Algoritmi e Strutture Dati & Laboratorio di Algoritmi e Programmazione

– Appello del 1 Settembre 2005 –

Esercizio 1 (ASD)

Sia $T(n) = 4T(\frac{n}{2}) + O(n^3)$. Dire, quale delle seguenti risposte è quella esatta. Giustificare la risposta.

(a) $T(n) = \Theta(n^2 \lg n)$

(b) $T(n) = \Theta(n^3)$

(c) $T(n) = \Theta(n^2)$

(d) Nessuna delle precedenti risposte è esatta.

Esercizio 2 (ASD)

Qual è la complessità dell'algoritmo di heapsort? Giustificare la risposta.

(a) O(n)

(b) O(log n)

(c) O(n log n)

(d) $\Theta(n^2)$

Esercizio 3 (ASD)

Disegnare un albero R/B che contiene le chiavi: 1,2,3,4,5,6,7,8,9,10,11,12 ed ha altezza massimale rispetto agli altri alberi R/B che contengono le stesse chiavi.

Esercizio 4 (Laboratorio)

Si vuole realizzare una struttura dati a lista concatenata che consenta di gestire un multi-insieme di numeri interi ordinati in senso crescente. Si implementi quindi:

1. una classe *Elemento* che memorizza un singolo numero intero del multi-insieme e tiene conto del numero delle sue occorrenze;
2. una classe *MultiInsieme* che gestisce il multi-insieme. La classe deve prevedere
   - un metodo costruttore;
   - un metodo *insert* per l'inserimento ordinato di un nuovo elemento nel multi-insieme;
   - un metodo *remove* per la cancellazione di un elemento dal multi-insieme. Il metodo deve cancellare una singola occorrenza dell'elemento e deve ritornare un valore booleano che indichi se l'operazione è andata a buon fine (elemento trovato e cancellato) oppure no (elemento non presente nel multi-insieme).

Esercizio 5 (ASD)

Scrivere lo pseudo codice di un algoritmo che verifica se un albero binario T è un albero binario di ricerca. Si assuma che i nodi di T siano stati aumentati aggiungendo agli attributi *key*, *left* e *right* anche gli attributi *min* e *max* che individuano rispettivamente la chiave minima e la chiave massima tra tutte le chiavi memorizzate nel sottoalbero radicato in quel nodo.

Esercizio 6 (ASD e Laboratorio)

1. **(ASD)** Scrivere lo pseudo codice di un algoritmo che trasforma un albero binario rappresentato come un albero generale, ovvero tramite gli attributi *key*, *child*, *sibling*, in un un albero binario rappresentato utilizzando gli attributi: *key*, *left*, *right*.

   Si utilizzi la funzione get_bnode per creare un nuovo nodo con gli attributi *key*, *left*, *right*.

2. **(LABORATORIO)** Si consideri il package *Trees* sviluppato durante il corso e relativo agli alberi generali. Si vuole aggiungere alla classe *GenTree* il seguente metodo che verifica se l'albero generale è un albero binario.

   ```
   // post: ritorna true se l'albero e' binario; ritorna false altrimenti
   public boolean isBinary() {...}
   ```

   Si richiede di implementare il metodo *isBinary* usando la ricorsione. Il metodo ritorna true se l'istanza corrente dell'albero è un albero binario, cioé se ogni suo nodo ha al più due figli. In caso contrario il metodo ritorna false. Se necessario utilizzare un metodo privato di supporto.

```
********************** CLASSE TreeNode  ********************************************************
package Trees;
class TreeNode {
    Object key;       // valore associato al nodo
    TreeNode parent;    // padre del nodo
    TreeNode child;      // figlio sinistro del nodo
    TreeNode sibling;     // fratello destro del nodo

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

    // post: ritorna un albero contenente value e i sottoalberi specificati
    TreeNode(Object ob, TreeNode parent, TreeNode child, TreeNode sibling) {
        key = ob;
        this.parent = parent;
        this.child = child;
        this.sibling = sibling;
    }
}
********************** CLASSE GenTree  *********************************************************
package Trees;
import Utility.*;
public class GenTree implements Tree{
    private TreeNode root;      // radice dell'albero
    private int count;           // numero di nodi dell'albero
    private TreeNode cursor;    // riferimento al nodo corrente

    // post: costruisce un albero vuoto
    public GenTree() {
        root = cursor = null;
        count = 0;
    }
    ...
    ...
 }
```