Grado en Ingeniería Informática	**Asignatura Estructura de Datos y Algoritmos** **27 Mayo 2015** **Convocatoria Ordinaria (Opción Evaluación continua)**

Nombre: .....
Apellidos: .....
Grupo: .....

LEA ATENTAMENTE ESTAS INSTRUCCIONES ANTES DE COMENZAR LA PRUEBA

1. Es necesario poner todos los datos del alumno en el cuadernillo de preguntas (este documento) y en las hojas de cuadros. Use un bolígrafo para rellenarlos. No se corregirá nada que esté escrito a lapicero.
2. El examen está compuesto por 3 problemas. La nota final se calcula sobre 6 puntos. Es necesario obtener al menos un 40% de la calificación máxima (es decir, al menos un 2,4) para poder optar a la evaluación continua (la nota final del curso es la suma de todas las pruebas).
3. Aquellos alumnos que no quieran seguir la opción de evaluación continua, tienen derecho a realizar un examen final (opción ordinaria + ejercicios extras). Los ejercicios extras se entregarán al finalizar esta primera parte.
4. Cuando finalice la prueba, se deben entregar el enunciado del examen y cualquier hoja que haya empleado.
5. No está permitido salir del aula por ningún motivo hasta la finalización del examen. Desconecten los móviles durante el examen.
6. **En este examen, no está permitido usar las clases de la librería EdaLib y tampoco las interfaces y clases que proporciona el api de java (ArrayList, LinkedList, etc). Tampoco está permitido utilizar Arrays. Es decir, en tus soluciones debes implementar tus propias estructuras de datos.**

NO PASE DE ESTA HOJA hasta que se le indique el comienzo del examen

Problema 1 (2 puntos): Dadas las clases

```
public class NodoSimpleEntero {

    public Integer elem;
    public NodoSimpleEntero next;

    public NodoSimpleEntero(Integer num) {
        elem = num;
    }
}
```

```
public class ListaSimpleEntero {

    public NodoSimpleEntero primero;

}
```

```
public class NodoDobleEntero {

    public Integer elem;
    public NodoDobleEntero sig;
    public NodoDobleEntero prev;

    public NodoDobleEntero(Integer entero) {
        this.elem = entero;
    }

}
```

```
public class ListaDobleEntero {

    public NodoDobleEntero header;
    public NodoDobleEntero trailer;

    public ListaDobleEntero() {
        header = new NodoDobleEntero(null);
        trailer = new NodoDobleEntero(null);
        header.sig = trailer;
        trailer.prev = header;
    }

}
```

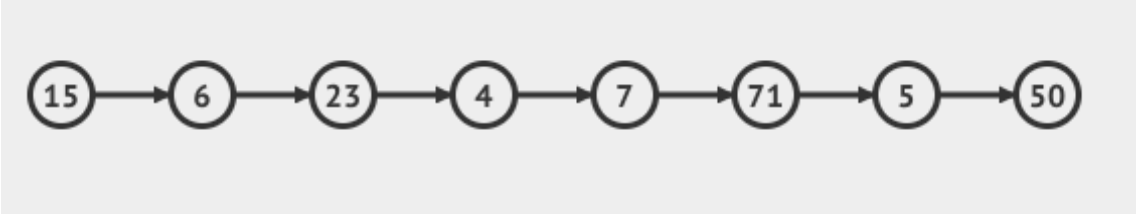
Implementa un método **buscarycopiar** que reciba una lista simple, una lista doble y un número entero:

```
boolean buscarycopiar(ListaSimpleEntero slist, ListaDobleEntero dlist, Integer elem) {
```

El método **buscarycopiar** debe buscar el valor **elem** dentro de la lista simple **slist**, y si existe se insertará en la lista doble **dlist** en la misma posición en la que se encuentra en la lista simple **slist**. Si la posición del número entero en la lista simple es mayor o igual que el tamaño de la lista doble, el número simplemente se insertará al final de la lista doble. El método debe devolver true si el número es encontrado en la lista simple y false en otro caso.

Ejemplo si se pide buscar el valor 7:

Lista Simple:



El elemento se encuentra en la posición 4, por tanto, debe ser almacenado en la posición 4 de la lista doble. Recuerda que las posiciones en los TADs se empiezan a numerar desde 0.

Lista Doble:

Notas:

- En la lista simple no existen duplicados.
- En la lista doble no es necesario controlar si ya existe el valor a insertar (se permiten duplicados).
- Las listas no están ordenadas.

SOLUCIÓN:

```
boolean searchAndCopy(SimpleListInt slist, DoubleListInt dlist, Integer elem) {
    int pos=slist.searchPosition(elem);
    if (pos==-1) {
        System.out.println(elem + " was not found!!!.");
        return false;
    } else {
        System.out.println("Position of "+ elem + ": " +pos);
    }
    dlist.insertAt(elem, pos);
    return true;
}
```

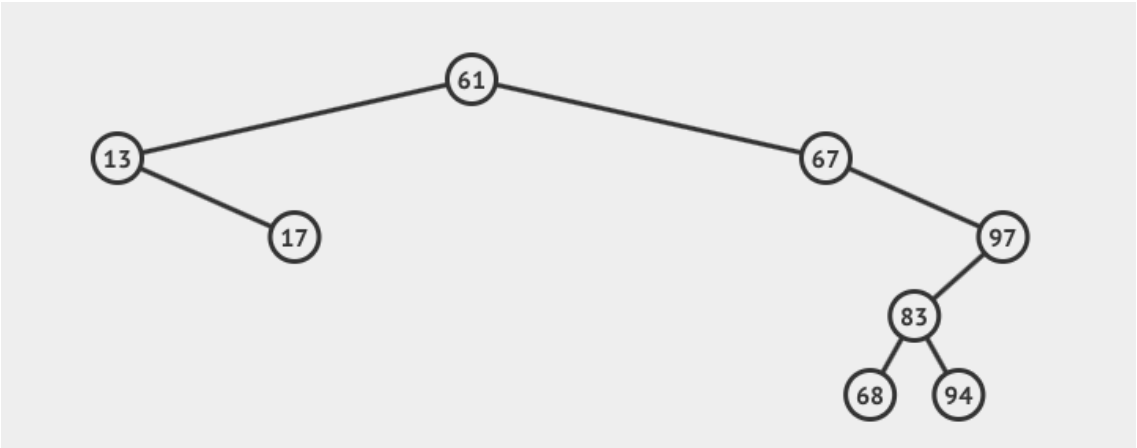
El método searchPosition debe ser un método de SimpleListInt:

```
/**
 * search the position of elem in the list.
 * If it is not found return -1
 * @param elem
 * @return
 */
public int searchPosition(int elem) {

    int pos=0;
    boolean found=false;
    SimpleNodeInt snodeIt = this.first;

    while (snodeIt != null && !found) {
        if (snodeIt.elem == elem) {
            found=true;
        }
        snodeIt = snodeIt.next;
        pos++;
    }

    if (!found) {
        pos=-1;
    }
    return pos;
}
```

El método inserAt debe ser un método de DoubleListInt

```
public void insertAt(int elem,int pos) {

    DoubleNodeInt nodeIt = this.header.next;
    int i=0;

    while (nodeIt != this.trailer && i<pos) {
        nodeIt = nodeIt.next;
        i++;
    }

    if (nodeIt==this.trailer) {
        System.out.println("position " + pos + " is greater "
                + "than the size of list (" + i +"). "
                + "The elem has to be added at the end of "
                + "the list.");
    }

    DoubleNodeInt newN = new DoubleNodeInt(elem);
    newN.next = nodeIt;
    newN.prev = nodeIt.prev;
    nodeIt.prev.next = newN;
    nodeIt.prev = newN;

}
```

Problema 2: Dado el siguiente árbol binario de búsqueda **(Total 0,5 punto)**:



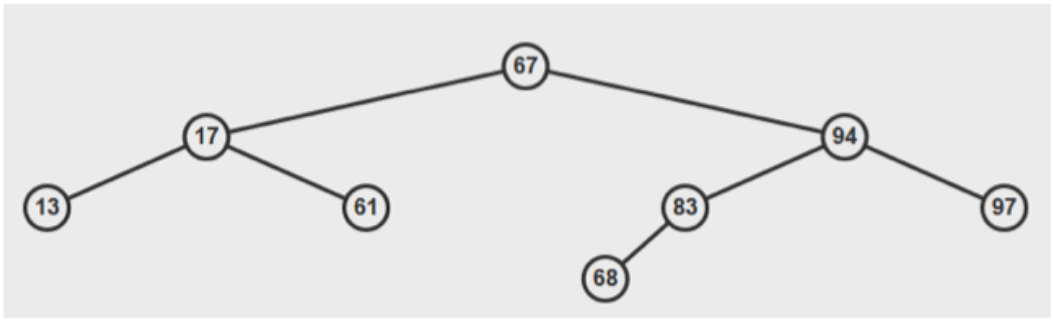
Se pide:

a) Versión perfectamente equilibrada del árbol (equilibrio en tamaño). (0.20)
b) Versión AVL (equilibrio en altura). (0.20)
c) ¿Cuál es el objetivo de equilibrar un árbol binario de búsqueda? (0.1)

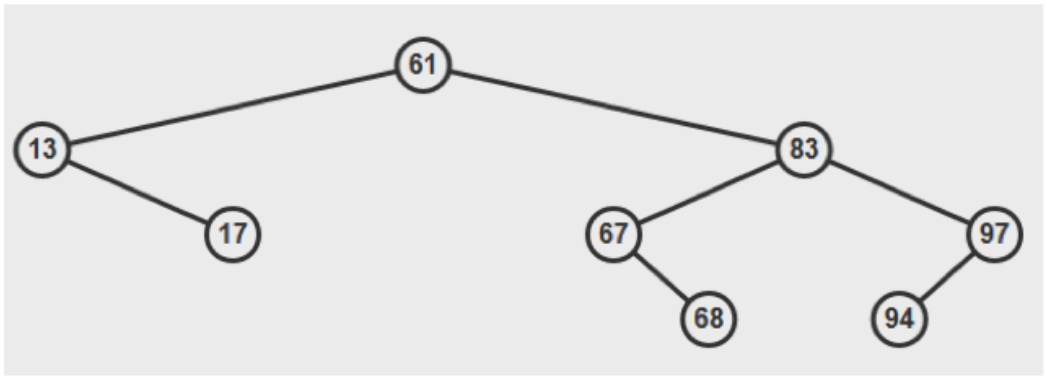
Nota: La solución debe incluir cada uno de los pasos que se han realizado hasta llegar al resultado final.

Solution:

a)



AVL



Problema 3 (**Total 3.5 puntos**)**:** Cada hora la torre de control del aeropuerto NiceFlights recibe las peticiones de los vuelos que necesiten usar su pista de vuelo (el aeropuerto dispone de una única pista). De cada vuelo, se necesita saber el número de vuelo (por ejemplo, IB1234) y el minuto en el que solicita utilizar la pista de vuelo (por ejemplo, 35).

Se pide:

a) (**0.5 puntos**) ¿Qué estructura propones utilizar para almacenar todas peticiones y que las operaciones de sus búsquedas, inserciones y borrados tengan complejidad logarítmica?. Desarrolla la(s) clase(s) necesarias conforme a la respuesta que hayas dado en la pregunta anterior. En este apartado SÓLO tienes que escribir la cabecera de la

clase, atributos y constructores (si los consideras necesarios). Es decir, no debes añadir ningún método por el momento.

b) (**1 punto**) Escribe el método **pistaOcupada**(). Este método recibe como parámetro un entero que representa el minuto en el que se quiere comprobar si la pista estará ocupada. Si la pista está reservada el método debe devolver el número de vuelo que hizo la reserva. En caso contrario, el método deberá devolver null. El método debe ser recursivo. (**Si el alumno opta por la versión iterativa la puntuación será 0.5**).

c) (**1 punto**) Escribe un método **recibirPetición**. Este método recibe como parámetros el número de vuelo y el minuto en el que va a utilizar la pista y almacena dicha petición. Si la pista ya está reservada, simplemente se informará que ya está ocupada. El método debe ser recursivo. (**Si el alumno opta por la versión iterativa la puntuación será 0.5**).

d) (**1 puntos**) Para evitar posibles colisiones en la pista es necesario evitar que varios vuelos puedan coincidir en la pista. El tiempo máximo estimado de una avión en pista es de 3 minutos. Por tanto, una vez reservada la pista en un determinado minuto, no será posible asignar más reservas para los 3 minutos anteriores ni tampoco para los 3 minutos posteriores. Por ejemplo, si se reserva la pista en el minuto 25, no se podrán realizar reservas para los minutos 22,23,24,26,27,28 (tampoco para el 25). Escribe una nueva versión del método **recibirPeticion**() que evite las colisiones en pista según lo explicado en este apartado.

Solución:

a) ABB

```java
public class Flight {

    public int minute;
    public String idFlight;

    public Flight parent, left, right;

    public Flight(int minuto, String vuelo) {
        this.minute = minuto;
        this.idFlight = vuelo;
    }

}
```

```java
public class AirStrip {

    public Flight root;
```

b)

```java
public String isTaken(int minuto) {
    return this.search(root, minuto);
}

private String search(Flight requestFlight, int minuto) {
    if (requestFlight != null) {
        // search in the left child
        if (requestFlight.minute > minuto)
            return search(requestFlight.right, minuto);
        // search in the right child
        if (requestFlight.minute < minuto)
            return search(requestFlight.left, minuto);
        // return the id flight
        return requestFlight.idFlight;
    } else
        return null;
}
```

c)

```
public void saveRequest(int minuto, String vuelo) {
    if (isEmpty()) {
        root=new Flight(minuto,vuelo);
        return;
    }
    this.insert(root, minuto, vuelo);
}

private void insert(Flight requestFlight, int minuto, String vuelo) {
    if (requestFlight != null) {
        //the airtrip is taken
        if (requestFlight.minute == minuto) {
            System.out.println("Airtrip is already taken in " + minuto);
            return;
        }
        // busca por la derecha
        if (requestFlight.minute > minuto)
            if (requestFlight.right != null)
                insert(requestFlight.right, minuto, vuelo);
            else {
                requestFlight.right = new Flight(minuto, vuelo);
                requestFlight.right.parent = requestFlight;
            }
        if (requestFlight.minute < minuto)
            if (requestFlight.left != null)
                insert(requestFlight.left, minuto, vuelo);
            else {
                requestFlight.left = new Flight(minuto, vuelo);
                requestFlight.left.parent = requestFlight;
            }
    }
}
```

d)

```
public void saveRequest3(int minuto, String vuelo) {
    if (isEmpty()) {
        root=new Flight(minuto,vuelo);
        return;
    }
    this.insert3(root, minuto, vuelo);
}

private void insert3(Flight request, int minuto, String vuelo) {
    if (request != null) {
        if (request.minute + 3 > minuto && request.minute - 3 < minuto) {
            System.out.println("Airstrip is taken in "
                    + "(" + (request.minute - 3) + "-" + (request.minute + 3)+")");
            return;
        } else if (request.minute > minuto)
            if (request.right != null)
                insert3(request.right, minuto, vuelo);
            else {
                request.right = new Flight(minuto, vuelo);
                request.right.parent = request;
            }
        else if (request.minute < minuto)
            if (request.left != null)
                insert3(request.left, minuto, vuelo);
            else {
                request.left = new Flight(minuto, vuelo);
                request.left.parent = request;
            }
        }
}
```