Universidad del Valle

# Práctica 8

**Fecha límite de entrega: 17 de Septiembre de 2021**

Desarrolle los siguientes ejercicios los cuales se deben tener en cuenta: árboles, listas y recursividad utilizando el paradigma funcional y lenguaje de programación JavaScript. Usted debe enviar el código fuente y pasar los test a través de la plataforma INGInious M-IDEA (http://ingin.ddns.net/courselist). Puede apoyarse de la herramienta repl.it (https://repl.it/~)

**Documento de repaso**
https://docs.google.com/document/d/1kWkNKLZ9dak7lEcAwWuF0Eq4uah1AJ4UhzOO9Oz0G_U/edit?usp=sharing

**IMPORTANTE**

Un árbol contiene un dato en su raíz y un número cualquiera de hijos, que también son árboles. Es común utilizar estructuras para lograr su representación en el código. Cada nodo tiene un valor que puede ser numérico y dos sub árboles que poseen nodos que a su vez tiene un valor. Un ejemplo a continuación.

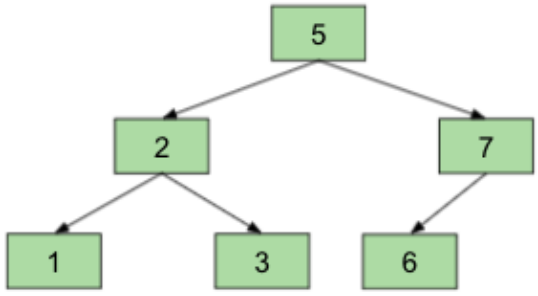
**Recuerde no usar for, while, etc.**

## 1 Árbol BST (Binary Search Tree)

### EJEMPLO

Realiza la representación del árbol de búsqueda binaria (BST) e implemente en javascript usando (value, left, right) **Árbol: [5, 2, 7, 6, 1, 3]**

**Representación del árbol**




**Código Javascript**

```
const tree2 = {
  value: 5,
  left: {
    value: 2,
    left: {
      value: 1
    },
    right: {
      value: 3
    }
  },
  right: {
    value: 7,
    left: {
      value: 6
    }
  }
}
```

**1. Desarrolle la función leafT que cuente el numéro de hojas que posee un arbol.**

```
const tree = {"value":5, left:{"value":4}, right:{"value":6}}
```

```
leafT(null)
leafT({})
leafT({"value":5})
leafT({"value":5, left:{"value":4}})
leafT({"value":5, right:{"value":6}})
leafT( tree )
```

```
0
0
1
?
?
2
```

## Solución

**En el código recuerde poner:**

const { cons, first, rest, isEmpty, isList,length } = require('functional-light');

**crear el arbol** const tree = {"value":5, left:{"value":4}, right:{"value":6}}

**Código:**

```
/**
* Contrato: <leafT><tree> --> <number>
* Propósito: Retornar el conteo de todas las hojas del árbol
* @param tree Variable objeto para Árbol.
* @example:
* //return 0
* leafT(null)
*/

const leafT = (tree) => {
    if (!tree) {
        return 0
    } else if (!tree.value) {
        return 0
    } else if (!tree.left && !tree.right) {
        return 1
    } else {
        return leafT(tree.left) + leafT(tree.right)
    }
}
```

**Invocación:**

```
console.log(leafT(null))
console.log(leafT({}))
console.log(leafT({"value":5}))
console.log(leafT({"value":5, left:{"value":4}}))
console.log(leafT({"value":5, right:{"value":6}}))
console.log(leafT(tree))
```

**2.** Desarrolle la función internalT que cuente el número de nodos internos que posee un árbol.

```
const tree = {"value":5, left:{"value":4}, right:{"value":6}}
```

```
leafT(null)
leafT({})
leafT({"value":5})
leafT({"value":5, left:{"value":4}})
leafT({"value":5, right:{"value":6}})
leafT( tree )
```

```
0
0
1
?
?
2
```

**3.** Desarrolle la función leafTL que retorne una lista con los valores que hay en las hojas de un árbol.

```
const tree = {"value":5, left:{"value":4}, right:{"value":6}}
```

*Invocación*

```
leafTL(null)
leafTL({})
leafTL({"value":5})
leafTL({"value":5, left:{"value":4}})
leafTL({"value":5, right:{"value":6}})
leafTL( a_tree )
```

*Salida*

```
[]
[]
?
[4]
?
?
```

**4**. Desarrolle la función matchTL que recibe un árbol y una lista e indica sí se puede construir la lista a partir de un recorrido en el árbol.

```
const tree = {"value":5, left:{"value":4}, right:{"value":6}}
```

*Invocación*

```
matchTL({}, [])
matchTL(null, [])
matchTL({"value":5}, null)
matchTL({"value":5}, [5])
matchTL(a_tree, [4, 5, 6])
```

*Salida*

```
false
?
false
?
?
```

**5.** Realice una función paresTree que reciba un árbol binario de números y cuente la cantidad de nodos con valores pares que existen en él.

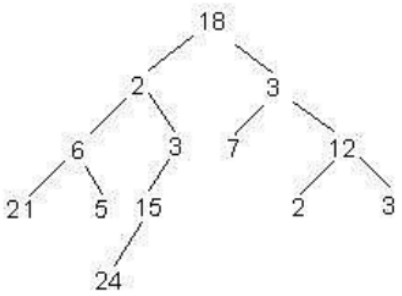



Figura No. 1: Tree

*Entradas*

```
paresTree(Tree)
```

*Salidas*

```
6
```

**6.** Realice una función LisTree que reciba un árbol binario de números y **liste** todos los nodos con valores pares que existen en él. Puesto que la lista que se genera depende del orden en que se recorra el árbol

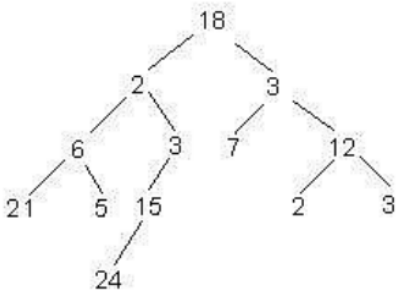



Figura No. 1: Tree

*Entradas*

```
Tree
```

*Salidas*

```
[ 18, ?, ?, ?, 12, ? ]
```

**7.** Realice una función deepTree que reciba un árbol binario y determine su profundidad (la cantidad de niveles que tiene).

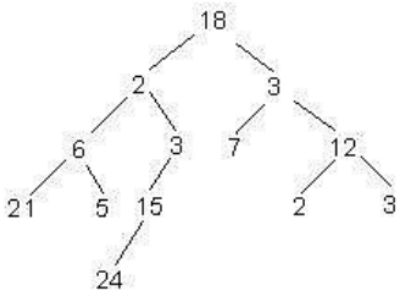



Figura No. 1: Tree

*Entradas*

```
deepTree(Tree)
```

*Salidas*

```
5
```

**8.** Realice una función searchTree que reciba un árbol binario de búsqueda y un elemento y determine si el elemento está o no en el árbol.

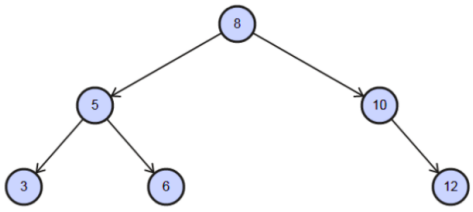



Figura No. 1: Tree

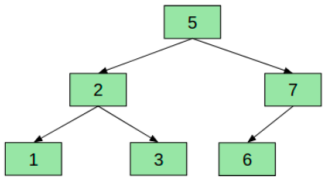



Figura No. 1: TreeOne

*Entradas*

```
searchTree (Tree, 12)
searchTree (Tree, 8)
searchTree (Tree, 0)
searchTree (TreeOne, 2)
searchTree (TreeOne, 1)
searchTree (TreeOne, 4)
```

*Salidas*

```
true
?
false
?
true
?
```

**9.** Realice una función que reciba un árbol binario de búsqueda que retorne una lista con los valores de los nodos en el orden recorrido, de acuerdo a un **recorrido preorden**.

const tree

const treeOne

*Entrada*

```
preOrden(tree)
preOrden(treeOne)
```

*Salida*

```
[ 6, ?, 1, ?, 4, ?, 8, ?, 11, ?, 12 ]
[ ?, 2, 1, ?, 7, ? ]
```

**10.** Realice una función que reciba un árbol binario de búsqueda que retorne una lista con los valores de los nodos en el orden recorrido, de acuerdo a un **recorrido inorden**.

const tree

const treeOne

*Entrada*

```
inOrden(tree)
inOrden(treeOne)
```

*Salida*

```
[ 1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13 ]
[ 1, 2, 3, ?, 6, 7 ]
```

**11.** Realice una función que reciba un árbol binario de búsqueda que retorne una lista con los valores de los nodos en el orden recorrido, de acuerdo a un **recorrido postorden**.

const tree

const treeOne

*Entrada*

```
postOrden(tree)
postOrden(treeOne)
```

*Salida*

```
[ 1, ?, 5, ?, 8, 10, 12, 11, 13, ?, 6 ]
[ 1, ?, ?, 6, 7, ? ]
```