Laboratorio di Algoritmi e Programmazione

— Appello del 18 Gennaio 2011 —

Esercizio 1

Data la classe

```
package Esercizio1;
class NodoLista {
    String key;             // valore memorizzato nell'elemento
    int occorrenze;         // numero occorrenze di key
    NodoLista next;         // riferimento al prossimo elemento

    // post: costruisce un nuovo elemento con valore v,
    //       e prossimo elemento nextel
    NodoLista(String key, NodoLista nextel) {
        this.key = key;
        occorrenze = 1;
        next= nextel;
    }

    // post: costruisce un nuovo elemento con valore key, e next a null
    NodoLista(String key) { this(key,null);}
}
```

si vuole realizzare una lista concatenata con nodi di tipo *NodoLista*, in cui gli elementi compaiono ordinati rispetto al campo *key*. Il numero delle occorrenze di ciascuna *chiave* presente in lista è memorizzato nel campo *occorrenze*. Si richiede di implementare il metodo *insert* della seguente classe *ListaOrdinataConOccorrenze*:

```
package Esercizio1;
public class ListaOrdinataConOccorrenze {
    private NodoLista head;   // testa della lista

    public ListaOrdinataConOccorrenze() { head = null; }

    // pre: chiave non nulla
    // post: inserisce elem in lista mantenendo l'ordinamento crescente.
    //       Se elem non e' presente crea un nuovo record; se elem e' gia'
    //       presente incrementa il numero delle sue occorrenze.
    public void insert(String elem) {...}
}
```

Esercizio 2

Dato il package *BinTrees* sviluppato durante il corso e relativo agli alberi binari, si vuole aggiungere alla classe *BinaryTree.java* il seguente metodo:

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

Si richiede di implementarlo utilizzando un metodo privato di supporto ricorsivo. L'algoritmo deve avere complessità lineare rispetto al numero di nodi presenti nell'albero.
Si ricorda che un albero binario è bilanciato se, per ogni suo nodo n, le altezze dei sottoalberi sinistro e destro di n differiscono al piu' di uno.

```
********************** 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 Queues.*;
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;
    }

    ...
}
```