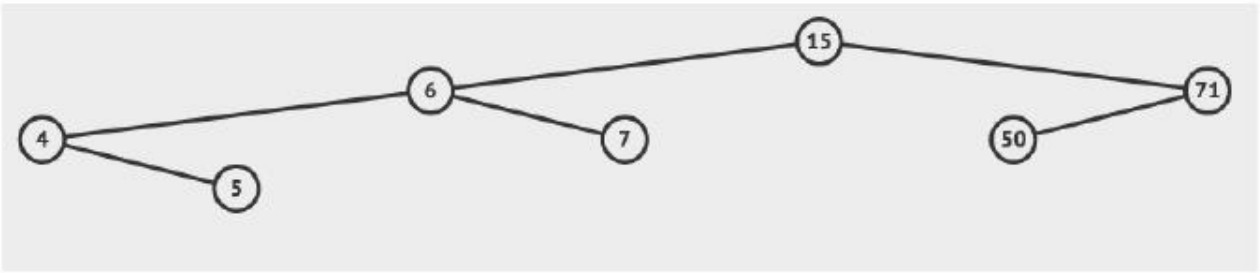
Conceptos

1. Sea el siguiente árbol T representado en la figura. [20 puntos]



- Denote la secuencia pre-order al recorrer el árbol desde la raíz.

[15, 6, 4, 5, 7, 71, 50]

- Denote la secuencia in-order al recorrer el árbol desde la raíz.

[4, 5, 6, 7, 15, 50, 71]

- Denote la secuencia post-order al recorrer el árbol desde la raíz.

[5, 4, 7, 6, 50, 71, 15]

- Denote la altura máxima del árbol.

4 o 3

- Denote si el árbol está balanceado o no.

Si, dif H = 1

2. Responda con V o F para afirmar si las siguientes sentencias son verdaderas o falsas. Para el caso de las afirmaciones falsas justifique su respuesta.[8 Puntos]

a. F La complejidad de tiempo para buscar en un BST depende de la altura, es decir $\mathcal{O}(h)$.

Si esta balanceado log(n)
O(h)

b. V Si un BST está balanceado las inserciones tienen complejidad $\mathcal{O}(lg(n))$.

d. F El recorrido in-order de un BST tiene complejidad $\mathcal{O}(lg(n))$ porque depende de la altura.

O(n)

Análisis

Sea la siguiente función:

```
1    static void f(Node root) {
2        if(root == null) {
3            return;
4        }
5
6        f(root.right);
7        System.out.println(root.value);
8        f(root.left);
9    }
```

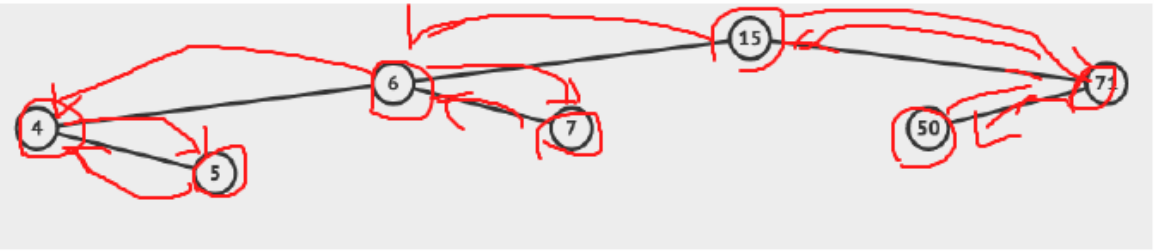
left
Print
Right

⟹

Right
Print
left

[12 puntos]

- Describa la secuencia de salida para el árbol:



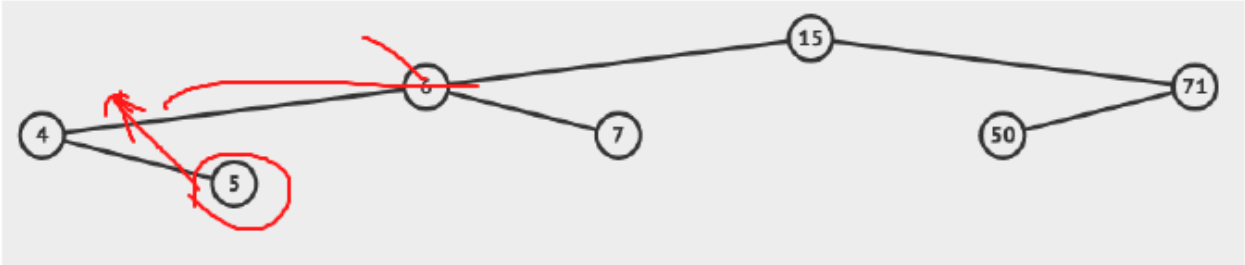
[71,50,15,7,6,5,4]

- Describa la complejidad de tiempo de ejecución para el algoritmo en términos de $\mathcal{O}(f(n))$.

O(n)

Desarrollo

Diseñe un algoritmo que como entrada reciba dos parámetros, la raíz de un bst y un número entero x. La salida del algoritmo debe ser el sucesor de x en el árbol. Si no existe un sucesor de x entonces el algoritmo debe retornar 0. Después de diseñar su algoritmo realice el análisis respectivo para indicar la complejidad de tiempo de este en términos de $\mathcal{O}(f(n))$ [20 puntos]
Por ejemplo sea el siguiente árbol:



Respecto el problema se puede observar:

- El sucesor de 4 es 5
- El sucesor de 5 es 6
- El sucesor de 50 es 71
- El sucesor de 71 es 0

O(n)

O(h)

1) Recursividad (root, x)
Node sucesor = null
While(root != null)
Recursividad(root.left, x)
else
recursividad(root.right, x)

2) inorden

for
arr[i] == x
return arr[i+1]
else
arr[i+1] = null
return 0