

Algoritmi e Strutture Dati

&

Laboratorio di Algoritmi e Programmazione

— Appello del 26 Giugno 2006 —

Esercizio 1 (ASD)

Considerata la ricorrenza:

$$T(n) = 2T\left(\frac{n}{3}\right) + n \lg n$$

si richiede di:

- risolverla utilizzando il teorema principale;
- dire se $T(n) = O(n^2)$, giustificando la risposta.

Esercizio 2 (ASD)

Sia T un albero R/B. Per ciascuna delle seguenti affermazioni dire se essa è vera o falsa. Giustificare la risposta.

- Per ogni nodo x di T , su ogni cammino da x ad una foglia esiste almeno un nodo nero.
- Per ogni nodo x di T , su ogni cammino da x alla radice esiste almeno un nodo rosso.
- Per ogni nodo x di T , su ogni cammino da x ad una foglia esiste lo stesso numero di nodi neri.
- Per ogni nodo x di T , su ogni cammino da x ad una foglia il numero dei nodi neri è almeno il doppio di quelli rossi.
- Per ogni nodo x di T , su ogni cammino da x ad una foglia il numero dei nodi neri è al più il doppio di quelli rossi.

Esercizio 3 (ASD)

Si consideri la struttura dati max-heap e si sviluppi un algoritmo efficiente (scrivere lo pseudo codice) che dato un max-heap memorizzato in un array A ed un indice i , $1 \leq i \leq \text{heap_size}[A]$, incrementa $A[i]$ di una quantità positiva k e restituisce un nuovo max-heap, ancora memorizzato in A .

Discutere la complessità dell'algoritmo proposto.

Esercizio 4 (ASD + Laboratorio)

Si consideri il package *Dizionario* visto a lezione e, in particolare, la classe *DizBST* che implementa il tipo di dato Dizionario mediante un albero binario di ricerca.

1. **(Laboratorio)** Si vuole aggiungere alla classe *DizBST* il seguente metodo, che verifica se l'albero che rappresenta il dizionario è un albero AVL:

```
// post: ritorna true sse l'albero che rappresenta il dizionario e' un albero AVL
public boolean isAVL() {...}
```

Si richiede di implementare il metodo *isAVL* usando la ricorsione. La complessità del metodo deve essere $O(n)$, dove n è il numero degli elementi presenti nel dizionario.

Se necessario, è possibile definire un eventuale metodo privato di supporto.

NOTA: si ricorda che un albero AVL gode della seguente proprietà di bilanciamento: *Per ogni nodo x , le altezze dei sottoalberi sinistro e destro di x differiscono di al più una unità.*

2. (ASD) Scrivere l'algoritmo al punto precedente in pseudo-codice. Dimostrare la correttezza dell'algoritmo ricorsivo proposto.

Esercizio 5 (Laboratorio)

Si vuole realizzare un'implementazione del tipo di dato Dizionario mediante una tabella hash, in cui le collisioni sono risolte mediante liste (semplici) di collisioni. Data la classe:

```
package Esercizio5;
class Nodo {
    Comparable key;          // chiave
    Object elem;             // elemento
    Nodo next;               // riferimento al prossimo elemento

    // post: costruisce un nuovo elemento con chiave key e valore ob
    Nodo(Comparable k, Object ob, Nodo nextel) {key = k; elem = ob; next = nextel;}

    // post: costruisce un nuovo elemento con chiave key e valore ob
    Nodo(Comparable k, Object ob) {
        this(k,ob,null);
    }
}
```

che rappresenta una coppia (chiave, elemento) da memorizzare nel dizionario e il riferimento al prossimo elemento della lista di collisione, si richiede di:

1. Completare l'implementazione della seguente classe *DizHashCollisioni*, ipotizzando che il dizionario non ammetta chiavi duplicate e che le liste di collisione siano ordinate rispetto alla chiave:

```
package Esercizio5;
public class DizHashCollisioni {
    private static final int DEFSIZE = 113;    // capacita' array
    private Nodo[] diz;                        // tabella hash
    private int count;                          // totale coppie nel dizionario

    //post: costruisce un dizionario vuoto
    public DizHashCollisioni() { diz = new Nodo[DEFSIZE]; count = 0; }

    // pre: key diverso da null
    // post: ritorna l'indice dell'array associato a key
    private int hash(Comparable key) { return (key.hashCode() % DEFSIZE); }

    // pre: key, ob diversi da null
    // post: aggiunge la coppia (key,ob) nel dizionario. Ritorna true
    //       se l'operazione e' andata a buon fine; false altrimenti
    public boolean insert(Comparable key, Object ob) {...} // COMPLETARE!
}
```

2. Definire la nozione di *fattore di carico* di una tabella hash. Nel caso in cui le collisioni siano gestite mediante liste di collisioni, cosa succede al fattore di carico della tabella hash?

Esercizio 6 (ASD)

1. Si elenchino tutte le operazioni definite sulla struttura dati albero binario di ricerca (BST), descrivendone le funzionalità.
2. Siano A un albero binario di ricerca contenente n interi e B un array ordinato (\leq) contenente m interi.
 - (a) Scrivere lo pseudo codice di una procedura efficiente per stampare in ordine non decrescente (\leq) l'unione (compresi eventuali duplicati) di tutti gli elementi di A e di B.
 - (b) Indicare la complessità della procedura in funzione di n ed m .

```

***** classe DizBSTNode.java *****
package Dizionario;
class DizBSTNode {

    Comparable key;        // chiave associato al nodo
    Object elem;           // elemento associato alla chiave
    DizBSTNode parent;     // padre del nodo
    DizBSTNode left;       // figlio sinistro del nodo
    DizBSTNode right;      // figlio destro del nodo

    // post: ritorna un albero di un solo nodo, con chiave key, elemento ob
    //       e sottoalberi sinistro e destro vuoti
    DizBSTNode(Comparable key, Object ob) {
        this.key = key;
        elem = ob;
        parent = left = right = null;
    }
}

***** classe DizBST.java *****
package Dizionario;
public class DizBST implements Dizionario {
    private DizBSTNode root;        // radice dell'albero che rappresenta il dizionario
    private int count;              // numero di nodi dell'albero

    // post: costruisce un albero di ricerca vuoto
    public DizBST() {
        root = null;
        count = 0;
    }
    ....
    ....
}

***** interfaccia Dizionario.java *****
package Dizionario;
public interface Dizionario {

    // post: ritorna il numero di elementi nel dizionario
    public int size();

    // post: ritorna true sse il dizionario e' vuoto
    public boolean isEmpty();

    // post: svuota il dizionario
    public void clear();

    // pre: key, ob diversi da null
    // post: aggiunge la coppia (key,ob) nel dizionario. Ritorna true
    //       se l'operazione e' andata a buon fine; false altrimenti
    public boolean insert(Comparable key, Object ob);

    // pre: key diverso da null
    // post: cancella dal dizionario la coppia con chiave key. Ritorna
    //       true se l'operazione e' andata a buon fine; false altrimenti
    public boolean delete(Comparable key);

    // pre: key diverso da null
    // post: se key e' presente nel dizionario ritorna l'elemento ad essa
    //       associato. Ritorna null altrimenti.
    public Object search(Comparable key);
}

```