Dentro del método del nodo habría que tomar como primera referencia al valor de nuestro nodo y compararlo con el insertado. Hay que tener cuidado que el nodo no tenga valores que ya estén en el árbol, en ese caso no se inserta el nodo y se retornará falso. De lo contrario habría que revisar si corresponde ir al subárbol derecho o izquierdo, viendo si es mayor o menor y así recorrer recursivamente el árbol hasta encontrar un hijo que esté sin ocupar y que cumpla el orden del árbol.

Para el árbol lo único que habría que hacer sería retornar la llamada a insertar de la raíz de nuestro árbol siempre que la raíz no esté nula. Si la raíz es nula decimos que el nuevo nodo es nuestra raiz

Precondiciones:

Para el árbol hay que revisar que tanto la etiqueta del nodo como el nodo sean válidos, que no sean null.

Para el nodo no tendríamos que revisar nada ya que el árbol se encarga de tomar que casos son válidos

Postcondiciones:

Del método del nodo si se encuentra una etiqueta a la nuestra se retorna false ya que no se pueden repetir claves. Si el dato se logra insertar se retorna true, colocando al nuevo nodo como un hijo de la que antes era una hoja.

Del método del árbol: se revisa la validez de los datos y si no cumplen las condiciones se retorna false. Si la raíz es nula insertamos el nodo en la misma. Y sino llamamos al insertar del nodo de la raíz y retornamos el resultado de esta.

Orden de ejecución:

El orden de ejecución de algoritmo sería de O(n) en el peor de los casos ya que el peor de los casos sería donde todos sean hijos de lado izquierdo o de lados derecho ya que habría que pasar por cada nodo hasta llegar hasta la única hoja e insertarlo ahí. Por ejemplo, en un árbol en el cual se insertaron en orden el 1,2,3,4,5 y queremos insertar el 6, ahí tendríamos que pasar por el 1 después al 2 y así sucesivamente hasta llegar al 5.

Por otro lado, en un árbol perfectamente balanceado, siendo el mejor de los casos tendríamos un orden de ejecución de O(log N) en base 2.

Insertar(TAElemento)

Comparación 🡨TAElemento.getClave().comparableTo(clave) //O(1)

Si comparación es 0 //O(1)

Retornar falso //O(1)

Sino

Si comparación es menor a 0 //O(1)

Si hijoIzquierdo es null //O(1)

hijoIzquierdo 🡨TAElemento //O(1)

retornar verdadero //O(1)

Sino

Retornar hijoIzquierdo.insertar(TAElementos) //Depende de la llamadas que hagamos

Sino

Si hijoDerecho es null //O(1)

hijoDerecho 🡨 TAElemento //O(1)

retornar verdadero //O(1)

Sino

Retornar hijoDerecho.insertar(TAElemento) //Depende de las llamadas que hagamos

Hay que contar la cantidad de nodos que no tienen hijo derecho ni izquierdo (caso base), revisando siempre que no intente recorrer un hijo que no esté.

Precondiciones:

No puedo que se cuenten las hojas cuando la raíz es nula.

PostCondiciones:

Retorno la cantidad de hojas si es posible

Orden de ejecución de O(n) ya que tengo que recorrer cada nodo y chequear los hijos de este

ContarHoja()

Si el HijoIzquierdo y el hijoDerecho son null

Retornar 1

Si el hijoIzquierdo es null

Retornar hijoderecho.contarhojas()

Si el hijoDerecho es null

Retornar hijoIzquierdo.contarhojas()

Retornar hijoIzquierdo.contarhojas() + Retornar hijoderecho.contarhojas()

Hay que recorrer cada nodo del arbol e ir sumando su valor al total, pasando por cada hijo llegando hasta cada hoja y cuidar nunca recorrer una hoja que no esté.

Precondicion:

La raíz no puede ser nula

Postcondicion:

Retornar si es posible la suma de los nodos, sino retornar 0.

calcularSumaTotal()

Si el hijoIzquierdo y el derecho son nulos entonces

Retornar nodo.getClave()

SI hijoIzquierdo es nulo entonces

Retornar nodo.getClave() + HijoDerecho.calcularSumaTotal()

SI hijoDerecho es nulo entonces

Retornar nodo.getClave() + HijoIzquierdo.calcularSumaTotal()

Retornar nodo.getClave() + HijoDerecho.calcularSumaTotal() + HijoIzquierdo.calcularSumaTotal()

El algoritmo tiene un orden de O(n) ya que tiene que calcular la suma de cada n elemento siendo el caso base cada hoja del árbol y para calcular los nodos que no son hojas tiene que hacer la suma de sus hijos más si mismos, teniendo que recorrer cada nodo para obtener su valor.

Hay que recorrer el arbol y contar la cantidad de nodos que tengo en un determinado nivel, el nivel me indicará cuantos veces tendré que bajar (cuantas veces voy a recurrir a revisar un hijo)