Estructura de Datos. Grado en Ingeniería Informática. Grupo A. Control 2 (1.75 puntos)

Nombre:……

En este ejercicio se trata de elaborar funciones que nos permitan hacer una representación gráfica de un árbol con información en los nodos. Para ello se define la clase `NodeB`:

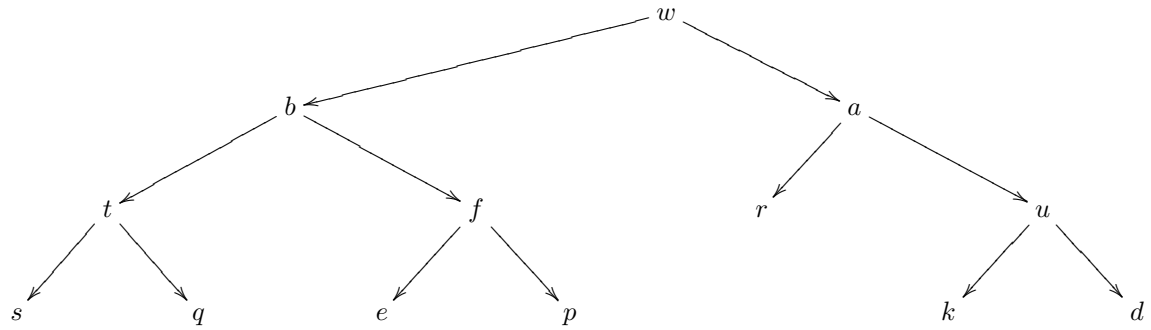
```
public class NodeB<T> {
    NodeB<T> lt;
    NodeB<T> rt;
    T elem;

    public NodeB(NodeB<T> i, T v, NodeB<T> d) {
        lr = i;
        elem = v;
        rt = d;
    }
    public NodeB(T v) {
        this(null,v,null);
    }
...
}
```

Como ejemplo de un dato de esta estructura tenemos el siguiente árbol:

```
    public static NodeB<Character> ejemplo =
        new NodeB<>(
                new NodeB<> (new NodeB<> (new NodeB<>('s'), 't', new NodeB<> ('q')), 'b', new NodeB<> (new NodeB
            'w',
            new NodeB<>(new NodeB<>('r'), 'a', new NodeB<> (new NodeB<>('k'), 'u', new NodeB<> ('d'))));
```

que representa al árbol de la figura siguiente



Como se ve en la figura, cada nodo del árbol está situado en una fila y en una columna distinta. Para poder pintar el árbol debemos conocer en qué fila y columna debe ir cada nodo así como las longitudes que deberán tener las flechas que van al hijo izquierdo y derecho de cada nodo.

La siguiente figura representa el mismo árbol pero resaltando la fila y columna donde debe colocarse cada nodo (`'w'` en la fila 0 columna 7, `'b'` en la fila 1 columna 3, `'a'` en la fila 1 columna 9, etc.)(filas y columnas empiezan en `0`). También se observa que la flecha que deberá ir de `'w'` a `'b'` deberá tener longitud horizontal 4 y la que va de `'w'` a `'a'` deberá tener longitud 2:

							w					
			b						a			
	t				f			r			u	
s		q		e		p				k		d

1. Método de clase `columnas` (1 pto.)

Define el método de clase

```
public static <S> int columnas(NodeB<S> ar)
```

que toma un árbol y devuelva cuántas columnas se necesitan para pintar el árbol. Así, el subárbol con raíz `'b'` vemos que necesita 7 columnas (3 que ocupa su hijo izquierdo, 3 de su hijo derecho y 1 la raíz). El `ejemplo` necesita 13 columnas (7+5+1).

Con la información que produce el método `columnas` no es suficiente para poder pintar el árbol. Para ello, vamos a crear otros métodos.

Supongamos que dado un árbol como el anterior, construimos otro en el que la información contenida en cada nodo es una terna (La clase `Tuple3` se proporciona). La primera componente de la terna es el valor del nodo del árbol inicial, la segunda, las columnas que necesita su rama izquierda para poderse pintar y la tercera las que necesita su rama derecha. En el ejemplo anterior, el nodo raíz `'w'` daría lugar a la terna `('w',7,5)`, el nodo `'b'` daría lugar a la terna `('b',3,3)`, etc.

Dado un árbol con estas características, calcular las columnas que se necesitan para pintarlo es mucho más sencillo.

2. Método de clase `columnasAM` (1 pto.)

Define el método de clase no recursivo

```
public static <S> int columnasAM(NodeB<Tuple3<S,Integer,Integer>> ar)
```

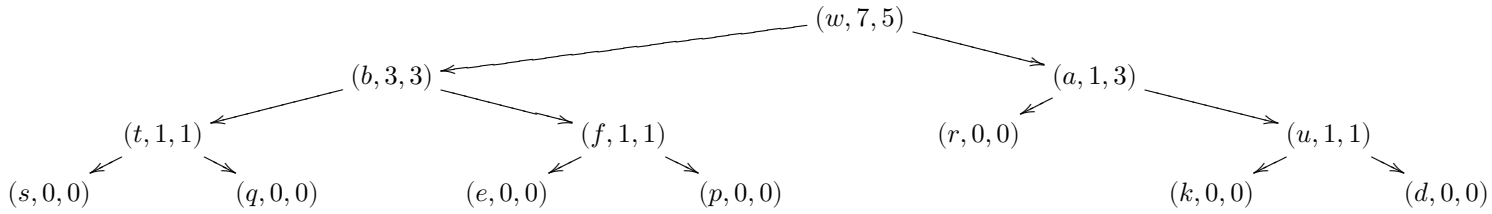
que calcula las columnas que necesita un árbol donde los nodos almacenan ternas como las mencionadas.

3. Método de clase `anota` (2 ptos.)

Define el método de clase

```
public static<S> NodeB<Tuple3<S,Integer,Integer>> anota(NodeB<S> ar)
```

que toma un árbol y devuelve otro árbol con ternas como valor (árbol con ternas posicionales). La primera componente de la terna es el valor del árbol original, la segunda indica las columnas que ocupa su hijo izquierdo y la tercera las columnas que ocupa su hijo derecho. Así, `anota(ejemplo)` devolverá el siguiente árbol:



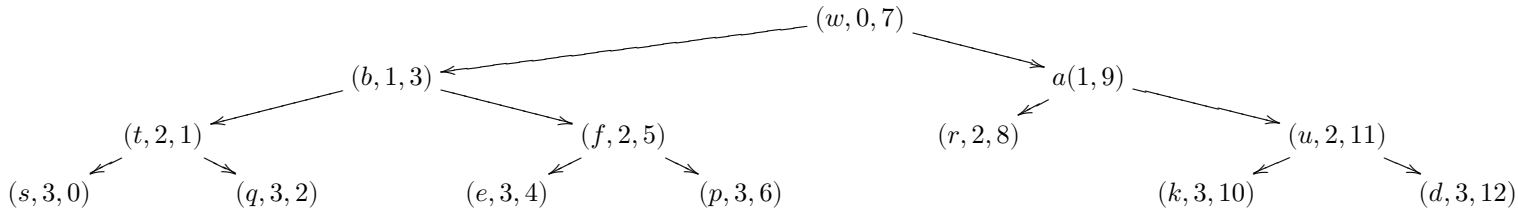
4. Método de clase `posCuadro` (3 ptos.)

Define el método de clase

```
public static <S> NodeB<Tuple3<S,Integer,Integer>> posCuadro(NodeB<Tuple3<S,Integer,Integer>> ar)
```

en el que dado un árbol con ternas posicionales (como devuelve el método `anota`), devuelve un árbol también con ternas. En cada terna, la primera componente es el valor del nodo (primera componente de la terna original), la segunda es la fila en la que debe pintarse ese nodo y la tercera la columna.

Así, si ejecutamos `posCuadro(anota(ejemplo))` devolverá el árbol:



Para su implementación, es conveniente apoyarse en otro método de clase privado

```
private static <S> NodeB<Tuple3<S,Integer,Integer>> posCuadrop(NodeB<Tuple3<S,Integer,Integer>> ar,
                                                               int fila,
                                                               int col)
```

con dos parámetros acumuladores, el primero indica la fila desde donde partimos en la cuadrícula, y el segundo la columna de la que partimos. Así, la implementación de `posCuadro` quedaría como sigue:

```
public static <S> NodeB<Tuple3<S,Integer,Integer>> posCuadro(NodeB<Tuple3<S,Integer,Integer>> ar) {
    return posCuadrop(ar,0,0);
}
```

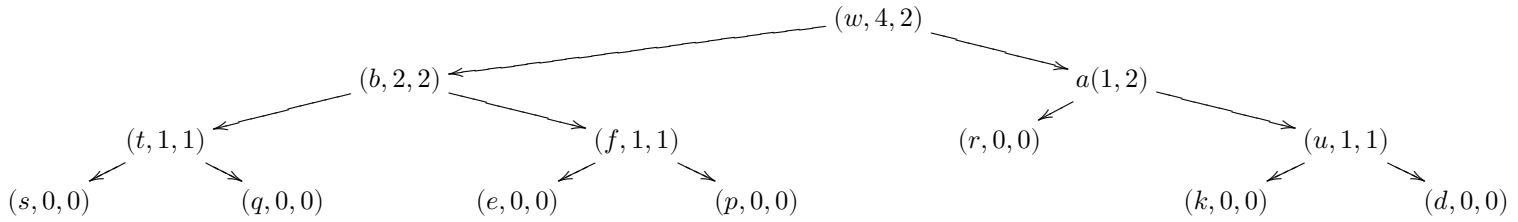
5. Método de clase `longFlecha` (2 ptos.)

Define el método de clase

```
public static <S> NodeB<Tuple3<S,Integer,Integer>> longFlecha(NodeB<Tuple3<S,Integer,Integer>> ar)
```

en el que dado un árbol con ternas posicionales, devuelve otro donde cada nodo contiene también una terna. La primera componente de la terna es el valor del árbol original, la segunda es la longitud horizontal en cuadrículas que debe tener la flecha izquierda que sale de ese nodo hacia su hijo izquierdo y la tercera, la longitud horizontal que debe tener la flecha derecha que sale de ese nodo hacia su hijo derecho. Por ejemplo, las flechas que salen del nodo `'w'` tienen una longitud 4 a su izquierda y 2 a su derecha.

Así, si ejecutamos `longFlecha(anota(ejemplo))` obtendremos el árbol



6. Pregunta. ¿Cuál será el valor del nodo raíz del árbol resultante de ejecutar la siguiente expresión? (1 pto.)

```
posCuadro(anota(longFlecha(anota(ejemplo))))
```