

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 *delete* 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 3) e trovarne la soluzione.

Esercizio 2 (ASD)

Qual è la complessità dell'algoritmo di inserimento 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: 6,23,13,11,16,2,4,17,35,3,7
2. Scrivere poi la sua rappresentazione lineare (array).

Esercizio 4 (ASD)

1. Scrivere un algoritmo (pseudo-codice) per trovare il *secondo massimo* in un array di $n \geq 2$ numeri interi non ordinati. (Il secondo massimo è l'elemento che segue il massimo nell'ordinamento decrescente).
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 uguali (sovrapponibili), sia strutturalmente che nel 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 *strutturalmente uguali*, (sovrapponibili, tranne al più il contenuto dei nodi).

```
// post: ritorna true se i sottoalberi sinistro e destro sono strutturalmente uguali,  
// (cioè' sovrapponibili, tranne al più il contenuto dei nodi)  
public boolean structEquals() {...}
```

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 i voti distinti di un appello d'esame per successive analisi statistiche sulla distribuzione dei voti. Ciascun nodo dell'albero è quindi relativo ad un voto distinto e deve conteggiare il numero delle sue occorrenze nell'appello d'esame che si sta considerando.

Si richiede di:

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

```
public class ContaVoti {
    Voto root;           // radice dell'albero
    int votidistinti;     // conta i voti distinti dell'appello
    int votiappello;      // conta tutti i voti dell'appello

    // NOTA si assume che l'appello d'esame in input sia
    // memorizzato in un array di interi
    public ContaVoti(int[] appello) {
        root = null;
        votidistinti = 0;
        votiappello = 0;
        if (appello != null)
            memorizzaAppello(appello);
    }

    // post: inserisce l'appello nell'albero binario di ricerca
    private void memorizzaAppello(int[] appello) {
        for (int i=0; i < appello.length; i++) {
            insert(appello[i]);
            votiappello++;
        }
    }

    // post: se il voto non esiste nell'albero lo inserisce come nuovo nodo;
    //        altrimenti aggiorna il numero di occorrenze del nodo corrispondente
    public void insert(int voto) {...}

    // post: stampa i voti dell'albero e le loro rispettive occorrenze
    //        in ordine crescente di voto
    public void stampaVoti( ) {...}
}
```

Se necessario, definire eventuali metodi privati di supporto.

```

***** CLASSE BTreeNode *****
package BinTrees;
class BTreeNode {

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

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

    // post: ritorna un albero contenente value e i sottoalberi specificati
    BTreeNode(Object ob,
               BTreeNode left,
               BTreeNode right,
               BTreeNode parent) {
        key = ob;
        this.parent = parent;
        setLeft(left);
        setRight(right);
    }
}

***** CLASSE BinaryTree *****
package BinTrees;
import Utility.*;
public class BinaryTree implements BT {
    private BTreeNode root;        // la radice dell'albero
    private BTreeNode 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;
    }
    ...
}

```