UNIVERSIDAD CATÓLICA DE SANTA MARÍA ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO 08:

ÁRBOL EQUILIBRADO AVL

I OBJETIVOS

- Analizar el comportamiento de los árboles equilibrados AVL.
- Implementar el TDA AVL.
- Aplicar herencia para la implementación del TDA.
- Identificar las diferencias entre un BST y un AVL.
- Aplicar el TDA AVL a la solución de problemas propuestos.

II TEMAS A TRATAR

- Definición de árbol AVL
- Representación de un árbol AVL
- Operaciones de reestructuración del Balance
- Operaciones de inserción y eliminación en un AVL

III DURACIÓN DE LA PRÁCTICA

- Dos sesiones (4 horas)

IV RECURSOS

- Equipos de cómputo.
- Lenguaje de programación Java y pseudocodigo.
- IDE de desarrollo. Eclipse, VCode.
- Herramientas de seguimiento de versiones: Git & GitHub.

MARCO TEÓRICO

Revisar previamente los Cap. 19 del libro de Weiss p 706.

Árbol

Un árbol es una estructura de datos jerárquica, que está compuesto por nodos, que están conectados entre sí, donde cada nodo tiene un valor y un conjunto de hijos. El nodo superior se llama raíz y los nodos sin hijos son llamados hojas, Weiss, M. (2010)..

Árbol binario

Un árbol binario es una estructura de datos en la que cada nodo tiene como máximo dos hijos, que se llaman hijo izquierdo e hijo derecho. En un árbol binario, el hijo izquierdo de un nodo tiene un valor menor que el nodo, mientras que el hijo derecho tiene un valor mayor, Weiss, M. (2010)..

Árbol binario perfectamente equilibrado

Un árbol binario perfectamente balanceado cumple con las siguientes condiciones:

- La diferencia en la cantidad de nodos entre los subárboles izquierdo y derecho de una raíz no excede de una unidad.
- La altura de los subárboles izquierdo y derecho es idéntica respecto a la raíz.
- Tanto los subárboles izquierdo como derecho de cualquier raíz son árboles binarios perfectamente equilibrados.

Árbol AVL

El nombre AVL proviene de los apellidos de sus creadores, Adelson, Velsky y Landis, quienes introdujeron el concepto en 1962. Según Weiss, un árbol AVL es un árbol binario de búsqueda auto equilibrado en el que la diferencia de alturas entre los subárboles izquierdo y derecho de cualquier nodo no es mayor que uno. Esto significa que un árbol AVL, o árbol balanceado por altura, cumple con las siguientes condiciones:

- La diferencia de altura entre los subárboles izquierdo y derecho de la raíz no debe ser mayor que 1.
- Los subárboles izquierdo y derecho de cualquier nodo deben ser árboles AVL.

Si la altura del subárbol izquierdo es mayor que la del derecho en un nodo X, se dice que X tiene una altura a la izquierda. Si las alturas son iguales, se considera que tiene una altura equilibrada. Si el subárbol derecho es mayor, se dice que X tiene una altura a la derecha.

El factor de equilibrio de un nodo X se define como la diferencia entre las alturas del subárbol derecho e izquierdo de X. Para mantener el balance, esta diferencia debe ser 1, 0 o -1. Si es mayor o menor que estos valores, el árbol debe ser balanceado nuevamente.

Representación de un Árbol AVL

Cada nodo de un árbol AVL puede representarse mediante una estructura similar a la de un árbol binario de búsqueda (BST), con algunos campos adicionales para almacenar el factor de equilibrio. Se puede crear la estructura con la siguiente declaración:

```
class Node<E> {
       private E data;
       private int bf; //factor de equilibrio o balance
       private Node<E> left;
       private Node<E> right;
                //métodos
}
```

Dado que un árbol AVL es un árbol binario de búsqueda (BST), los algoritmos utilizados son los mismos. Sin embargo, en las operaciones de inserción y eliminación de elementos, es necesario tener en cuenta que el resultado de estas acciones podría generar árboles que ya no cumplan con los criterios de balance, lo que requeriría realizar una reconstrucción del árbol.
En las siguientes figuras se ve como se produciría un árbol desbalanceado, después de la inserción**.**

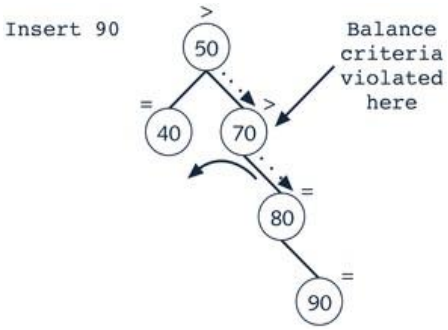


Figura 8.1. AVL después de la inserción del 90

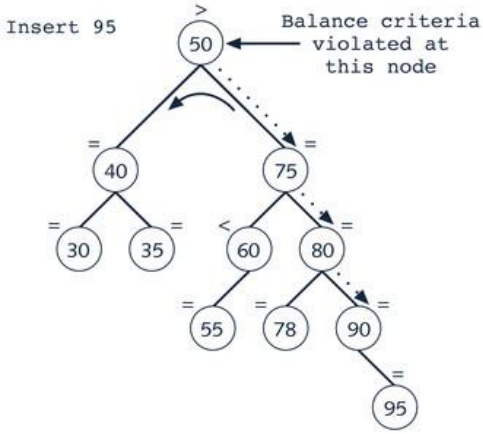


Figura 8.2. AVL después de la inserción del 95.

En las figuras 8.1 y 8.2, después de la inserción de los valores 90 y 95, se produce un desbalance en el árbol AVL debido a que la diferencia de alturas entre los subárboles izquierdo y derecho de la raíz supera el umbral permitido de 1. Este desbalance implica que se debe realizar una rotación para restaurar el equilibrio del árbol, garantizando así que las operaciones de búsqueda, inserción y eliminación mantengan su eficiencia.

OPERACIONES DE REESTRUCTURACIÓN DEL BALANCE

1. **ROTACIÓN SIMPLE IZQUIERDA (RSL):** Ciertos nodos del subárbol derecho de un nodo X se mueven al subárbol izquierdo, la raíz del subárbol derecho de X es la nueva raíz del árbol reconstruido.

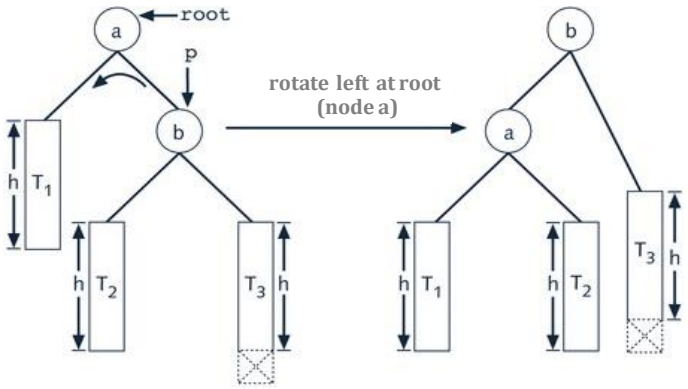


Figura 9.3. Rotación simple izquierda RSL en el nodo 'a'

En la figura 9.3, cuando el nodo **b** se encuentra en el subárbol derecho de la raíz **a**, se lleva a cabo las siguientes operaciones:

- El nodo **b** se convierte en la nueva raíz del árbol, y **a** se convierte en el hijo izquierdo de **b.**
- El subárbol izquierdo de **b** (denotado como T_2) se convierte en el subárbol derecho de **a**.
- Los subárboles T_1 y T_3 permanecen sin cambios en términos de su relación con **a** y **b**, pero sus alturas podrían ajustarse debido al cambio de estructura.

2. **ROTACIÓN SIMPLE DERECHA (RSR)**
   Es simétricamente similar a la anterior RSL. Es decir, Ciertos nodos del subárbol izquierdo de un nodo X se mueven al subárbol derecho. La raíz del subárbol izquierdo de X se convierte en la nueva raíz del árbol reconstruido.
   De esta forma, el nodo que estaba a la izquierda de la raíz original pasa a ser la nueva raíz del subárbol, mientras que la raíz original pasa a ser el hijo derecho de esta nueva raíz.

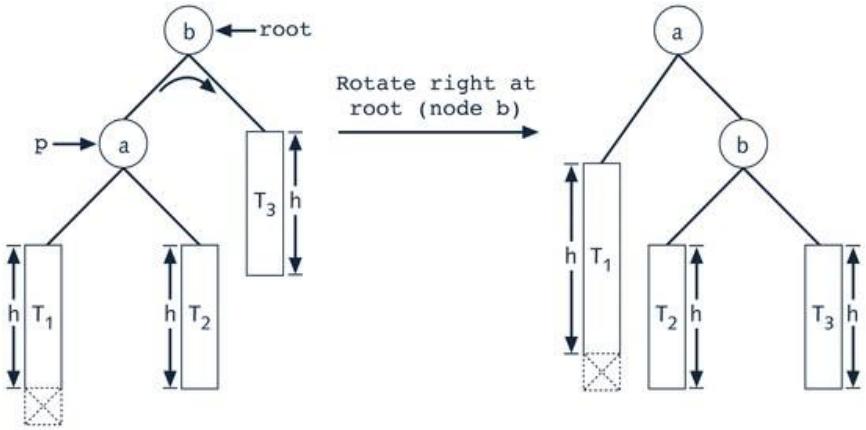


Figura 8.4. Rotación simple derecha en 'b'

3. **Rotación doble derecha (RDR):** se da cuando el subárbol izquierdo de un nodo X es dos unidades más grandes que el subárbol derecho, y el subárbol derecho del hijo izquierdo de X es más grande en una unidad que su lado izquierdo. Por tanto, hay que mover algunos nodos del lado izquierdo de X al derecho.
   Esta rotación se puede realizar directamente, o a través de dos rotaciones simples:
   - La primera es una RSL en el hijo izquierdo de X.
   - La segundo es una RSR en el nodo X.

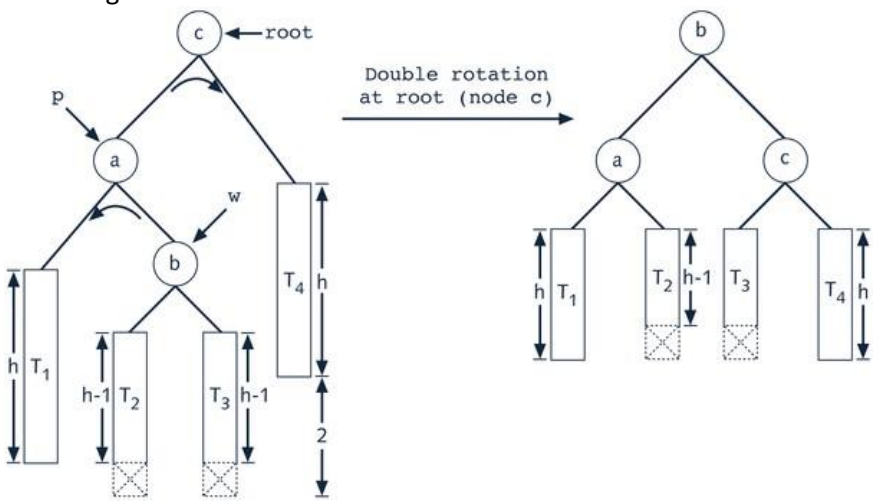


Figura 8.5. Rotación Doble Izquierda: 1ra RSL en 'a', luego RSR en 'c'

4. **Rotación doble izquierda (RDL):** es simétricamente similar a la anterior (RDR).

OPERACIÓN DE INSERCIÓN EN UN AVL

Antes de analizar los pasos que se deben seguir en el proceso de inserción, es necesario definir el concepto de "***Cambio de Altura en la inserción".***

• ¿Cuándo ocurre un cambio de altura en la inserción?

Se considera que ocurre un cambio de altura cuando, tras la inserción de un elemento, los nodos antecesores experimentan un cambio en su factor de equilibrio, pasando de 0 a 1 o -1. Esto indica que uno de los subárboles del nodo antecesor ***ha aumentado su altura*** en un nivel.

Si el factor de equilibrio de un nodo antecesor cambia de 1 o -1 a 0, se concluye que ***no ha habido un cambio de altura***. Esto significa que las alturas de los subárboles de dicho nodo han quedado equilibradas, sin ningún crecimiento en sus alturas.

Se usa el mismo procedimiento que se realiza para insertar un elemento en un BST:

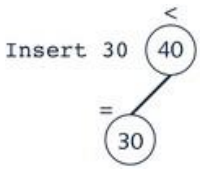
1. Se traza una ruta desde el nodo raíz hasta un nodo hoja (donde se realiza la inserción).
2. Se crea el nuevo nodo, se guarda el elemento en él y se enlaza al nodo padre.
3. A partir de este nodo nuevo, se debe retornar por la misma ruta de regreso al nodo raíz ajustando el factor de balance o equilibrio a lo largo de ella, siempre y cuando haya un cambio de altura. Si por el contrario se determina que no hay cambio de altura, se ha llegado a un punto en el que ya no es necesario revisar el balance del árbol, pues seguirá siendo AVL.
4. Cada vez que se retorne a un nodo antecesor se debe actualizar el factor de equilibrio:
   a) Si se sube por la izquierda, se disminuye en 1 el factor de equilibrio del nodo padre.
   b) Si se sube por la derecha, se incrementa en 1 el factor de equilibrio del nodo padre.
   c) Si luego de realizar algunas de las operaciones anteriores, el factor de equilibrio del padre cambia a +2 o -2, realizar alguna de las rotaciones según sea el caso.

Se vera un ejemplo en la Figura 8.6, en la que se ilustran las operaciones de rotaciones en sucesivas inserciones.

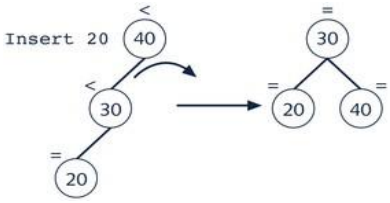
Se emplea el mismo procedimiento utilizado para insertar un elemento en un árbol binario de búsqueda (BST):

1. Se traza una ruta desde el nodo raíz hasta un nodo hoja, (donde se realizará la inserción).
2. Se crea el nuevo nodo, se coloca el elemento en él y se enlaza al nodo padre.
3. A partir de este nuevo nodo, **se regresa por la misma ruta hacia el nodo raíz**, ajustando el factor de equilibrio a lo largo del recorrido, siempre que haya un cambio de altura. En caso contrario, si no se detecta un cambio de altura, se concluye que no es necesario revisar el balance del árbol, ya que seguirá siendo un árbol AVL.
4. **Cada vez que se regresa a un nodo antecesor, se debe actualizar su factor de equilibrio:**
   a) Si se sube por el subárbol izquierdo, se disminuye en 1 el factor de equilibrio del nodo padre.
   b) Si se asciende por el subárbol derecho, se incrementa en 1 el factor de equilibrio del padre.
   c) Si, después de cualquiera de las operaciones anteriores, el factor de equilibrio del nodo padre cambia a +2 o -2, se debe realizar una rotación correspondiente, según el caso.

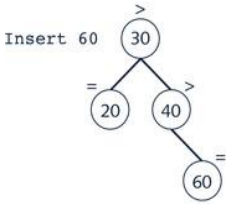
A continuación, se muestra un ejemplo en la Figura 8.6, que ilustra las operaciones de rotación en las sucesivas inserciones.



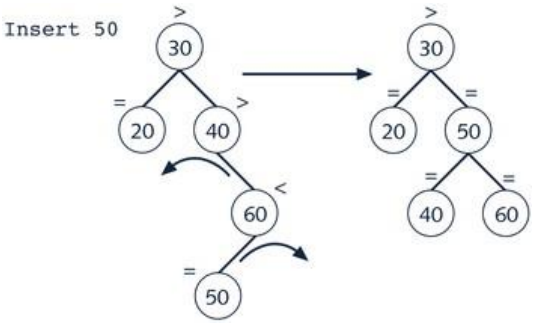
(a) Después de insert(30)



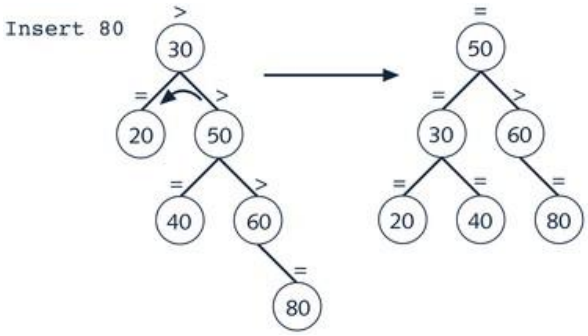
(b) Después de insert(20)



(c) Después de insert(60)

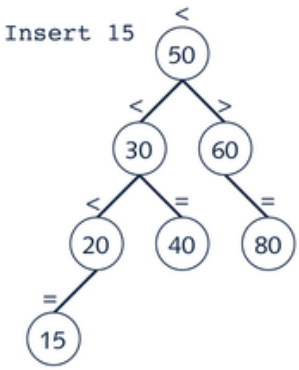


Después de insert(50)

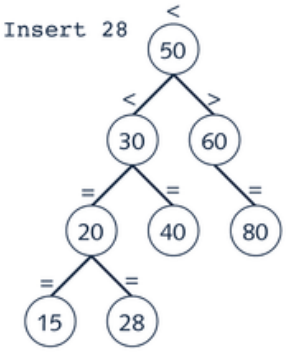


(d)

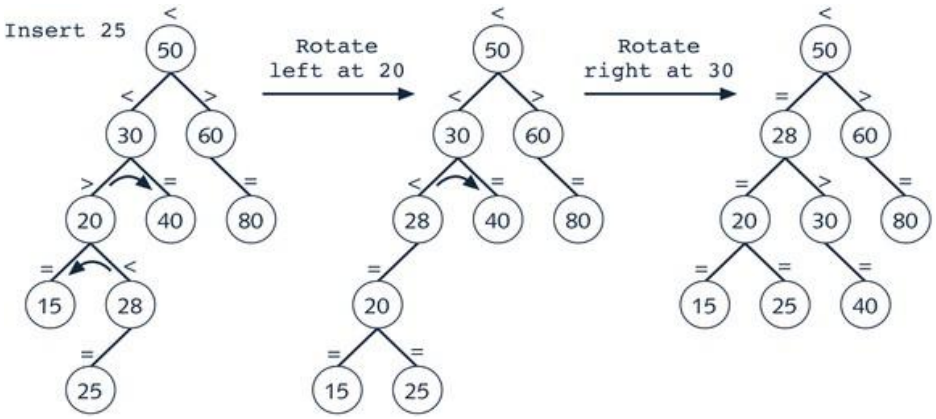
(e) Después de insert(80)



(f) Después de insert(15)



(g) Después de nsert(28)



(h) Después de insert(25)

Figura 8.6. Inserciones en un AVL

OPERACIÓN DE ELIMINACIÓN EN UN ÁRBOL AVL

Antes de analizar los pasos que deben realizarse en la eliminación, es necesario definir el concepto de "Cambio de Altura en la eliminación".

• ¿Cuándo ocurre un cambio de altura en la eliminación?

Se considera que ocurre un cambio de altura cuando, tras la eliminación de un elemento, los nodos antecesores experimentan un cambio en su factor de equilibrio, pasando de 1 o -1 a 0. Esto indica que uno de los subárboles del nodo antecesor ha disminuido su altura en un nivel, lo que puede generar un desbalance en nodos antecesores de niveles superiores.

Si el factor de equilibrio de un nodo antecesor cambia de 0 a -1 o 1, no se considera que haya un cambio de altura, lo que significa que la altura del subárbol con raíz en dicho nodo no ha variado, sino que uno de sus subárboles ha reducido su altura.

Para eliminar un nodo en un AVL, se utiliza el mismo procedimiento que para eliminar un nodo en un árbol binario de búsqueda (BST). Es necesario tener en cuenta ciertos escenarios específicos que requieren acciones particulares. Además, se debe haber trazado la ruta utilizada para descender en el árbol durante la búsqueda del elemento a eliminar.

Una vez eliminado el nodo, ya sea porque fue una hoja, tuvo un solo hijo o se eliminó su sucesor o antecesor en inorden debido a que tenía dos hijos, se deben seguir los siguientes pasos:

1. **Regresar por la ruta trazada en la búsqueda, ajustando el factor de equilibrio de los nodos antecesores a lo largo de la misma.** Este retorno solo debe llevarse a cabo si el ajuste del factor de equilibrio indica que ha ocurrido un cambio de altura.

   a) **Si se asciende por la izquierda**, se incrementa en 1 el factor de equilibrio del nodo padre.
   b) **Si se asciende por la derecha**, se decrementa en 1 el factor de equilibrio del nodo padre.
   c) **Si, después de realizar alguna de las operaciones anteriores, el factor de equilibrio del nodo padre cambia a +2 o -2**, se debe realizar una rotación correspondiente según el caso.
   d) **Es posible que, después de realizar alguna de las rotaciones, sea necesario realizar otra rotación.**

   Para verificar este caso, es necesario determinar el valor del factor de equilibrio del nodo que queda como raíz del subárbol reestructurado. Si se evidencia un cambio de altura, se debe realizar otra rotación.

Las figuras 8.7, 8.8 y 8.9 muestran el procedimiento de eliminación de algunos nodos y los ajustes que son necesarios realizar.

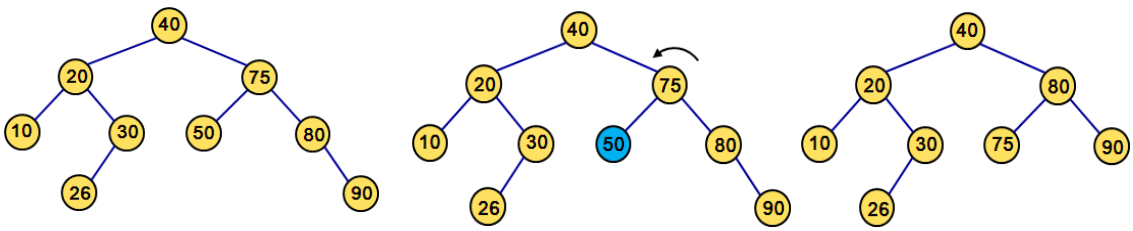


Figura 8.7. Eliminación del 50, provoca una rotación simple izquierda (RSL)

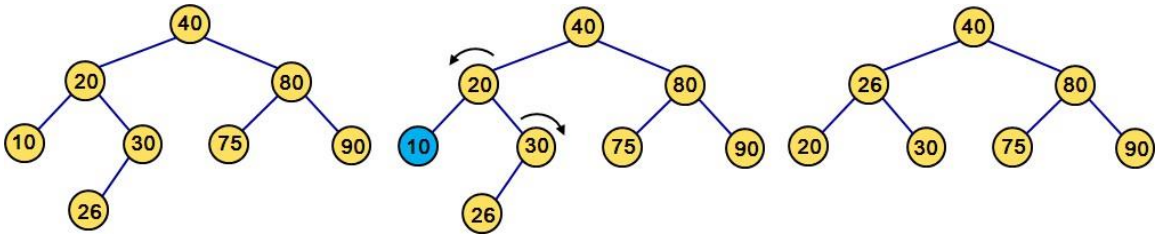


Figura 8.8. Eliminación del 10, provoca una rotación doble izquierda (RDL)

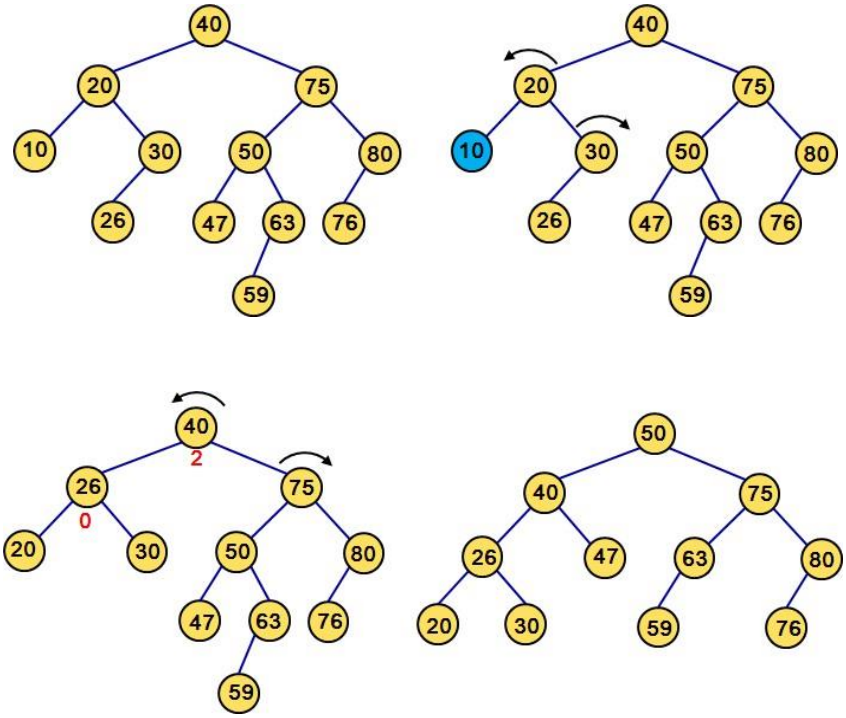


Figura 8.9. Eliminación del 10, provoca dos rotaciones: la primera una RDL en 20, luego otra RDL en 40

VI

ACTIVIDADES

Obtener el orden de complejidad de algoritmos iterativos

En una hoja realice los siguientes ejercicios:

1. En una hoja realice los siguientes ejercicios:

1.1. Inserte las siguientes claves en un árbol AVL inicialmente vacío: 30, 15, 20, 50, 40, 60, 70, 10, 25, 45, 55, 65, 75.

a) Muestre los árboles resultantes después de cada reestructuración.
b) Complete la siguiente tabla conforme vaya realizando el proceso de inserción, y en cada renglón indique:
    a. La clave que al ser insertada provoca un desbalance en el árbol (Inserción K).
    b. Este desbalance en qué nodo se genera (Nodo X).
    c. Cuál de las 4 rotaciones corresponde realizar para resolver el desbalance presentado (Rotación).
    d. Luego de realizar la rotación, ¿Cuál es el nodo que queda como raíz del subárbol cuya raíz era Nodo X? (Nodo Y)

Inserción K	Nodo X	Rotación	Nodo Y

1.2. En el árbol de la figura 8.10, elimine las siguientes claves: 12, 33, 46, 59, 45, 56. . De ser necesario utilice el sucesor en inorden.

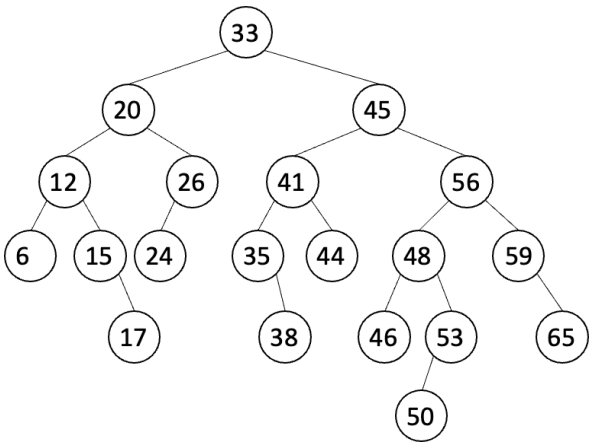


Figura 8.10. Árbol AVL para eliminación de elementos

a) Muestre los árboles resultantes de cada eliminación.

b) Complete la siguiente tabla conforme usted vaya realizando el proceso de eliminación, y en cada renglón indique:
   - Clave eliminada (**Eliminación K**)
   - Sabiendo que un AVL es un BST, a que caso de eliminación corresponde la eliminación de la clave K. (**Caso BST**)
   - Sólo si es el caso 3 de eliminación del BST, que nodo es su sucesor inorden. Sino fue este caso, deje la casilla en blanco. (**Sucesor**)
   - ¿La eliminación de esta clave genera desbalance en el árbol? Si/No (**¿Desbalanceo?**)
   - Solo si se genera desbalanceo indicar, en qué nodo se genera este desbalance (**Nodo X**)
   - Solo si se requiere alguna rotación, ¿Cuál de las 4 rotaciones corresponde realizar para resolver el desbalance presentado? (**Rotación**)
   - Solo si se hizo la rotación anterior, luego de realizarla, ¿Cuál es el nodo que queda como raíz del subarbol cuya raíz era Nodo X? (**Nodo Y**)
   - ¿Luego de hacer la rotación o no, se requiere otra rotación? Si/No (**¿Otra rotación?**). Si la respuesta es Si, complete el siguiente renglón desde la columna *Desbalance*o.

Eliminación K	Caso BST	Sucesor	Desbalanceo?	Nodo X	Rotación	Nodo Y	¿Otra rotación?

A partir de la actividad 2 debe evidenciar y explicar lo realizado en el informe correspondiente a este laboratorio.

2. Implementación del TAD AVL

En un nuevo proyecto, se debe incorporar el paquete de excepciones utilizado en los laboratorios anteriores y crear el paquete avltree.

Dado que un árbol AVL es un tipo de árbol binario de búsqueda (BST), la clase correspondiente al árbol AVL debe ser una subclase de *BSTree*. Además, los nodos en *BSTree* ya contienen atributos que serán utilizados en el árbol AVL, lo que permite aprovechar la herencia para reutilizar la implementación de los nodos de *BSTree*. En consecuencia, se deben modificar los atributos de la clase *Node* de *BSTree* para que su acceso sea protected.

2.1. **Creación de la clase AVL**

Ahora defina la clase AVLTree como una clase derivada de BSTree y haga lo propio con la clase NodeAVL en relación con Node. Además, agregue al nodo del AVL el factor de equilibrio (bf).

```
public class AVLTree<E extends Comparable<E>> extends BSTree<E>{
    class NodeAVL extends Node {
        protected int bf;

        public NodeAVL(E data) {
            //include your code here.
        }

        public String toString() {
            //include your code here.
        }
    }

    private boolean height;       //indicador de cambio de altura
}
```

- Complete el constructor del NodeAVL.
- Redefina el método toString del *NodeAVL*, de modo que al imprimir el contenido de un *NodeAVL*, se imprima el **dato** y el **factor de equilibrio** de ese nodo.

2.2. **Redefinir la inserción de la clase AVLTree**:

Una de las operaciones que se debe redefinir es la inserción, ya que se debe de verificar que el árbol siga siendo AVL luego de insertar una clave. El siguiente código es la implementación parcial de la operación de inserción.

```
public void insert(E x)  throws ItemDuplicated {
    this.height = false;
    this.root = insert(x, (NodeAVL)this.root);
}

protected Node insert(E x, NodeAVL node)throws ItemDuplicated{
    NodeAVL fat = node;

    if (node == null) {
        this.height =
        true; fat = new
        NodeAVL (x);
    }
    else {
        int resC = node.data.compareTo(x);
        if(resC == 0)throw new ItemDuplicated(x+" ya se encuentr
        en
                                                              el
                                                              arbol...");
        if(resC < 0) {
            fat.right = insert(x, (NodeAVL)node.right);
            if(this.height)
                switch(fat.bf) {
                case -1:    fat.bf = 0;
                            this.height = false;

                            break;
                case 0:
                            fat.b
                f = 1;
                            this.height = true;
                            break;
                case 1:     //bf = 2
                            fat =
                            balanceToLeft(fat);
                            this.height = false;
                            break;
                }
        }
        else {
            //include your code here.
        }
    }
    return fat;
}
```

- Analice el código anterior, y complete la parte cuando se debe insertar por la izquierda.

2.3. Implementación del método de soporte balanceToLeft de la clase AVLTree: El método **balanceToLeft()** se invoca cuando es necesario realizar una rotación a la izquierda, es decir, el subárbol con raíz en el nodo 'fat' esta crecido a la derecha, por lo tanto, es necesario mover nodos a la izquierda. En este nodo se determina que tipo de rotación a la izquierda debe realizarse. Además, una ajusta los factores de equilibrio para que tengan los valores adecuados después de la rotación.

```
private NodeAVL balanceToLeft(NodeAVL node){
    NodeAVL hijo = (NodeAVL)node.right;
    switch(hijo.bf) {
    case 1:
            node.bf = 0;
            hijo.bf = 0;
            node = rotateSL(node);
            b
    reak;
    case -1:
            NodeAVL nieto = (NodeAVL)hijo.left;
            switch(nieto.bf) {
            case -1: node.bf = 0; hijo.bf = 1; break;
            case 0:  node.bf = 0; hijo.bf = 0; break;
            case 1:  node.bf = 1; hijo.bf = 0; break;
```

```
                    }
                    nieto.bf =0;

                    node.right =
                    rotateSR(hijo); node =
                    rotateSL(node);
                }
                return node;
            }
```

- Implementar el método balanceToRight(). Observe que este método es simétricamente igual al método *balanceToLeft()*.

2.4. Implementación de los métodos de soporte para las rotaciones en la clase AVLTree:

- Ahora se debe de implementar los métodos que realicen las rotaciones. Dado que una rotación doble se puede realizar ejecutando las dos simples, definiremos las dos simples: a la derecha (RSR – rotateSR()) y a la izquierda (RSL – rotateSL()).

```
private NodeAVL rotateSL(NodeAVL node){
    NodeAVL p = (NodeAVL)node.right;
    node.right = p.left;
    p.left = node;
    node = p;
    return node;
}
```

- Implemente el método rotateSR(). Observe que este método es simétricamente igual al método *rotateSL()*.

2.5. Casos de pruebas:

En la clase TestAVL proponga mínimo 8 casos de prueba de inserción de elementos que se realicen en el mismo árbol, las cuales provoquen desequilibrios y se resuelvan aplicando por los menos 2 rotaciones de cada tipo: RSR, RSL, RDR, RDL. Evidencie en el informe lo realizado (capture pantallas) y explique cada prueba.

- Implemente el método rotateSR(). Observe que este método es simétricamente igual al método *rotateSL()*.

VII

EJERCICIOS

En el informe debe explicar cómo ha resuelto cada ejercicio, las pruebas realizadas y los resultados obtenidos, además de evidenciarlos a través de capturas de pantallas.

1. Considerando que la clase AVLTree hereda métodos de la clase BSTree, como search() y height(), se debe proponer un mínimo de 2 casos de prueba que permitan evidenciar las diferencias en el comportamiento entre un árbol binario de búsqueda (BST) y un árbol AVL.

2. En la clase AVLTree, se debe redefinir el método encargado de eliminar un elemento del árbol. Para ello, se debe analizar la posibilidad de reutilizar los métodos de soporte implementados durante la inserción. Es importante recordar que esta clase es una subclase de BSTree.

3. Crear una funcionalidad recursiva que permita realizar un recorrido por amplitud. Este recorrido tiene como objetivo visitar todos los nodos de un árbol por niveles: primero los del nivel 0, luego los del nivel 1 y así sucesivamente. Por ejemplo, para el árbol de la figura 8.11,

4. Implemente la funcionalidad recursiva que permita realizar un recorrido por amplitud (BFS). Este recorrido tiene como objetivo visitar todos los nodos de un árbol por niveles: primero los del nivel 0, luego los del nivel 1, y así sucesivamente.

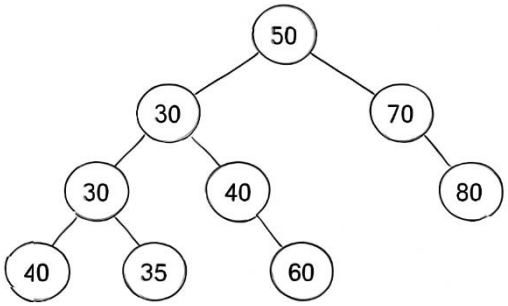


Figura 8.11. Árbol AVL ejemplo

Por ejemplo
Si Los nodos se visitarían en el siguiente orden: **50, 30, 70, 20, 40, 60, 80, 10, 25, 65**

5. Implementación de Recorrido en Preorden en un Árbol AVL

   Crear una funcionalidad recursiva para realizar un recorrido en preorden en un árbol AVL. En este recorrido, el orden es: primero el nodo raíz, luego los subárboles izquierdo y derecho. Realice pruebas con varios árboles AVL y evidencie los resultados con capturas de pantalla.

6. Insertar y Eliminar Nodos en un Árbol AVL con Casos de Rotación

   Realizar una secuencia de inserciones y eliminaciones en un árbol AVL y documente los casos en los que es necesario realizar rotaciones (simples o dobles) para equilibrar el árbol. Registre el árbol antes y después de cada rotación y complete una tabla indicando el tipo de rotación y el nodo raíz después de cada ajuste.

   Evidenciar el resultado de lo realizado en el informe.

VIII

REFERENCIAS

- Goodrich, M., Tamassia, R., & Goldwasser, M. (2014). Data Structures & Algorithms in Java. Wiley.
- Cutajar, J. (2018). Beginning Java Data Structures and Algorithms. Packt Publishing.
- Weiss, M. (2010). Data Structures & Problem Solving Using Java. Addison-Wesley.
- Knuth, D. E. (1976). Big O notation and asymptotic analysis. In The Art of Computer Programming (Vol. 1, pp. 10-15). Addison-Wesley.

RÚBRICA PARA LA CALIFICACIÓN DEL LABORATORIO

CRITERIO A SER EVALUADO		Nivel				
		A	B	C	D	NP
R1	Uso de estándares y buenas prácticas de programación	2.0	1.6	1.2	0.7	0.0
R2	Solución de las actividades a través de la definición y uso adecuado de .Arboles AVL, lo que se evidencia en el registro oportuno de los commits en los repositorios correspondientes, así como de las pruebas realizadas para su verificación.	4.0	3.2	2.4	1.4	0.0
R3	Solución de los ejercicios a través de la definición y uso adecuado de Arboles AVL lo que se evidencia en el registro oportuno de los commits en los repositorios correspondientes, así como de las pruebas realizadas para su verificación.	4.0	3.2	2.4	1.4	0.0
R4	Informe bien redactado, ordenado, bien detallado de las actividades y ejercicios resueltos, y de acuerdo a las normas establecidas. Mostrar evidencia suficiente de lo efectuado.	4.0	3.2	2.4	1.4	0.0
R5	Participación y contribución en el desarrollo del laboratorio del estudiante.	4.0	3.2	2.4	1.4	0.0
R6	Entrega oportuna de las actividades, ejercicios e informe.	2.0	1.6	1.2	0.7	0.0
Total		20				

A: Muy destacado

B: Destacado

C: Satisfactorio

D: Deficiente

NP: Muy deficiente o No presento