

# Algoritmos y Estructuras de Datos

Curso 2020

# Enunciado

Se define el valor de trayectoria pesada de una hoja de un árbol binario como la suma del contenido de todos los nodos desde la raíz hasta la hoja multiplicado por el nivel en el que se encuentra. Implemente un método que, dado un árbol binario, devuelva el valor de la trayectoria pesada de cada una de sus hojas.

Considere que el nivel de la raíz es 0. Para el ejemplo de la figura: trayectoria pesada de la hoja 4 es:  $(4 * 2) + (1 * 1) + (7 * 0) = 9$

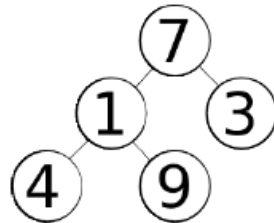


Figura 1: Árbol Binario de la trayectoria pesada

# Forma de encarar el ejercicio

El ejercicio se puede dividir en tres partes:

1. Recorrido del árbol.
2. Procesamiento de los nodos.
3. Almacenamiento del resultado.

Existen diferentes maneras de recorrer árboles, cada cual con sus ventajas y desventajas. Un recorrido recursivo puede ser interesante, en especial el preorden o posorden. Cualquiera de estos se puede aplicar al ejercicio.

# Implementación

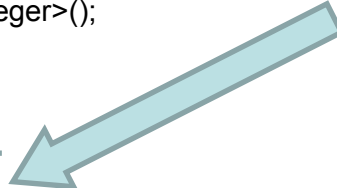
```
public class TrayectoriaPesada {
```

```
    public ListaGenerica<Integer> calcular(ArbolBinario<Integer> arbol) {  
        ListaGenerica<Integer> lista = new ListaEnlazadaGenerica<Integer>();  
        recorrido(arbol, 1, 0, lista);  
        return lista;  
    }
```

```
    private void recorrido(ArbolBinario<Integer> arbol, int nivel, int sumaParcial, ListaGenerica<Integer> lista) {  
        if (!arbol.esVacio()) {  
            procesar(arbol, nivel, sumaParcial, lista);  
            if (arbol.tieneHijoDerecho()) {  
                recorrido(arbol.getHijoDerecho(), nivel + 1, sumaParcial + nivel * arbol.getDatoRaiz(), lista);  
            }  
            if (arbol.tieneHijoIzquierdo()) {  
                recorrido(arbol.getHijoIzquierdo(), nivel + 1, sumaParcial + nivel * arbol.getDatoRaiz(), lista);  
            }  
        }  
    }
```

```
    private void procesar(ArbolBinario<Integer> arbol, int nivel, int sumaParcial, ListaGenerica<Integer> lista) {  
        if (arbol.esHoja()) {  
            lista.agregarFinal(sumaParcial + arbol.getDatoRaiz() * nivel);  
        }  
    }  
}
```

En el recorrido en preorden contamos con tres parámetros: el árbol (**arbol**), el número de nivel actual (**nivel**), la suma acumulada desde la raíz hasta el nodo actual (**sumaParcial**) y la lista resultado (**lista**).



Tener los métodos auxiliares como **private** permite que los detalles de implementación permanezcan ocultos.

# Errores comunes

1. No devolver los valores. El enunciado pide devolver los valores, por lo que imprimirlos mediante **System.out.print()**; no sería correcto. Hace falta utilizar una estructura que permita almacenar una cantidad variable de elementos para luego devolver la estructura completa.
2. No sumar el valor de la hoja como parte de la trayectoria. Al momento de agregar el valor de la trayectoria hace falta incluir el valor de la hoja multiplicado por su nivel. Manteniendo la idea explicada hasta ahora, el siguiente resultado es incorrecto:

```
private void procesar(ArbolBinario<Integer> arbol, int nivel, int sumaParcial,
ListaGenerica<Integer> lista) {
    if (arbol.esHoja()) {
        lista.agregarFinal(sumaParcial); //Incorrecto falta calcular la hoja.
    }
}
```

# Errores comunes (cont.)

3. Uso del operador “++” a derecha para incrementar el nivel. El operador “++” a derecha, por ejemplo en el caso de “nivel++”, hace que el valor de la variable nivel se modifique, pero este incremento se realiza después de devolver el valor original. Si se usa nivel++ como incremento de nivel, y esta expresión esta directamente en el llamado recursivo, el efecto deseado no se cumplirá. En cada llamada recursiva, el parámetro n siempre tomara el mismo valor, y no se estará incrementando el nivel.
4. De acuerdo a la definición vista en clase de **ArbolBinario**, los métodos **getHijolzquierdo** y **getHijoDerecho** de **ArbolBinario** nunca devuelven **null**, y por lo tanto la siguiente pregunta:  
**if (arbol.getHijolzquierdo( )==null)**  
nunca va a ser **True**.
5. Preguntar si **this** es **null**. El lenguaje Java garantiza que “**this**” jamás pueda ser **null**, ya que al momento de invocar un método de instancia en una variable con referencia **null**, se dispara un error de **NullPointerException**.
6. No avanzar en la recursión, ya que se utiliza en cada llamado recursivo el mismo árbol enviado como parámetro en lugar de utilizar los sub árboles izquierdo y derecho. Y así incurrir en una recursión infinita.

# Otra Alternativa

```
public class TrayectoriaPesada {
```

```
    public ListaGenerica<Integer> calcular(ArbolBinario<Integer> arbol) {  
        ListaGenerica<Integer> lista = new ListaEnlazadaGenerica<Integer>();  
        recorrido(arbol, 1, 0, lista);  
        return lista;  
    }
```

```
    private void recorrido(ArbolBinario<Integer> arbol, int nivel, int sumaParcial, ListaGenerica<Integer> lista) {  
        if (!arbol.esVacio()) {  
            procesar(arbol, nivel, sumaParcial, lista);  
            if (arbol.tieneHijoDerecho()) {  
                nivel++;  
                sumaParcial = sumaParcial + nivel * arbol.getDatoRaiz();  
                recorrido(arbol.getHijoDerecho(), nivel, sumaParcial, lista);  
            }  
            if (arbol.tieneHijoIzquierdo()) {  
                nivel++;  
                sumaParcial = sumaParcial + nivel * arbol.getDatoRaiz();  
                recorrido(arbol.getHijoIzquierdo(), nivel, sumaParcial, lista);  
            }  
        }  
    }
```

```
    private void procesar(ArbolBinario<Integer> arbol, int nivel, int sumaParcial, ListaGenerica<Integer> lista) {  
        if (arbol.esHoja()) {  
            lista.agregarFinal(sumaParcial + arbol.getDatoRaiz() * nivel);  
        }  
    }
```

En el recorrido en preorden contamos con tres parámetros: el árbol (**arbol**), el número de nivel actual (**nivel**), la suma acumulada desde la raíz hasta el nodo actual (**sumaParcial**) y la lista resultado (**lista**).

**Diferencia con la solución anterior:** Las operaciones aritméticas el número de nivel actual (**nivel**) y la suma acumulada desde la raíz hasta el nodo actual (**sumaParcial**) se realizan fuera del llamado recursivo.

**¿Resuelve lo solicitado?**

**¿Por qué?**