicLists;
import java.util.Iterator;
public interface List {
    // post: ritorna il numero di elementi della lista
    public int size();

    // post: ritorna true sse la lista non ha elementi
    public boolean isEmpty();

    // post: svuota la lista
    public void clear();

    // pre:  ob non nullo
    // post: aggiunge l'oggetto ob in testa alla lista. Ritorna true se l'operazione e' riuscita, false altrimenti
    public boolean insert(Object ob);

    // pre: l'oggetto passato non e' nullo
    // post: ritorna true sse nella lista c'e' un elemento uguale a value
    public boolean contains(Object value);

    // pre: l'oggetto passato non e' nullo
    // post: rimuove l'elemento uguale a value; ritorna true se l'operazione e' riuscita, false altrimenti
    public boolean remove(Object value);

    // post: ritorna un oggetto che scorre gli elementi
    public Iterator iterator();

    // post: ritorna una lista che rappresenta tutti gli elementi della lista, in sequenza
    public String toString();
}
****************************** classe SLList.java ***************************
package BasicLists;
import java.util.Iterator;
public class SLList implements List {
    SLRecord head;              // primo elemento
    int count;                  // num. elementi nella lista

    // post: crea una lista vuota
    public SLList() { head = null; count = 0; }
    ....
}
****************************** classe SLRecord.java *********************
package BasicLists;
class SLRecord {
    Object key;             // valore memorizzato nell'elemento
    SLRecord next;         // riferimento al prossimo elemento

    // post: costruisce un nuovo elemento con valore v,  e prossimo elemento nextel
    SLRecord(Object ob, SLRecord nextel) { key = ob; next= nextel; }

    // post: costruisce un nuovo elemento con valore v, e niente next
    SLRecord(Object ob) { this(ob,null); }
}
****************************** 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; }
    ....
}
```