**Laboratorio S08. AYED-02**

Jefer Alexis González Romero

**DOCUMENTO TÉCNICO**

**Operaciones Update - Delete para la clase Binary Tree**

**Requisitos**

**Especificación**

arbol =

**6**

**9**

**2**

**4**

**7**

**3**

**15**

**18**

**17**

**20**

arbol.delete(18)

**6**

**9**

**2**

**4**

**7**

**3**

**15**

**20**

**17**

**Entrada:**

Se tiene un árbol y se recibe un elemento.

**Salida:**

El elemento es eliminado del árbol, cumpliendo las reglas de BST (Sin balance).

**Diseño**

**Estrategia**

A los nodos se le agregará el atributo “higher” el cual va a ser un apuntador a la raíz del subárbol que contiene al nodo (padre), junto a este habrá dos nuevas operaciones para leer get\_higher() y para actualizar set\_higher(new\_node).

Para eliminar un elemento se busca con la función search, si no está, no se hace nada y si al contrario si se encuentra, se evalúa dependiendo del caso que presente:

* El elemento por eliminar no tiene hijos (es una hoja), entonces al padre se le asigna al subárbol derecho o izquierdo (según donde se encontraba el elemento) None.
* Si el elemento tiene un hijo, al padre se le asigna en el subárbol que corresponda el hijo del
* El elemento tiene dos hijos, para este caso se crea una nueva función que busque el sucesor de un nodo, el cual va a ser asignado como subárbol del padre que tenía el elemento, y para el padre que era del sucesor, si es diferente al elemento a eliminar, entonces el subárbol de este se deja en None.

También está la posibilidad de que se elimine la raíz del árbol, en este se evalúan los mismos casos, solo que, como no tiene padre, entonces únicamente se debe hacer las asignaciones correspondientes con la nueva raíz del árbol.

Adjunto el programa con nombre “AVL”.

**Casos prueba**

Para los casos prueba se va a usar el siguiente árbol:

arbol =

**Imagen que contiene bola de billar

Descripción generada automáticamente**

|  |  |  |
| --- | --- | --- |
| Entrada | Justificación | Salida |
| arbol.delete(9) | El elemento por eliminar no tiene hijos |  |
| arbol.delete(6) | El elemento tiene dos hijos |  |
| arbol.delete(15) | El elemento es la raíz |  |
| arbol.delete(7) | El elemento tiene un hijo |  |

**Análisis**

**Temporal**

En el mejor de los casos T(n) = Ω(1)

En el peor de los casos T(n) = O(Ln(n))

**Código**

**Documentación**

Dentro del código

**DOCUMENTO TÉCNICO**

**Atributo de factor de balance AVL a la clase y refactorización de las operaciones**

**Requisitos**

**Especificación**

**-1**

**0**

**0**

**0**

**0**

**+1**

**0**

**0**

**0**

**6**

**2**

**4**

**7**

**3**

**15**

**18**

**17**

**20**

**0**

**9**

* Factor de balance

**Entrada:**

Se recibe un árbol binario de búsqueda

**Salida:**

Cada nodo con su respectivo factor de balance

**Diseño**

**Estrategia**

A los nodos se le agregará el atributo “balance\_factor”, junto a este habrá una nueva operacion para leer get\_balance\_factor() y una función para calcular el factor de balance de un nodo calculate\_balance\_factor(), esta función a su vez va acompañada de otra que será get\_height() la cual calcula la altura de un nodo.

Para que cada nodo tenga su factor de balance correspondiente, al momento de insertar los elementos del árbol, se va a ir calculando y asignando su correspondiente factor de balance.

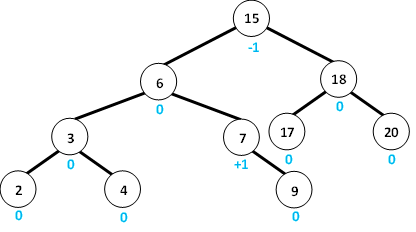
En la función de eliminar, los nodos a los cuales se modifica un subárbol se vuelven a calcular el factor de balance.

Adjunto el programa con nombre “AVL”.

**Casos prueba**

Para los casos prueba se va a usar el siguiente árbol:

arbol =



|  |  |  |
| --- | --- | --- |
| Entrada | Justificación | Salida |
| arbol.insert(10) | Función insertar |  |
| arbol.delete(6) | Función delete |  |

**Análisis**

**Temporal**

En el mejor de los casos T(n) = Ω(1)

En el peor de los casos T(n) = O(Ln(n))

**Código**

**Documentación**

Dentro del código

**Tercer punto**

La refactorización se encuentra dentro del código

**Cuarto punto**

Dentro del código al ejecutarlo se pueden ver la correctitud de las operaciones. La operación insert se usa al momento de crear el árbol binario, se puede ver como funciona el AVL, ya que al agregar el 9 el factor de balance de un nodo es 2.

**Quinto punto**

¿ Cómo reconstruir un árbol binario a partir de la salida generada por los recorridos vistos en clase ?

Para reconstruir el árbol solo se necesitarían dos recorridos, uno de ellos debe ser el InOrder y el otro puede ser PreOrder o PostOrder