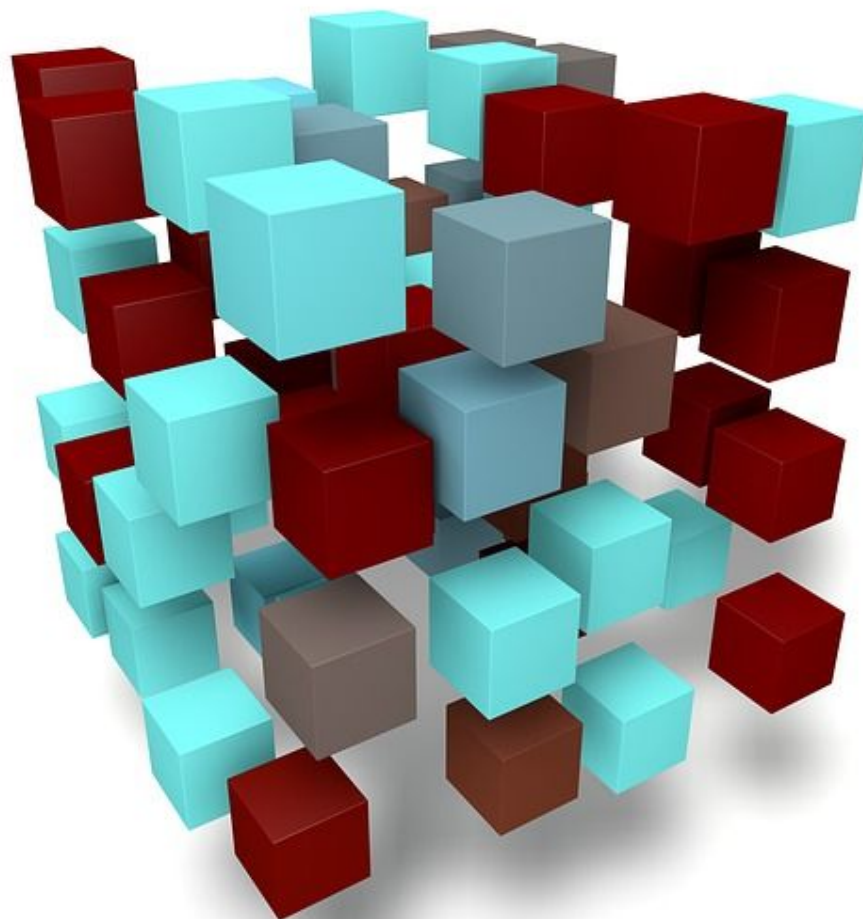


# Act \_Int\_3 - Actividad integradora de árboles binarios

Manuel Camacho Padilla [ A014123135 ]

25 / 10 / 2020



## ¿Qué tengo que hacer?

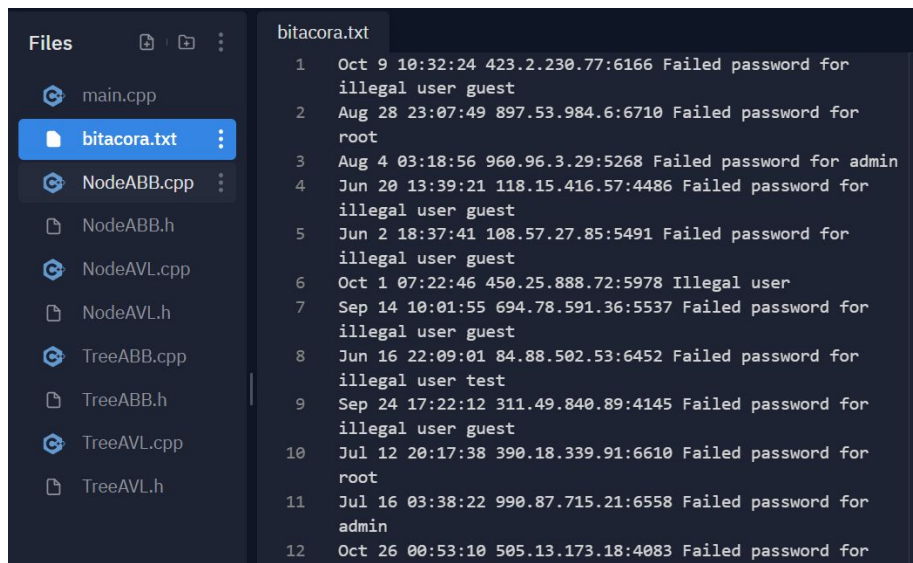
Una aplicación que realice la siguiente secuencia de operaciones utilizando tus DTA ABB y AVL. No construyas un menú.

1. Utiliza el archivo de entrada Bitacora.txt que utilizaste en la actividad integradora anterior.
2. Pregunta al usuario la cantidad de registros (N) del archivo que desea utilizar para construir las estructuras de datos. Valida que N sea un número positivo.
3. Construye un ABB y un AVL con los N registros del archivo utilizando la ip como dato del nodo.
4. Muestra los dos árboles construidos usando recorridos Preorden e Inorden.
5. Prueba borrar (1 nodo hoja, 1 nodo con 1 hijo y 1 nodo con dos hijos) en el ABB utilizando algunos de los datos insertados.

## Demostración

(Para los casos de prueba se imprime en pantalla las IP y los Factores de equilibrio para que sea más sencillo demostrar sus comportamientos)

- 1) Utiliza el archivo de entrada Bitacora.txt que utilizaste en la actividad integradora anterior.



```
Files    bitacora.txt
main.cpp
bitacora.txt
NodeABB.cpp
NodeABB.h
NodeAVL.cpp
NodeAVL.h
TreeABB.cpp
TreeABB.h
TreeAVL.cpp
TreeAVL.h

1 Oct 9 10:32:24 423.2.230.77:6166 Failed password for
  illegal user guest
2 Aug 28 23:07:49 897.53.984.6:6710 Failed password for
  root
3 Aug 4 03:18:56 960.96.3.29:5268 Failed password for admin
4 Jun 20 13:39:21 118.15.416.57:4486 Failed password for
  illegal user guest
5 Jun 2 18:37:41 108.57.27.85:5491 Failed password for
  illegal user guest
6 Oct 1 07:22:46 450.25.888.72:5978 Illegal user
7 Sep 14 10:01:55 694.78.591.36:5537 Failed password for
  illegal user guest
8 Jun 16 22:09:01 84.88.502.53:6452 Failed password for
  illegal user test
9 Sep 24 17:22:12 311.49.840.89:4145 Failed password for
  illegal user guest
10 Jul 12 20:17:38 390.18.339.91:6610 Failed password for
  root
11 Jul 16 03:38:22 990.87.715.21:6558 Failed password for
  admin
12 Oct 26 00:53:10 505.13.173.18:4083 Failed password for
  illegal user guest
```

- 2) Pregunta al usuario la cantidad de registros (N) del archivo que desea utilizar para construir las estructuras de datos. Valida que N sea un número positivo.

```
Digite cantidad de registros
(Recomiendo 10 para poder validar casos de pruebas)
n: -1

El numero tiene que ser positivo y diferente de cero.
❖
```

```
Digite cantidad de registros
(Recomiendo 10 para poder validar casos de pruebas)
n: 10

ARBOL ABB

Antes de borrar nodos.

Pre orden
IP: 423
```

- 3) Construye un ABB y un AVL con los N registros del archivo utilizando la ip como dato del nodo.

```
38 TreeABB readingABB(int n) {
39     /*
40     Función que lee los datos de un archivo de texto y los
      covertirá en una estructura arbol ABB y una estructura
      AVL.
41     Input: Un archivo de texto
42     Output: Un objeto tipo TreeABB
43     */
```

```
63 TreeAVL readingAVL(int n) {
64     /*
65     Función que lee los datos de un archivo de texto y los
      covertirá en una estructura arbol ABB y una estructura
      AVL.
66     Input: Un archivo de texto
67     Output: Un objeto tipo TreeABB
68     */
```

- 4) Muestra los dos árboles construidos usando recorridos Preorden e Inorden.

```
ARBOL ABB

Antes de borrar nodos.

Pre orden
IP: 423
IP: 118
IP: 108
IP: 84
IP: 311
IP: 390
IP: 897
IP: 450
IP: 694
IP: 960

En orden
IP: 84
IP: 108
IP: 118
IP: 311
IP: 390
IP: 423
IP: 450
IP: 694
IP: 897
IP: 960
```

```
ARBOL AVL

Pre orden
Dato: 423 Fe: 0
Dato: 108 Fe: 1
Dato: 84 Fe: 0
Dato: 311 Fe: 0
Dato: 118 Fe: 0
Dato: 390 Fe: 0
Dato: 897 Fe: -1
Dato: 450 Fe: 1
Dato: 694 Fe: 0
Dato: 960 Fe: 0

En orden
Dato: 84 Fe: 0
Dato: 108 Fe: 1
Dato: 118 Fe: 0
Dato: 311 Fe: 0
Dato: 390 Fe: 0
Dato: 423 Fe: 0
Dato: 450 Fe: 1
Dato: 694 Fe: 0
Dato: 897 Fe: -1
Dato: 960 Fe: 0
```

- 5) Prueba borrar (1 nodo hoja, 1 nodo con 1 hijo y 1 nodo con dos hijos) en el ABB utilizando algunos de los datos insertados.

```
removed[TRUE]: Se borra nodo que es hoja.
               Dato "84" eliminado.

removed[TRUE]: Se borra nodo con una sola rama.
               Dato "311" eliminado.

removed[TRUE]: Se borra nodo con dos ramas.
               Dato "423" eliminado.

Despues de borrar nodos.

Pre orden
IP: 450
IP: 118
IP: 108
IP: 390
IP: 897
IP: 694
IP: 960

En orden
IP: 108
IP: 118
IP: 390
IP: 450
IP: 694
IP: 897
IP: 960
```

## Investigación

### ¿Qué son las estructuras de datos?

En ciencias de la computación, una **estructura de datos** es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

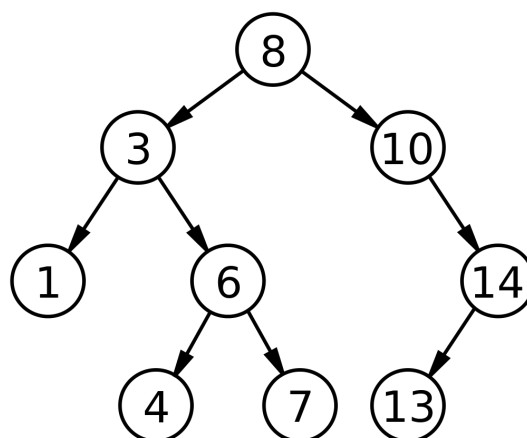
Las estructuras de datos son un medio para manejar grandes cantidades de datos de manera eficiente para usos tales como grandes bases de datos y servicios de indexación de Internet. Por lo general, las estructuras de datos eficientes son clave para diseñar algoritmos eficientes. Algunos métodos formales de diseño y lenguajes de programación destacan las estructuras de datos, en lugar de los algoritmos, como el factor clave de organización en el diseño de software.

Las estructuras de datos se basan generalmente en la capacidad de un ordenador para recuperar y almacenar datos en cualquier lugar de su memoria.

### Árbol ABB

Un **árbol binario de búsqueda** también llamado *BST* (*Binary Search Tree*) es un tipo particular de árbol binario que presenta una estructura de datos en forma de árbol usada en informática. **Su complejidad puede ser bien  $O(\log n)$  o  $O(n)$** , dependiendo de los casos.

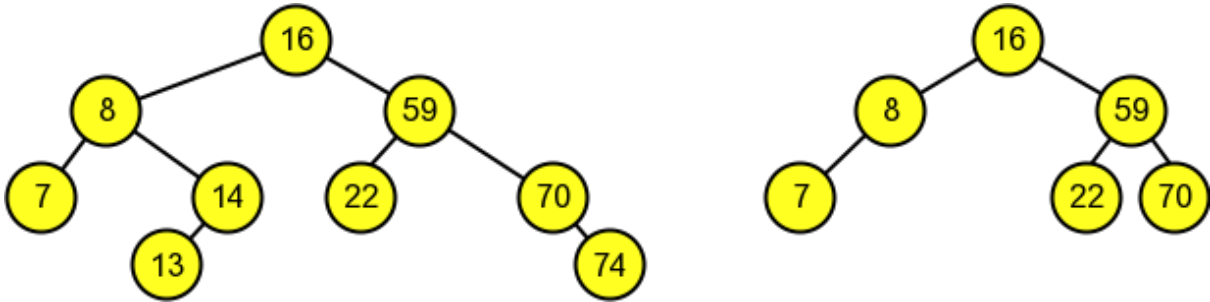
En resumen, es un árbol binario que cumple que el subárbol izquierdo de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el subárbol derecho (si no está vacío) contiene valores mayores.



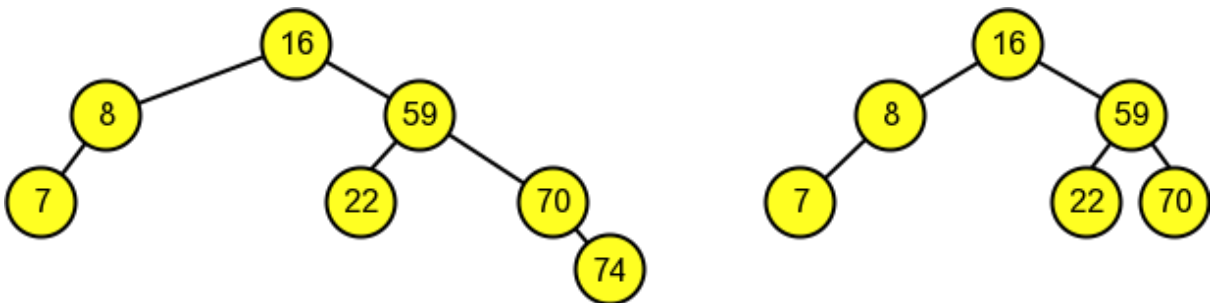
## El borrado de un nodo en Árbol ABB

La operación de borrado no es tan sencilla como las de búsqueda e inserción. Existen varios casos a tener en consideración (La complejidad es la misma al árbol ABB):

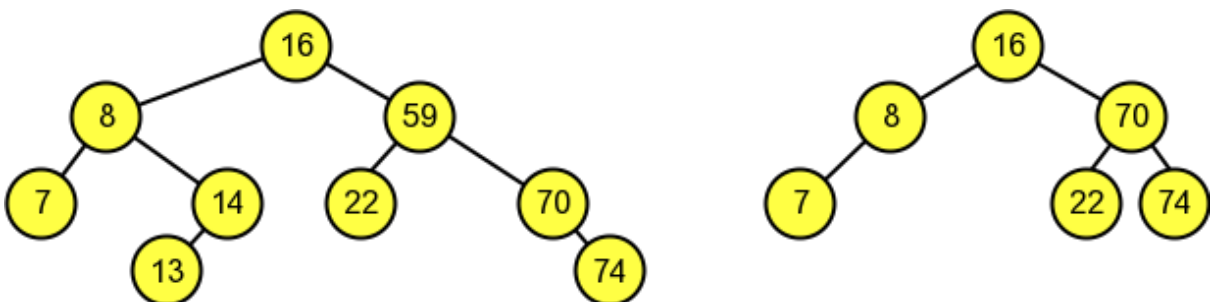
- **Borrar un nodo sin hijos o nodo hoja:** simplemente se borra y se establece a nulo el apuntador de su padre.



- **Borrar un nodo con un subárbol hijo:** se borra el nodo y se asigna su subárbol hijo como subárbol de su padre.



- **Borrar un nodo con dos subárboles hijo:** la solución está en reemplazar el valor del nodo por el de su predecesor o por el de su sucesor en *inorden* y posteriormente borrar este nodo. Su predecesor en *inorden* será el nodo más a la derecha de su subárbol izquierdo (mayor nodo del subárbol izquierdo), y su sucesor el nodo más a la izquierda de su subárbol derecho (menor nodo del subárbol derecho). En la siguiente figura se muestra cómo existe la posibilidad de realizar cualquiera de ambos reemplazos:



## Recorridos de un Árbol ABB y Árbol AVL

Se puede hacer un recorrido de un árbol en profundidad o en anchura.

Los recorridos en anchura son por niveles, se realiza horizontalmente desde la raíz a todos los hijos antes de pasar a la descendencia de alguno de los hijos.

El coste de recorrer el ABB o AVL es  $O(n)$ , ya que se necesitan visitar todos los vértices, **pero en nuestro método utilizamos un función recursiva, por lo que la complejidad se ve reducida a  $O(\log n)$** . El recorrido en profundidad lleva al camino desde la raíz hacia el descendiente más lejano del primer hijo y luego continúa con el siguiente hijo. Como recorridos en profundidad tenemos inorden, preorden y postorden.

## Árbol AVL

Un **árbol AVL** es un tipo especial de árbol binario ideado por los matemáticos rusos **Adelson-Velskii** y **Landis**. Fue el primer árbol de búsqueda binario auto-balanceable que se ideó. Sus operaciones pueden ser rotaciones a la Derecha, Izquierda, doble a la Derecha y doble a la Izquierda.

Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de **complejidad  $O(\log n)$** . El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

Para el factor de equilibrio, cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecho e izquierdo, igual que los árboles binarios de búsqueda (ABB), y además el dato que controla el factor de equilibrio.

El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo:

$$FE = \text{altura subárbol derecho} - \text{altura subárbol izquierdo}$$

Por definición, para un árbol AVL, este valor debe ser -1, 0 ó 1.

Si el factor de equilibrio de un nodo es:

0 -> el nodo está equilibrado y sus subárboles tienen exactamente la misma altura.

1 -> el nodo está equilibrado y su subárbol derecho es un nivel más alto.

-1 -> el nodo está equilibrado y su subárbol izquierdo es un nivel más alto.

Si el factor de equilibrio  $|FE| \geq 2$  es necesario re-equilibrar.



## Reflexión

Creo que cada actividad integradora lleva concordancia con el avance que el curso nos ofrece. Para mi gusto, disfruto mucho esta materia, y siento gratitud cuando seguimos un orden, ya que, de esto mismo van las estructuras de árbol, de llevar un orden. En lo particular me gustan tanto ABB como AVL, aunque el más óptimo sea el AVL, por mantener siempre un equilibrio perfecto, o casi perfecto  $(-1, 0, 1)$ .

Creo que es más cómodo de programar una estructura de árbol ABB, ya que la los nodos derechos e izquierdos siempre van a ser los mismos, al menos que borre, pero al momento de insertar cada nodo ya tiene un posición fija, pero si al momento de trabajar se requiere, entonces el árbol AVL, sería la mejor opción ya que reduce demasiado la complejidad comparada con el peor de los casos en un árbol ABB, el cual es de  $O(n)$ , pero en contraparte tiene un código más complejo ya que contiene un coeficiente de equilibrio que no es fácil de implementar, al menos no tanto como el del ABB.



## Referencias:

- NA. (2020). Estructura de datos. octubre 25, 2020 , de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/Estructura\\_de\\_datos](https://es.wikipedia.org/wiki/Estructura_de_datos)
- NA. (2020). Árbol binario de búsqueda. octubre 20, 2020 , de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/%C3%81rbol\\_binario\\_de\\_b%C3%BAsqueda](https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda)
- NA. (2020). Árbol AVL. octubre 20, 2020 , de Wikipedia Sitio web: [https://es.wikipedia.org/wiki/%C3%81rbol\\_AVL#:~:text=Un%20%C3%A1rbol%20AVL%20es%20un,auto%2Dbalanceable%20que%20se%20ide%C3%B3](https://es.wikipedia.org/wiki/%C3%81rbol_AVL#:~:text=Un%20%C3%A1rbol%20AVL%20es%20un,auto%2Dbalanceable%20que%20se%20ide%C3%B3)