uc3m | Universidad Carlos III de Madrid

OpenCourseWare

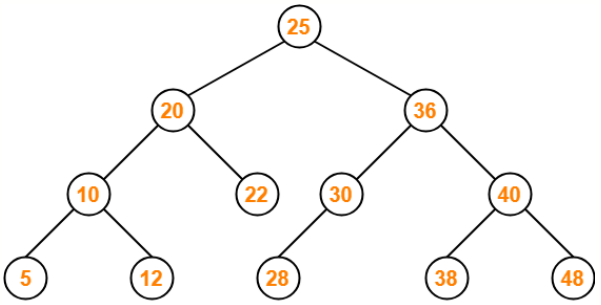
Tema 5 - Árboles

Hoja 2 de Problemas (Problemas de exámenes)

Problema 1 (Mayo 2020)

MyBST es una subclase de la clase BinarySearchTree. En un árbol binario, dos nodos son **primos** si están en el mismo nivel (tienen la misma profundidad) y además sus padres son hermanos.

Por ejemplo, en el siguiente árbol:



Binary Search Tree

- Son primos: (10,30), (10,40), (22,30), (22,40), (28,38), (28, 48).

Se pide:

En la clase MyBST, implementa un método **check_cousins(a,b)**, que toma dos elementos a y b, y devuelve True si sus nodos son primos, y False en otro caso.

Nota: En la implementación de BinarySearchTree que usamos en 2022, la clase BinaryNode no tiene el atributo parent.

Problema 2 (Junio 2020):

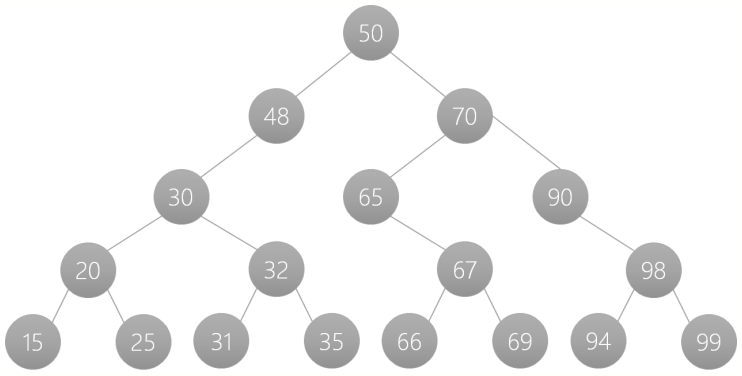
Sea MyBST la clase que implementa un árbol binario de búsqueda en Python (es una versión reducida que sólo incluye los métodos necesarios para este problema).

En un árbol binario, el **mínimo ancestro común** *(lowest common ancestor)* de dos nodos *n* y *m* es el nodo que contiene a los nodos n, m y a todos sus descendientes.

Implementa una función, ***lwc***, que reciba dos enteros a y b, encuentre sus nodos y obtenga el mínimo ascendiente común de ambos nodos. La función debe devolver el elemento asociado de dicho nodo ascendente. Si a ó b, no existen, la función debe devolver None. Si el árbol está vacío, la función debe devolver None.

Pista: La solución más fácil sigue un enfoque recursivo. Puedes utilizar una función auxiliar.

Dado el siguiente árbol, veamos algunos ejemplos:



tree.lwc(48,70)=50

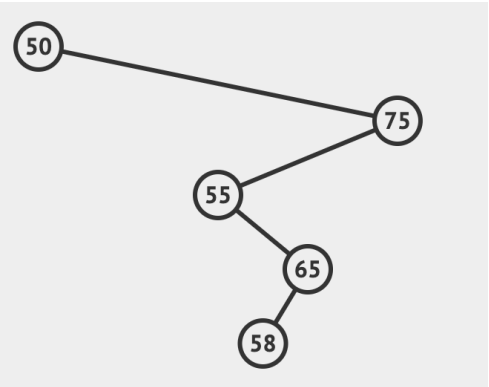
- tree.lwc(30,70)=50
- tree.lwc(30,65)=50

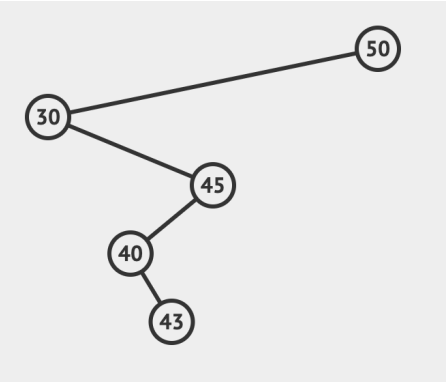
- tree.lwc(20,32)=30
- tree.lwc(67,90)=70
- tree.lwc(69,99)=70
- tree.lwc(31,94)=50
- tree.lwc(15,25)=20
- tree.lwc(15,35)=30
- tree.lwc(15,32)=30
- tree.lwc(20,30)=30
- tree.lwc(67,70)=70
- tree.lwc(30,35)=30

Problema 3 (abril 2021):

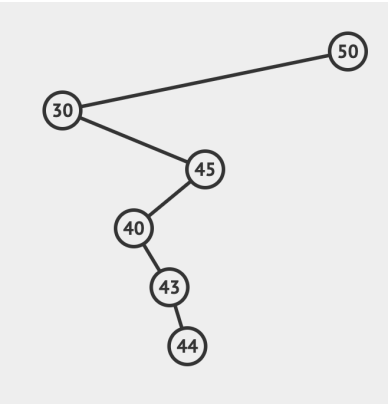
En la clase MyBST, implementa un método recursivo, **is_zig_zag**, que devuelva True si el árbol tiene forma de zizag y False en otro caso. En un árbol con forma de zig-zag, si un nodo es hijo izquierdo de su padre, dicho nodo debe ser un nodo hoja o tener únicamente un hijo derecho. De la misma forma, si el nodo es el hijo derecho de su padre, podrá ser hoja o tener un único hijo izquierdo.

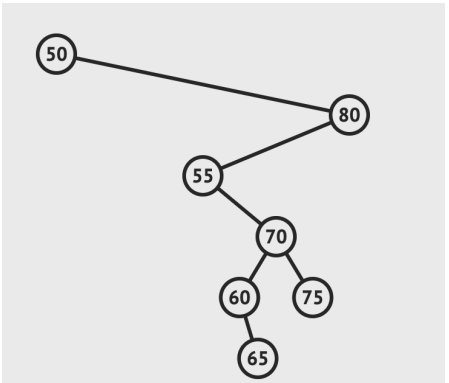
Los siguientes ejemplos son árboles con forma de zigzag:





Los siguientes árboles no tienen forma de zigzag.

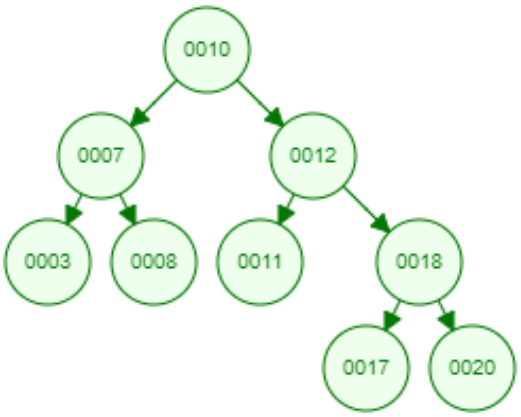




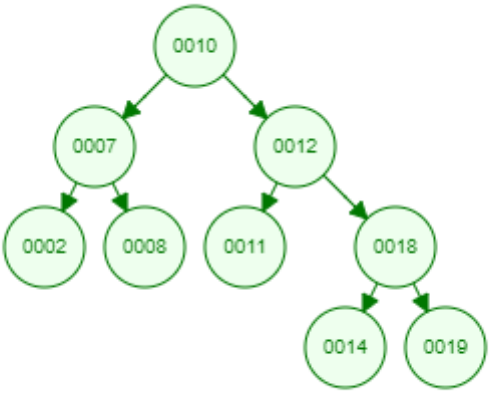
Problema 4 (abril 2021):

En la clase MyBST, implementa un método recursivo, **is_left_odd_right_even**, que devuelva True si en cada nodo del árbol el elemento de su hijo iziquierdo es impar y el elemento del hijo derecho es par. En cualquier otro caso, devuelve False.

Ejemplo: **is_left_odd_right_even** True

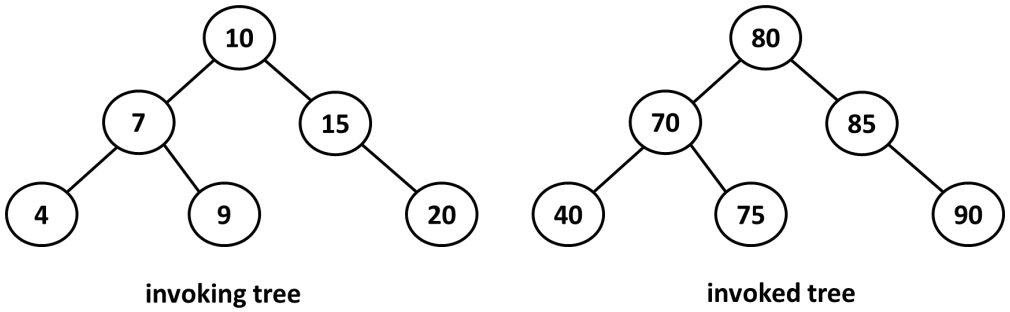


Ejemplo **is_left_odd_right_even** False



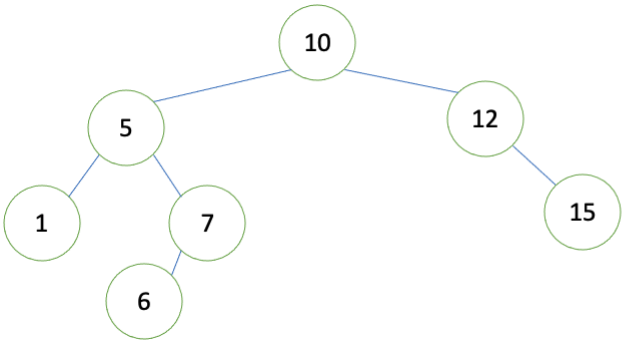
Problema 5 (abril 2021):

En la clase MyBST, implementa un método recursivo **is_same_shape** que reciba como argumento un objeto de la clase MyBST. El método debe comprobar si ambos árboles tienen la misma forma. Por ejemplo, en el siguiente ejemplo, el método deberá devolver True.



Problema 6 (abril 2021): En la clase MyBST, implementa un método recursivo, **closest**, que reciba un número entero y devuelva el elemento en el árbol más cercano a dicho valor entero. Si en el árbol existen dos valores que están a la misma distancia, se deberá devolver el mayor de ellos.

Por ejemplo, en el siguiente árbol, para el argumento de entrada, value = 7, el método debería devolver 7 (es el valor más cercano que existe en el árbol). Sin embargo, si el método recibe como parámetro de entrada el valor 14, el método deberá devolver 15. Si el valor de entrada es 11, como existen dos nodos en el árbol con igual distancia (que son 10 y 12), se deberá devolver el mayor, es decir, 12.



Problema 7 (abril 2021): En la clase MyBST, implementa un método recursivo, **isBST**, que compruebe si el árbol es un árbol binario de búsqueda.

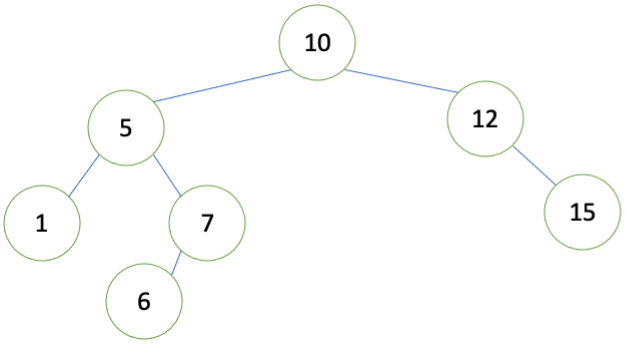
Problema 8 (abril 2021)

En la clase MyBST, implementa un método recursivo, **update_adding_left_child**, que modifique el elemento de cada nodo sumándole todos los valores que aparecen en su hijo izquierdo.

```
Input
        1
       / \
      2   3
     / \   \
    4   5   6
Output:
       12
      / \
     6   3
    / \   \
   4   5   6
```

Problema 9: (examen simulacro abril 2020)

En la clase MyBST, implementa un método **get_non_leaves** que devuelva una lista (de Python) con los elementos de los nodos no hojas. La lista debe estar ordenada de mayor a menor. No se puede utilizar ningún algoritmo para ordenar la lista o para invertirla.



En el anterior árbol, el método debería devolver la siguiente lista: [12, 10, 7, 5]

Problema 10: (examen mayo 2018)
Implementa una función, **array2bst**, que reciba una lista (de Python) ordenada en forma ascendente y devuelve un árbol binario de búsqueda (objecto de MyBST) que esté balanceado.

Problema 11: (examen mayo 2019)
En la clase BinaryNode, implementa el operador == (__eq__) que reciba un nodo, other_node, y que devuelva True si ambos nodos son iguales, y False en otro caso. Dos nodos son iguales si tienen el mismo elemento, y además sus hijos izquierdo son iguales, y sus hijos derecho son iguales, también.
En la clase BinaryTree, implementa el operador == (__eq__) que reciba un árbol binario, other_tree, y que devuelva True si ambos árboles son iguales, y False en otro caso. Dos árboles binarios son iguales si sus nodos raices son iguales.
Nota: la solución ya está incluida en el fichero bintree.py

Problema 12: (examen junio 2019)

Implementa una función, **mirror**, que reciba un árbol binario y que lo transforme en su árbol espejo. Veamos un ejemplo: