Algoritmi e Strutture Dati & Laboratorio di Algoritmi e Programmazione

– Appello del 12 Luglio 2005 –

Esercizio 1 (ASD)

1. Qual è il tempo di esecuzione di una operazione *insert* in un array ordinato di n elementi nel caso peggiore? Giustificare la risposta.
   (a) $O(\sqrt{n})$
   (b) $O(\log n)$
   (c) $O(n^2)$
   (d) $O(n)$.
2. Scrivere una ricorrenza che può essere risolta con il Master Theorem (caso 1) e trovarne la soluzione.

Esercizio 2 (ASD)

Qual è la complessità dell'algoritmo di cancellazione di una chiave in un albero R/N? Giustificare la risposta.

(a) O(n) nel caso peggiore

(b) O(log n) nel caso peggiore

(c) O(n log n) nel caso peggiore

(d) O(log n) nel caso medio ed O(n) nel caso peggiore

Esercizio 3 (ASD)

1. Disegnare un min-heap binario che contiene le chiavi: 28,23,15,7,24,2,9,18,37,3,10
2. Scrivere poi la sua rappresentazione lineare (array).

Esercizio 4 (ASD)

1. Scrivere un algoritmo (pseudo-codice) per trovare il *secondo minimo* in un array di $n \geq 2$ numeri interi non ordinati. (Il secondo minimo è l'elemento che precede il minimo nell'ordinamento descrescente).
2. Dimostrare la correttezza dell'algoritmo proposto utilizzando la tecnica dell'invariante.

Esercizio 5 (ASD e Laboratorio)

1. **(ASD)** Scrivere lo pseudo codice di un algoritmo che ritorna true se due alberi generali sono *strutturalmente uguali* (sovrapponibili, tranne al più il contenuto dei nodi).

   Per gli alberi generali si consideri la rappresentazione key, child, sibling per ciascun nodo.
2. **(LABORATORIO)** Si consideri il package *BinTrees* visto durante il corso e relativo agli alberi binari. Si vuole aggiungere alla classe *BinaryTree* il seguente metodo, che ritorna true se e solo se i sottoalberi sinistro e destro dell'albero sono uguali (sovrapponibili), sia strutturalmente che nel contenuto dei nodi.

```
// post: ritorna true se i sottoalberi sinistro e destro sono uguali
// sia strutturalmente che nel contenuto dei nodi; ritorna false altrimenti
public boolean uguali() {...}
```

   Si richiede di completare l'implementazione del metodo usando la ricorsione. Se necessario si utilizzi un metodo privato di supporto.

Esercizio 6 (Laboratorio)

Si vuole utilizzare un albero binario di ricerca per memorizzare le parole distinte di un testo. Ciascun nodo dell'albero è quindi relativo ad una parola (stringa) distinta e deve conteggiare il numero delle sue occorrenze nel testo in esame. Si richiede di:

1. Scrivere la classe *Nodo* che rappresenta un nodo dell'albero binario di ricerca;
2. Completare l'implementazione della seguente classe *ContaOccorrenze* scrivendo i metodi *insert* e *stampaOccorrenze*.

```
public class ContaOccorrenze {
    Nodo root;           // radice dell'albero
    int paroledistinte;  // conta le parole distinte del testo
    int paroletesto;     // conta tutte le parole del testo

    // NOTA: si assume che il testo in input sia memorizzato in un array di stringhe
    public ContaOccorrenze(String[] testo) {
        root = null;
        paroledistinte = 0;
        paroletesto = 0;
        if (testo != null)
           memorizzaTesto(testo);
    }

    // post: inserisce il testo nell'albero binario di ricerca
    private void memorizzaTesto(String[] testo) {
        for (int i=0; i < testo.length; i++) {
           insert(testo[i].toUpperCase());
           paroletesto++;
        }
    }

    // post: se parola non esiste nell'albero la inserisce come nuovo nodo;
    //       altrimenti aggiorna il numero di occorrenze del nodo corrispondente.
    //       Aggiorna il numero di parole distinte del testo.
    public void insert(String parola) {...}

    // post: stampa le parole dell'albero e le loro rispettive occorrenze
    //       in ordine crescente di parola
    public void stampaOccorrenze( ) {...}
}
```

Se necessario, definire eventuali metodi privati di supporto.

```
********************** CLASSE BTNode      ********************************************************
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  ********************************************************
package BinTrees;
import Utility.*;
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;
    }
    ...
}
```