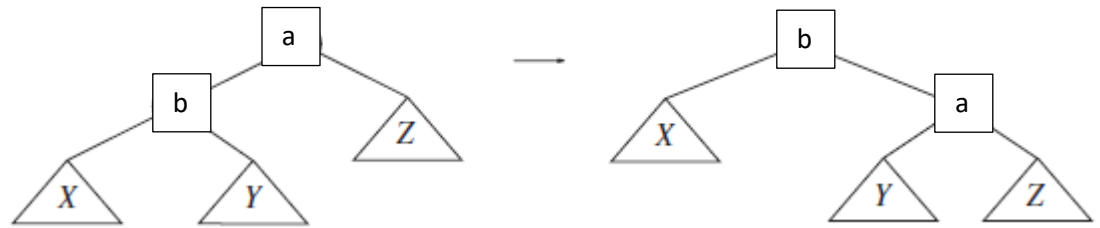


Práctica: AVL Trees

26 y 28 de octubre – Estructura de Datos

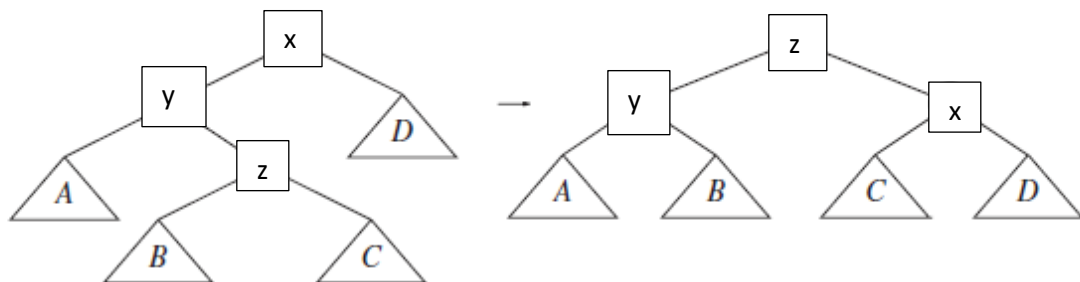
1. Ejercicio teórico: Ingresa los siguientes datos a un árbol AVL, dibuja el árbol cada vez que haya un ingreso, si se desequilibra, aplica la rotación adecuada, dibuja el árbol nuevamente e indica qué tipo de rotación hiciste y en qué nodo apareció el desequilibrio: 84, 10, 8, 92, 66, 88, 29, 27, 75, 72, 68, 62, 18, 80, 36, 1, 40.
2. **Crea la clase AVLTree**. Que deberá ser genérica. Utiliza la letra E como referencia a tipo de dato y no se te olvide que un AVL es un BST, para que limites con la característica de los datos que pueden guardarse en un AVL.
3. **Crea la clase privada Node**, que tendrá como variables de instancia, el elemento. referencia a su hijo izquierdo y al derecho, además de contar con la información de la altura de cada nodo.
 - a. Crea el **constructor de la clase Node**, que reciba, el elemento y los hijos derecho e izquierdo (no olvides inicializar la altura de manera adecuada).
 - b. Crea un método **toString de la clase Node** que imprima el elemento y la altura de ese nodo en el siguiente formato [elem – height]
4. Declara la variable de instancia de AVL, llámala root.
5. Agrega **private static final int ALLOWED_IMBALANCE = 1;**
6. Crea el constructor que inicie un árbol a vacío.
7. Copia y adapta a esta clase el **método inOrder()** de tu clase ABB. O el iterador. Para que tengas alguna manera de recorrer e imprimir el árbol.
8. Agrega el **método privado Node insert(E elem, Node n)** que agregue un nuevo elemento al árbol. Adapta el método privado visto en clase y crea también el **método público void insert (E element)**. Este método privado utilizará el método balance que aún no tenemos, lo agregaremos en el siguiente paso.
9. Agrega el **método privado int height(Node n)**, este método debe devolver 0 si n es nodo nulo o la correspondiente altura del nodo n.
10. Agrega el **método privado Node balance(Node n)**. Este método lo que hace es detectar si se ha roto la característica de un AVL (la diferencia de altura de sus subárboles, no debe ser mayor a 1) y en caso de ser necesario balancea. Además, este método actualiza la variable de instancia height del nodo. Este método ya lo tienes en tu presentación de clase. Este método llama a los métodos que hacen las rotaciones dobles y sencillas. Las cuales implementaremos más adelante.
11. Ahora empezaremos a agregar los métodos para las rotaciones simples y dobles necesarias. Crea el **método Node rotateWithLeftChild(Node a)**, este método realiza la rotación simple hacia la derecha con los siguientes pasos:
 - a. En un nodo b guardamos la referencia del izquierdo de a
 - b. El izquierdo de a, ahora debe referenciar al derecho de b.
 - c. El derecho de b, ahora debe referenciar a "a".
 - d. Corregir la altura de a, tomando la altura máxima entre su subárbol derecho e izquierdo y luego sumando uno.

- e. Corregir la altura de b, tomando la altura máxima entre su subárbol derecho e izquierdo y sumando uno.
- f. Devolver el nodo b



Weiss. M.A. *Data Structures and algorithm analysis in Java* (pag. 135).

12. Crea el método **Node rotateWithRightChild(Node a)**, este método realiza la rotación simple hacia la izquierda, recuerda que este método es un método simétricamente opuesto al anterior.
13. Crea el método **Node doubleWithLeftChild(Node x)**, este método realiza una rotación doble derecha-izquierda, de acuerdo a los siguientes pasos:
 - a. El nodo izquierdo de x, ahora referenciará al nodo obtenido al hacer una rotación hacia la derecha mandando como parámetro el hijo izquierdo de x (y en el dibujo).
 - b. Por último, devolvemos el nodo obtenido al hacer una rotación hacia la izquierda mandando como parámetro el nodo x.



Weiss. M.A. *Data Structures and algorithm analysis in Java* (pag. 135).

14. Crea el método **Node doubleWithRightNode(Node x)**, este método realiza una rotación doble izquierda-derecha, simétricamente opuesta al anterior.
15. Prueba tu clase con las inserciones del ejercicio del punto 1 y despliega un recorrido del árbol para que revises que todo esté correcto.
16. **Adicional:** Genera un método que te imprima el árbol por niveles y acomodado de tal manera que se visualice como estructura jerárquica.

Nota: Documenta tipo javadoc y anexa comentarios de funcionalidad a tus métodos. La clase sin comentarios no tendrá validez.