Obtener la menor clave del árbol

1. Descripción en lenguaje natural

Para este algoritmo se debe recorrer el árbol comenzando por la raíz y recorriendo recursivamente los hijos izquierdos hasta donde sea posible, ese último nodo a la izquierda del todo va a ser el de menor clave.

2.

Precondiciones:

El árbol no está vacío.

El árbol BB está correctamente estructurado

Postcondición:

Devolvemos el nodo con la clave más pequeña

El árbol no se ve modificado

3. Casos de prueba

Árbol con solo un nodo. Se retornará ese nodo

Árbol con solo nodos a la izquierda. El último nodo a la izquierda del todo será el retornado

Árbol balanceado promedio. Va a ir lo más a la izquierda posible dentro de este

4. Seudocódigo

Arbol:

obtenerMenorClave()

Inicio

Si raíz es null entonces

    Retornar null

Sino

    Retornar Raíz.obtenerMenorClave()

Nodo:

obtenerMenorClave()

Si hijoIzq es null

Retornar this

Sino

retornar hijoIzq.obtenerMenorClave()

5. Análisis de orden del tiempo de ejecución

El mejor caso es cuando el árbol está balanceado y tengo por ejemplo 7 nodos pero recurro a la recursión 3 veces, siendo de orden O(log n). El peor caso sería un árbol con todos hijos izquierdos y tendría un orden de ejecución de O(n)

Obtener la mayor clave del árbol

1. Descripción en lenguaje natural

Para este algoritmo se debe recorrer el árbol comenzando por la raíz y recorriendo recursivamente los hijos derechos hasta donde sea posible, ese último nodo a la derecha del todo va a ser el de mayor clave.

2. Precondiciones y Postcondiciones

Precondiciones:

El árbol no está vacío.

El árbol BB está correctamente estructurado.

Postcondición:

Devolvemos el nodo con la clave más grande.

El árbol no se ve modificado.

3. Casos de prueba

Árbol con solo un nodo → se retornará ese nodo.

Árbol con solo nodos a la derecha → el último nodo a la derecha del todo será el retornado.

Árbol balanceado promedio → irá lo más a la derecha posible dentro de este.

4. Seudocódigo

Arbol:

obtenerMayorClave()

Inicio

    Si raíz es null entonces

    Retornar null

    Sino

    Retornar Raíz.obtenerMayorClave()

Fin

Nodo:

obtenerMayorClave()

Inicio

    Si hijoDer es null entonces

    Retornar this

    Sino

    Retornar hijoDer.obtenerMayorClave()

Fin

5. Análisis del orden del tiempo de ejecución

El mejor caso es cuando el árbol está balanceado y tengo, por ejemplo, 7 nodos pero recurro a la recursión solo 3 veces, siendo de orden O(log n).
El peor caso sería un árbol completamente degenerado hacia la derecha, y tendría un orden de ejecución O(n).

Obtener la clave inmediata anterior a una clave dada

1. Descripción en lenguaje natural

El algoritmo debe recorrer el árbol binario de búsqueda buscando el nodo cuya clave es la inmediatamente menor a la clave dada. Para ello, se recorre el árbol:

Si la clave actual es menor que la clave buscada, se recorre buscardo su hijo más a la derecha

Si la clave es igual o mayor se descarta.

2.

Precondiciones:

El árbol no está vacío.

La clave pasada por parámetro existe en el árbol

Es un árbol binario de búsqueda correctamente estructurado

Postcondición:

Se devuelve el nodo con la clave más grande dentro de las claves más pequeñas que la recibida por parametros

El árbol no se modifica.

Casos de prueba

Árbol con solo un nodo. Se retorna null

Árbol donde la clave está a buscar es la raíz

Árbol donde la clave es la más pequeña. Retornará null

Árbol balanceado. El algoritmo busca por el subárbol derecho del menor que la clave dada hasta encontrar el más cercano.

4. Seudocódigo

Árbol:

obtenerAnterior(clave)

Inicio

    Si raíz es null entonces

    retornar null

Sino

retornar raíz.obtenerAnterior(clave, null)

Fin

Nodo:

obtenerAnterior(clave, anterior)

Inicio

Si etiqueta es mayor o igual clave entonces

Si hijoIzq ≠ null entonces

retornar hijoIzq.obtenerAnterior(clave, anterior)

Sino

retornar anterior

Sino

anterior ← this

Si hijoDer ≠ null entonces

retornar hijoDer.obtenerAnterior(clave, anterior)

Sino

retornar anterior

Fin

5. Análisis del orden de ejecución

El algoritmo sigue la lógica de búsqueda binaria.

El mejor caso sería de O(log n) ya que hace una especie de búsqueda binaria. El peor caso sería con una especie de lista donde solo se tengan hijos desde la raíz de un tipo, solo izquierdos o solo derechos

Obtener la cantidad de Nodos de un nivel dado

1. Descripción en lenguaje natural

En la clase ArbolBB primero se verifica si está vació, si está vacío entonces retornará 0, ya que no hay ningún nodo. Si no está vació entonces se procede a llamar al método CantNodosNivel de TElementoABB. El método en esta clase recibe 2 parámetros, el nivel actual y el nivel a buscar. Comienza con un contador en 0.
Primero que nada revisa si el nivel actual coincide con el nivel a buscar, si es así entonces sumo 1 al contador.
Si el nodo actual tiene un hijo derecho entonces al contador le sumo la recursión del método cantNodosNivel del hijo derecho, y aumento en 1 el nivel actual. Hago lo mismo con el hijo izquierdo, verifico si no es nulo y hago recursión sumando 1 al nivel actual. por último cuando no tengo más nodos para recorrer retorno la cantidad de nodos encontrados en el nivel buscado.

2. Precondiciones:

No tiene

Postcondiciones: Si el árbol está vacío o si el nivel que se le da es mayor al nivel de árbol entonces devolverá cero. Sino devolverá la cantidad de nodos que hay en ese nivel indicado.

Casos de prueba

Árbol vacío. devuelve 0

Árbol con x niveles y se busca nivel x+1. devuelve 0

Árbol con raíz y 2 hijos, se busca nivel 1. devuelve 2

4. Seudocódigo

Árbol:

```
public int cantNodosNivel(int nivelBuscado)
        si (raiz esta vacia)
                devolver 0
        sino
                raiz.cantNodosNivel(0,nivelBuscado)
        FinSI
```

FIN

Nodo:

```
public int cantNodosNivel(int nivelactual, int nivelbuscar)
    int cantidad = 0
    si (nivelactual == nivelbuscar)
        cantidad++;
    FinSi
    si (hijoDer !no es nulo)
        cantidad+= hijoDer.cantNodosNivel(nivelactual+1, nivelbuscar);
    FinSi
    Si (hijoIzq no es nulo)
        cantidad+= hijoIzq.cantNodosNivel(nivelactual+1, nivelbuscar);
    FinSi
    devolver cantidad;
  FIN
```

5. Análisis del orden de ejecución

El algoritmo siempre va a recorrer todos los nodos del árbol, por lo que su orden de tiempo de ejecución será O(n).

Listar las hojas del árbol con su nivel

1. **Lenguaje Natural**

   Este método en Arbol no recibirá ningún parámetro, lo único que hará será revisar si el árbol está vacío, si es así entonces devolverá 0. Sino llamará al método listarHojasConNivel con la raíz y le dará como nivel inicial 0, ya que le nivel de la raís es 0.

   Este método el Nodo recibirá un parámetro (el del nivel actual). Primero revisará que el nodo actual sea un hoja, es decir que no tenga hijo izquierdo ni derecho, y lo imprimirá en consola diciendo su etiqueta y nivel actual. Por el contrario si el nodo

actual tiene un hijo Izquierdo entonces hará recursividad con el hijo izquierdo y aumentará el nivel actual en 1. Hará lo mismo con el hijo derecho.

2. Precondiciones:

No tiene

Postcondiciones:

Si el árbol está vació imprimirá “El árbol está vacío”. Sino imprimirá todas las hojas del árbol, cada una con su nivel correspondiente.

3.Seudocódigo:

Arbol:

```
public listarHojasConNivel():void
    si (no tiene raiz)
        Imprimir “El arbol esta vacio”
    Sino
        raiz.listarHojasNodoConNivel(0)
    FinSi
Fin
```

Nodo:

```
public void listarHojasConNivel(int nivelActual)
  Si (no tiene hijos)
    Imprimir("El nodo con la etiqueta " + this.etiqueta + "es hoja en el nivel " +
    nivelActual + ".")
  FinSi
  Si (hijoIzq no es nulo)
    hijoIzq.listarHojasConNivel(nivelActual+1)
  FinSi
  Si (hijoDer no es nulo)
```

```
        hijoDer.listarHojasConNivel(nivelActual+1);

    FinSi

Fin
```

5. Análisis del orden de ejecución

El algoritmo siempre va a recorrer todos los nodos del árbol en preorden para poder alcanzar todas las hojas, por lo que su orden de tiempo de ejecución siempre será O(n).

Verificar si es de búsqueda

1. **Lenguaje Natural**

Este algoritmo en la clase Árbol no recibirá ningún parámetro y retornará un booleano. Primero revisará si está vacío, si está vacío entonces retornará false. Sino creará una lista vacía, la cual enviará al método inOrderEtiquetas en Nodo, este método lo que hará será modificar esta lista e irá añadiendo las etiquetas de los nodos del árbol en inOrden. Primero revisará si el nodo actual tiene un hijo izquierdo, si es así entonces hará recursión con el mismo. Luego añadirá la etiqueta del nodo actual a la lista, y por último revisará si tiene un hijo derecho, si es así entonces hará recursión con el hijoDerecho.

Luego de haber recorrido todo el árbol, se recorrerá la lista con 2 auxiliares, i e i+1, en base a estos auxiliares se van a ir comparando los valores de la lista de a pares, si el i es mayor que el i+1 en algún momento entonces retornará falso, ya que en un árbol binario de búsqueda el hijo izquierdo de un nodo tiene que ser menor que el mismo. Pero si durante todo el recorrido de la lista se cumple que i<i+1 entonces retornará true, ya que cumple con la característica de los árboles binario de búsqueda.

2. Precondiciones:

El árbol no debe estar vacío

Postcondiciones:

Si está vacío entonces retornará falso. Si se cumple que el hijo izquierdo es menor que el nodo actual y que su hijo derecho es mayor que el actual entonces retornará true. Si no se cumple en un nodo entonces retornara false.

3.Seudocódigo:

Arbol:

```
public esBinarioDeBusqueda():boolean
        si (la raiz es nula)
                devolver falso
        Sino
                new ArrayList lista
                raiz.inOrderEtiquetas(lista)
                para cada etiqueta en la lista
                        si la etiqueta actual es mayor que etiquetasiguiente
                                devolver falso
                        FinSi
                FinPara
        FinSi
Fin
```

Nodo:

```
public inOrderEtiquetas(ArrayList<Comparable> elementos): void
      Si (hijoIzq  no es nulo)
         hijoIzq.inOrderEtiquetas(elementos);
      FinSi
      elementos.add(etiqueta);
      Si (hijoDer no es nulo)
         hijoDer.inOrderEtiquetas(elementos);
      FinSi
   Fin
```

5. Análisis del orden de ejecución

El algoritmo siempre va a recorrer todos los nodos del árbol en inOrden para enlistar todos los nodos del mismo, y luego recorrerá cada elemento de esa lista, la cual será del tamaño del árbol, por lo que hace O(n)+O(n), por ende su orden del tiempo de ejecución es O(n)