Universidad Carlos III de Madrid

Programación de Sistemas
Grado en Ingeniería Telemática

Leganés, 24 de abril de 2013 — Examen parcial 2 (problemas)
Duración de la prueba: 45 min — Puntuación: 5 puntos sobre 10 del examen

Problema 1 (1,5 puntos)

Implementa de forma recursiva el método *mcd*, el cual permite calcular el máximo común divisor (*MCD*) de los dos números enteros positivos que recibe como argumento. Como parte de este método deberás comprobar que los dos números enteros recibidos son positivos y, en caso contrario, devolver *-1*. **PISTA**: Puedes tener en cuenta que el *MCD(a,b)* es igual al *MCD(a-b,b)* cuando *a* es mayor que *b*. **IMPORTANTE**: No se considerará válida una solución que no sea recursiva.

Problema 2 (2 puntos)

En este problema vamos a trabajar con una *lista enlazada* como estructura de almacenamiento de datos. Esta lista hace uso de la clase *Node*, la cual ya está programada con los atributos y métodos habituales que hemos visto en clase, y por tanto no es necesario que la programes. La clase *List* implementa la *lista enlazada* siguiendo la estructura:

```
public class List {
    private Node first;
    (...)
    public int contar (Object info) {...}
}
```

Programa el método *public int contar(Object info)* de la clase *List* para que devuelva como resultado el número de elementos de la lista cuya información almacenada sea igual a la del objeto que se recibe como argumento. Usa el método *equals(Object o)* de la clase *Object* para saber si dos objetos son iguales.

(continúa en el reverso de la hoja)

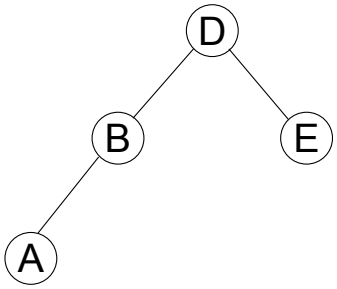
Problema 3 (1,5 puntos)

En este problema vamos a trabajar con un ejemplo de estructura de árbol binario. Para ello, partimos de las clases *BNode* y *BTree*, las cuales ya están programadas, y por tanto no es necesario que programes. Estas clases tienen, entre otros, los siguientes métodos:

```
public class BTree{
    (...)
    public BTree() {...}                    // crea árbol vacío
    public BTree(Object info) {...}         // crea árbol con sólo un nodo y sin hijos
    public void setRoot(BNode root) {...}
    public BNode getRoot() {...}

    // Si te basas en la implementación en que un nodo contiene árboles:
    public void setLeft(BTree left) {...}              // inserta a la izquierda
    public void setRight(BTree right) {...}            // inserta a la derecha
    public void insert(Btree subtree, int side) {...}  // inserta en el lado que se indique
}

public class BNode{
    public static final int LEFT_SIDE = 1;
    public static final int RIGHT_SIDE = 2;
    (...)
    public BNode(Object info) {...}      // crea un nodo con un dato y sin hijos

    // Si te basas en la implementación en que un nodo contiene árboles:
    public void setLeft(BTree left) {...}              // inserta a la izquierda
    public void setRight(BTree right) {...}            // inserta a la derecha
    public void insert(Btree subtree, int side) {...}  // inserta en el lado que se indique

    // Si te basas en la implementación en que un nodo contiene nodos:
    public void setLeft(BNode left) {...}              // inserta a la izquierda
    public void setRight(BNode right) {...}            // inserta a la derecha
    public void insert(BNode subtree, int side) {...}  // inserta en el lado que se indique
}
```

Apartado 1 (1 punto)

Programa un método *public void createTree()* en la clase de prueba *BTreeTest* que permita crear la estructura de árbol que se presenta a continuación. Puedes utilizar cualquiera de las posibles implementaciones de árboles vistas en clase, cuyos métodos se muestran arriba.



Apartado 2 (0,5 puntos)

Si el árbol se recorre siguiendo el orden alfabético (empezando por la "A"), ¿cuál de los tres tipos de recorrido vistos en clase se está utilizando?

Problema 1

```
public int mcd(int a, int b){
    if (a < 0 || b < 0) {
        return -1;
    } else if (b == 0) {
        return a;
    } else if (a == 0) {
        return b;
    } else if (a >= b) {
        return mcd(a - b, b);
    } else {
       return mcd (a, b - a);
    }
}
```

Problema 2

Apartado 1

```
public int contar(Object info) {
    int contador = 0;
    Node n = first;
    while (n != null) {
        if (n.getInfo().equals(info)) {
            contador++;
        }
        n = n.getNext();
    }
    return contador;
}
```

Problema 3

Apartado 1

Solución basada en que cada nodo recibe a izquierda y derecha un nodo. Los alumnos pueden implementar la opción alternativa explicada en clase según la cual cada nodo debe recibir a izquierda y derecha un nuevo árbol (aunque esté vacío).

```
public class BTreeTest {
    public void createTree() {
        BTree tree = new BTree();
        BNode root = new BNode("D");
        BNode nodeB = new BNode("B");
        nodeB.setLeft(new BNode("A"));
        root.setLeft(nodeB);
        root.setRigth(new BNode("E"));
        tree.setRoot(root);
    }
}
```

La solución alternativa, cuando los hijos de un nodo son árboles:

```
public class BTreeTest {
    public void createTree() {
        BTree root = new BTree("D");
        BTree treeB = new BTree("B");
        treeB.setLeft(new BTree("A"));
```

```
        root.setLeft(treeB);
        root.setRight(new BTree("E"));
    }
}
```

Apartado 2

Inorden