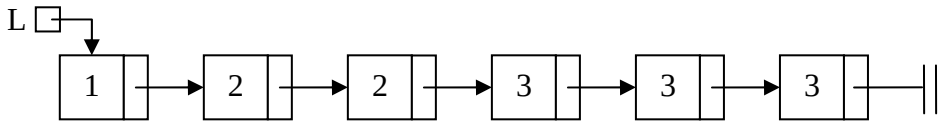
ESTRUCTURA DE DATOS Y ALGORITMOS **5-Febrero-2004**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**

1. Ejercicio (3 puntos)

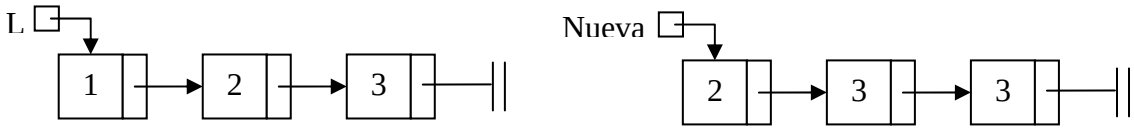
Diseñar y escribir en Java un subprograma que tenga como argumento de entrada una lista *L*, de la que se sabe que tiene nodos repetidos y estos están juntos. El subprograma creará otra lista cuyos nodos contendrán la información repetida en la lista *L y* eliminará dichos nodos de la lista *L*. También se requiere:

- Definir los tipos de datos para representar las listas.
- Especificar los métodos que se vayan a utilizar.
- Implementar los métodos relacionados con la eliminación, en caso de su uso.

Por ejemplo:



Después de ejecutar el subprograma:

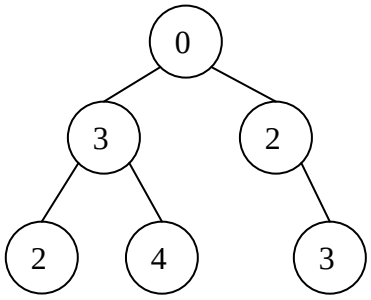


2. Ejercicio. (3 puntos)

Especificar, diseñar e implementar un método recursivo que dado un árbol binario de enteros, escriba por pantalla el peso de los nodos izquierdos y el de los derechos.
Nota: El contenido del nodo raíz siempre es 0.

```
Public void pesosIzquierdosYDerechos();
```

Ejemplo:



En la pantalla, obtendríamos el siguiente resultado:

Peso de nodos izquierdos: 5
Peso de nodos derechos: 9

3. Ejercicio (4 puntos)

La impresión de trabajos en una oficina se realiza siguiendo el orden establecido por una cola de prioridades. Los elementos de la cola de prioridades son de tipo Trabajo:

```
class Trabajo{
        int id;          //identificador de trabajo
        int llegada;     //minuto de llegada a la cola de prioridad
}
```

Hay cinco prioridades establecidas (1-5), siendo la 1 la de mayor prioridad.

En una cola de prioridad, los elementos se van despachando dependiendo de su prioridad en la impresión. Es decir, si en un momento dado pueden imprimirse dos trabajos, se imprimirá el primero aquel que tenga una prioridad mayor, aunque haya llegado posteriormente.

Especificación de los métodos de la clase **ColaConPrioridad**

void inicializar();
Inicializa la cola con prioridad a vacía.

boolean esVacia();
Decide si la cola con prioridad está vacía.

void cargarDatos(*String file***);**
Carga los datos almacenados en el fichero *file* en la cola con prioridad.

Trabajo **primero();**
Devuelve el primer elemento que entró en la cola.

void insertarElemento(*Object elem, int prioridad***);**
Añade el elemento *elem* como último entre los de prioridad *prioridad*.

void borrarElemento();
Borra primero(). No hace nada si la cola esta vacía.

Trabajo **primeroEnMinuto(***int m***);**
Devuelve el elemento de mayor prioridad de entre los que llegaron hasta el minuto *m.*
Si todos los trabajos de la cola llegaron en minutos superiores a *m*, devuelve null.

void borrarPrimeroEnMinuto(*int m***);**
Borra el elemento de mayor prioridad de entre los que llegaron hasta el minuto *m.*
Si todos los trabajos de la cola llegaron en minutos superiores a *m*, no borra ningún elemento

ESTRUCTURA DE DATOS Y ALGORITMOS **5-Febrero-2004**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**

SE PIDE:

1. Definir en Java una representación adecuada para la clase **ColaConPrioridad** basada en el tipo de **cola**, de tal forma que todas las operaciones anteriormente especificadas sean de O(1). (Podéis suponer que las operaciones del tipo de cola son de orden constante y que el número de prioridades también es constante).

2. Implementar las operaciones:
   `cargarDatos`, `primeroEnMinuto` y `borrarPrimeroEnMinuto`
   según la representación elegida.

3. A partir del fichero **colaTrabajos.txt** dónde cada línea contiene:
   - el número que identifica el trabajo
   - la prioridad del trabajo
   - el minuto de llegada a la cola de prioridades (número natural)

   En ese fichero los trabajos están ordenados por orden de llegada.

   Y utilizando las siguientes normas de impresión:
   - El primer trabajo en llegar se imprime inmediatamente
   - Cada trabajo se imprime como mínimo transcurridos 3 minutos desde la última impresión, según su prioridad.
   - Respetando la norma anterior y las prioridades de impresión, un trabajo se debe imprimir lo antes posible

   Diseñar e implementar un subprograma que escriba en pantalla, los números de trabajo y sus respectivos minutos de impresión. Dicha relación deberá estar ordenada según el minuto de impresión.

   Ejemplo:

   Para el siguiente fichero:

10	1	6
40	5	8
33	3	11
55	4	13
77	4	20
86	4	21
99	5	22
32	3	23

   Obtendríamos en la pantalla:

TRABAJO	MINUTO IMPRESION
10	6
40	9
33	12
55	15
77	20
32	23
86	26
99	29

Para la realización de este ejercicio se dispone de las siguientes clases:

```
class Token {
        //Inicializa el fichero
        Token(String file);

        //cierto si existen más líneas en el fichero, falso en caso contrario
        bolean hasMoreTokens();

        //devuelve los datos de una línea del fichero en un objeto de tipo Datos
        Datos getToken();
}
```

```
class Datos{
 //devuelve el identificador del trabajo
public int getIdTrabajo();
//devuelve la prioridad del trabajo
public int getPrioridad();
//devuelve el minuto de llegada
public int getMinuto();
}
```

```
class Trabajo{
//Constructor
Trabajo(int pIdTrabajo, int pMinuto);
 //devuelve el identificador del trabajo
public int getIdTrabajo();
//devuelve el minuto de llegada
public int getMinuto();
}
```

Anexo II: Especificación de las clases BTNode, BinTree y BinTreeItr

NOTA: Todo método necesario tendrá que ser implementado.

```
public class BTNode{
        Object content;
        BTNode left;
        BTNode right;
}

public class BinTree{
        BTNode root;
}
```

```
public class BinTreeItr{
        BTNode actual;
        BinTree bTree;
}
```