Algoritmi e Strutture Dati
&
Laboratorio di Algoritmi e Programmazione

— Appello del 14 Gennaio 2009 —

Esercizio 1 - ASD

Provare la verità o la falsità di ciascuna delle seguenti affermazioni.

- La soluzione della ricorrenza $T(n) = 5T(\frac{n}{3}) + 3n^2 + 2\sqrt{n^3}$ è nella classe $\Theta(n^2)$
- $\Omega(n \lg n) + n^2 = O(n^2)$
- La soluzione della ricorrenza $T(n) = 2T(\sqrt{n}) + 3n^2$ è nella classe $\Theta(n^2)$

Esercizio 2 - ASD

Discutere la correttezza di ciascuna delle seguenti affermazioni. Dimostrare formalmente la validità delle risposte date.

1. La complessità asintotica dell'algoritmo di ricerca di una chiave in un albero R/B con n nodi è $\Theta(\lg n)$.
2. La complessità asintotica dell'algoritmo di ricerca di una chiave in un albero BST con n nodi è $\Omega(\lg n)$.
3. La complessità asintotica dell'algoritmo di ricerca di una chiave in un min-heap binario con n nodi è $\Theta(n)$.
4. Esistono alberi R/N che hanno tutti i nodi neri.
5. Possiamo trasformare un albero R/B con n nodi in un min-heap con complessità asintotica $\Theta(n)$.

Esercizio 3 - ASD

Diciamo che T è un *interval* BST se è un BST a chiavi intere che soddisfa la seguente proprietà: per ogni intero k, se le chiavi k e $k+2$ sono in T allora anche la chiave $k+1$ è in T.

Proporre un algoritmo che verifica se un dato BST a chiavi intere è un interval BST.

Dire qual è la complessità dell'algoritmo e spiegare perché è corretto.

Esercizio 4 - ASD

Si realizzi una operazione RIMUOVI(A, k, P) che soddisfa la seguente specifica:

Precondizione: A è un array che rappresenta un max-heap di dimensione `heapsize`$[A]$ a chiavi intere; $0 < k \leq 100$ e $P > 0$ sono due costanti.

Postcondizione: L'algoritmo restituisce un nuovo array A che rappresenta il max-heap ottenuto eliminando da quello iniziale i k più grandi elementi di A maggiori di P oppure tutti gli elementi di A maggiori di P nel caso vi siano in A meno di k elementi maggiori di P.

Dire qual è la complessità dell'algoritmo rispetto alla dimensione iniziale di A e spiegare perché è corretto.

Esercizio 1 (Laboratorio)

Dato il package *Liste* realizzato durante il corso, aggiungere il seguente metodo alla classe *ListaDoppia* che realizza le operazioni sulle liste mediante una lista doppia circolare con sentinella:

```
// post: ritorna il numero di elementi distinti presenti in lista
public int  distinct() {...}
```

Ad esempio, se la lista contiene, nell'ordine, gli elementi *x, g, h, g, v, x, y, g* allora il metodo deve ritornare il valore 5. Si richiede di completare l'implementazione del metodo e di scrivere gli invarianti di ciclo (senza dimostrarli!).
NOTA: per la soluzione di questo esercizio non possono essere utilizzate strutture dati d'appoggio e non possono essere richiamati gli altri metodi della classe *ListaDoppia*.

Esercizio 2 (Laboratorio)

Dato il package *Tree* realizzato durante il corso e relativo agli alberi generali, aggiungere il seguente metodo alla classe *GenTree*:

```
// pre: k diverso da null
// post: ritorna true sse tutte le foglie dell'albero hanno chiave uguale a k
public boolean foglieUguali(Object k) {...}
```

Si richiede di completare l'implementazione del metodo usando la ricorsione. L'algoritmo deve avere complessità lineare rispetto al numero di nodi dell'albero. È possibile utilizzare un metodo privati di appoggio.

```
********************** classe RecordLD ************************************
package Liste;
class RecordLD {
    Object key;             // valore memorizzato nell'elemento
    RecordLD next;          // riferimento all'elemento successivo
    RecordLD prev;          // riferimento all'elemento precedente

    // post: costruisce un nuovo elemento con valore v,
    //       elemento successivo nextel e precedente prevel
    RecordLD(Object ob, RecordLD nextel, RecordLD prevel) {
        key = ob;
        next= nextel;
        if (next != null)
            next.prev = this;
        prev = prevel;
        if (prev != null)
            prev.next = this;
    }

    // post: costruisce un nuovo elemento con valore v, e niente next e prev
    RecordLD(Object ob) {
        this(ob,null, null);
    }
}
********************** classe ListaDoppia **********************************
package Liste;
public class ListaDoppia implements Lista {
    private RecordLD sentinel;          // riferimento alla sentinella
    private int count;                  // num. elementi nella lista


    // metodo costruttore
    // post: crea una lista vuota
    public ListaDoppia() {
        // la sentinella ha chiave puntatori nulli
        sentinel = new RecordLD(null,null, null);
        sentinel.next = sentinel;
        sentinel.prev = sentinel;
        count = 0;
    }

   ...
}
********************** classe TreeNode *************************************
package Trees;
class TreeNode {

    Object key;                 // valore associato al nodo
    TreeNode parent;     // padre del nodo
    TreeNode child;        // primo figlio del nodo
    TreeNode sibling;     // fratello destro del nodo


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


    // post: ritorna un albero contenente la chiave ob 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;
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;
    }
   ...
}
```