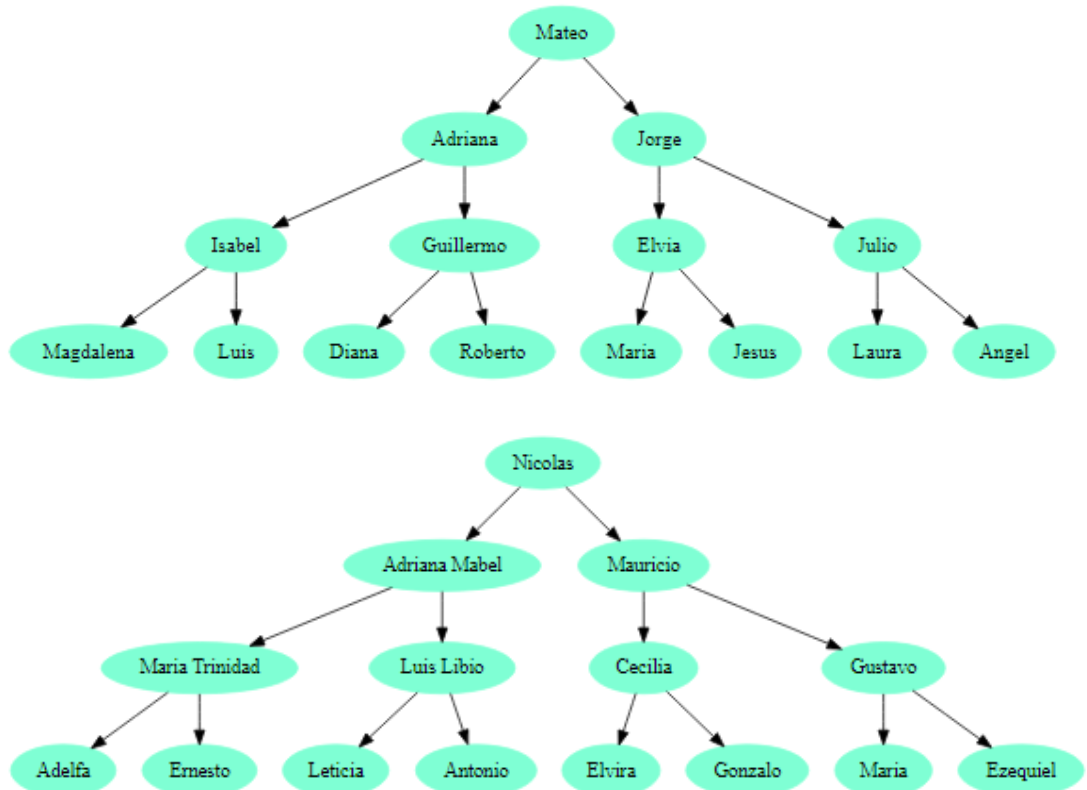


Laboratorio No. 5: Árboles Binarios

Nicolás Restrepo LópezUniversidad Eafit
Medellín, Colombia
nrestrepol@eafit.edu.co**Mateo Restrepo Sierra**Universidad Eafit
Medellín, Colombia
mrestrepos@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos.

1.1. Modelen sus ascendentes directos como un árbol binario utilizando la implementación en Java de árboles binarios.



Tomado de <http://www.webgraphviz.com>


Véase Anexos: *Node*, *BinarySearchTree*, *Punto1*.

2.1. Utilice las funciones buscar, tamaño, altura e imprimir recursivamente para verificar que el árbol quedó construido correctamente. Demuéstrelo en el main.

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

	UNIVERSIDAD EAFIT ESCUELA DE INGENIERÍA DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS	Código: ST0245
		Estructura de Datos y Algoritmos I

Véase Anexos: Node, BinarySearchTree, Punto1.

- 3.1.** Escriba un método para la clase BinarySearchTree que calcule quién es la abuela materna de una persona. No escriba el método main.

Véase Anexos: Node, BinarySearchTree.

- 2.1.** Resuelvan el ejercicio planteado usando árboles binarios.

Véase Anexos: Node, Punto2.

- 3.1.** Investiguen cuáles son los nombres de sus padres, los nombres de sus abuelos y los nombres de sus bisabuelos, y construyan sus árboles genealógicos.

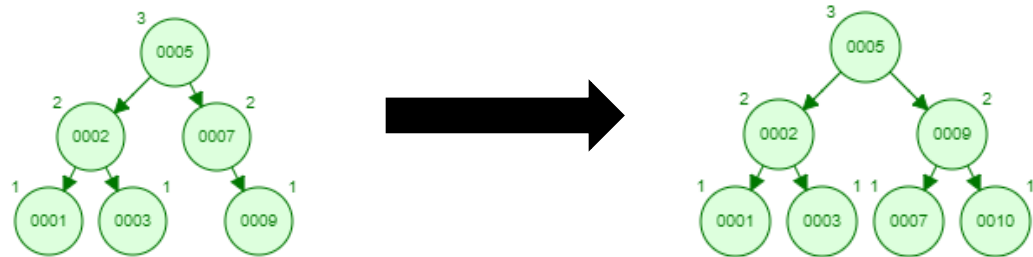
Véase Anexos: Node, BinarySearchTree, Punto1, Punto3.

Véase Ejercicio 1.1.

- 3.2.** ¿Se puede implementar más eficientemente un árbol genealógico para que la búsqueda e inserción se puedan hacer en tiempo logarítmico? ¿O no se puede?

Cuando se implementa un árbol de búsqueda binario se asume que sus complejidades serán $O(\log n)$ tomando como ideal que los elementos no serán introducidos en orden. No obstante, de insertarse en orden, se genera una Lista Enlazada y las complejidades se vuelven $O(n)$. Esto puede solucionarse mediante la implementación de árboles de búsqueda binarios de auto balanceo, como lo son los árboles tipo AVL y Rojo-Negro. La propiedad del auto balanceo les permite eliminar este problema, ya que el árbol se verá reordenado.

El árbol AVL establece que la diferencia entre la altura de los nodos hijos no puede ser mayor a 1. Por tal motivo, al superarse esta diferencia, se procede a modificar los nodos mediante un nuevo balance de los subárboles que se hace de manera recursiva hasta que la diferencia entre las alturas sea nuevamente menor o igual a 1. De esta manera se asegura que las complejidades serán siempre $O(\log n)$, en casos promedios y en los peores escenarios. En los siguientes gráficos, véase como se reorganiza el subárbol derecho cuando se inserta el número 10. Los índices que se muestran sobre cada nodo corresponden a los valores de balanceo.



Tomado de <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>

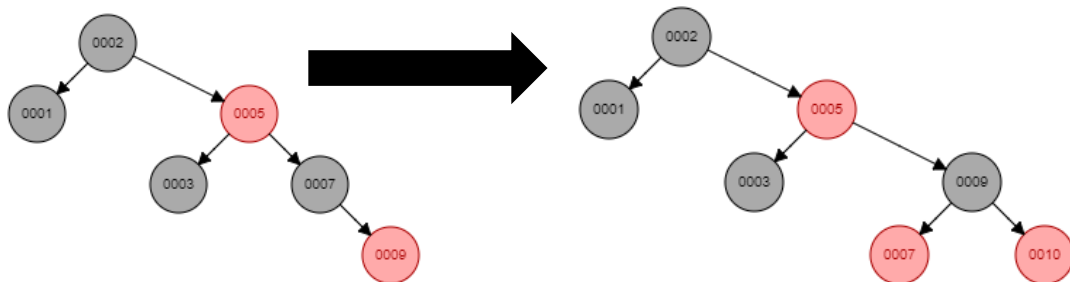
El árbol Rojo – Negro asigna dos posibles colores a cada nodo. Deben siempre cumplirse las siguientes reglas:

- Los hijos de un nodo rojo deben siempre ser negros.
- No puede haber dos nodos rojos contiguos, pero sí dos negros.
- Toda hoja negra es nula (Para algunas implementaciones).

Lo anterior permite que la altura del árbol sea como máximo $2\log(n+1)$, siendo n la cantidad de nodos. Además, de no cumplirse las propiedades anteriormente mencionadas al momento de insertar un nuevo nodo, se procede a realizar un balance, similar a lo que se hace en árboles AVL, de modo que se cambian los colores de los nodos de ser necesario.

Tomado de https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

En los siguientes gráficos, véase cómo se reorganiza el subárbol derecho cuando se inserta el número 10. Nótese el cambio de color de los nodos 7 y 9, ya que un nodo nuevo será rojo.



Tomado de <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

3.3. Expliquen con sus propias palabras cómo funcionan los ejercicios del numeral 2.1 y añádanla al PDF.

```
public class Node
{
    public Node left;           //C1
    public Node right;          //C2
    public String data;          //C3
    public int intData;          //C4

    public Node (String d)
    {
        data = d;               //C5
    }
    public Node (int d)
    {
        intData = d;            //C6
    }
}

public class Punto2
{
    Node root;                  //C7

    public Punto2 ()
    {
        root = null;            //C8
    }
    public void insertar (int n)
    {
        root = insertar(root, n); //O(Log n)
    }
    private Node insertar (Node nodo, int n)
    {
        if (nodo == null)        //C9
            return new Node(n);  //C1 + C2 + C4 + C6
        else if (nodo.intData < n) //C10 + C11
        {
            nodo.right = insertar(nodo.right, n); //C12 + T(n/2)
            return nodo;           //C13
        }
        nodo.left = insertar(nodo.left, n);      //C14 + T(n/2)
        return nodo;
    }
    public void posOrden()
    {
        posOrden(root);           //O(n)
    }
    private void posOrden(Node n)
    {
        if (n != null)            //C15
        {
            posOrden(n.left);      //C16 + T(n/2)
            posOrden(n.right);     //C17 + T(n/2)
            System.out.print(n.intData + " "); //C18
        }
    }
}
```

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

```
public static void main(String [] args)
{
    Punto2 ejemplo = new Punto2();                //C7 + C8

    ejemplo.insertar(50);                          //O(Log n)
    ejemplo.insertar(30);                          //O(Log n)
    ejemplo.insertar(24);                          //O(Log n)
    ejemplo.insertar(5);                           //O(Log n)
    ejemplo.insertar(28);                          //O(Log n)
    ejemplo.insertar(45);                          //O(Log n)
    ejemplo.insertar(98);                          //O(Log n)
    ejemplo.insertar(52);                          //O(Log n)
    ejemplo.insertar(60);                          //O(Log n)

    ejemplo.posOrden();                            //O(n)
}
```

El código consta de dos clases: Node y Punto2. La primera clase establece los nodos que van a componer el árbol. Esta clase tiene dos métodos constructores con una cadena de caracteres y un número entero. El primer constructor no tiene relevancia ya que se implementa para los árboles genealógicos. En este caso utilizamos el segundo constructor, ya que el árbol a implementar implementará de números enteros.

La clase Punto2 implementa la clase Node, así como un árbol de búsqueda binario. No obstante, para efectos del ejercicio solo se incorporaron los métodos de inserción e impresión pos orden. Esta clase solo posee una variable de instancia, que es el nodo raíz. Luego, el método constructor se encarga de hacerlo nulo.

A continuación, están dos métodos de inserción. El primero es el método principal que recibe como parámetro un número entero que va a ser insertado, y no hace más que llamar al segundo método de tipo privado. Este otro método es recursivo, y recibe como parámetros el número a insertar, así como el nodo en el cual comenzará el proceso. Se entra en un condicional, en el cual, si el nodo es nulo, se creará un nuevo nodo en dicha posición con el entero que se ingresó como dato. De lo contrario, si el número ingresado es mayor al dato del nodo actual, se procede a llamar recursivamente al método desde el nodo hijo derecho. Si ninguno de estos dos casos se lleva a cabo, se llama recursivamente al método desde el nodo hijo izquierdo.

Luego, están los dos métodos posOrden que imprimen los nodos del árbol de manera recursiva. El primer método es el principal, que llama al segundo método de tipo privado. Este segundo método recibe como parámetro el nodo desde el cual comenzará el proceso. Este tiene un condicional que indica que,

si el nodo es nulo, se procede a llamar recursivamente al método desde el hijo izquierdo, luego e derecho y finalmente se imprime el dato del nodo actual.

El método restante es el *Main*, que crea una instancia de la clase Punto2 y procede a insertar los números que allí se muestran. Al final, llama al método posOrden para imprimir el árbol en el recorrido Izquierda – Derecha – Raíz.

3.4. Calculen la complejidad del ejercicio realizado en el numeral 2.1 y agréguenla al informe PDF.

En el punto anterior tenemos ya el algoritmo con sus complejidades comentadas, así que simplemente sumaremos las complejidades del método principal. Obtenemos lo siguiente:

$$\begin{aligned}T(n) &= C_7 + C_8 + 9 * O(\log n) + O(n) \\T(n) &= C + C_1 * O(\log n) + O(n) \\T(n) &= C_1 * O(\log n) + O(n), \text{ por Regla de la Suma.} \\T(n) &= O(\log n) + O(n), \text{ por Regla del Producto.} \\T(n) &= O(n), \text{ por Regla de la Suma.}\end{aligned}$$

De este modo tenemos que la complejidad asintótica del ejercicio 2.1 es del orden de $O(n)$.

3.5. Expliquen con sus palabras qué son las variables m y n del cálculo de complejidad del numeral 3.4.

El cálculo de la complejidad solo posee una variable, que es n . Dicha variable corresponde a la cantidad de nodos que se encuentran en el árbol. Los demás términos corresponden a valores que permanecen constantes, y poco influyen en la complejidad asintótica del algoritmo cuando se trata de valores pequeños de n . Es por ello que no son tenidos en cuenta. Para valores grandes ya pueden incidir de manera considerable.

4) Simulacro de Parcial.

1. El nodo de un árbol binario se define así:

```
01 class Nodo {
02     Nodo izq;
03     Nodo der;
04     int dato;
05 }
```

Kefo ha dañado a Dayla su código para determinar cuál es la altura máxima de un árbol binario. No recuerda como lo había hecho y te pide ayuda para que lo completes.

```
01 int altura (Nodo raiz){  
02 if (raiz == null)  
03 return 0;  
04 int izq = _____;  
05 int der = _____;  
06 return Math.max(izq, der);}
```

a) Complete el espacio en línea 04.

altura(raiz.izq)

b) Complete el espacio en línea 05.

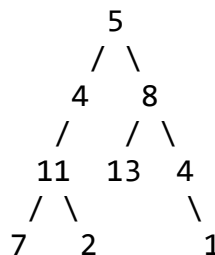
altura(raiz.der)

2. A un árbol binario de búsqueda se le ingresan 9 elementos en este orden: 12, 3, 5, 8, 2, 1, 9, 17. ¿Cuántos nodos hay que recorrer antes de encontrar el número 8? Se cuenta la raíz, pero no se cuenta el nodo donde está el 8.

- a) 2
- b) 1
- c) 3**
- d) 5

3. Definimos el camino desde la raíz hasta una hoja en un árbol binario como una secuencia de nodos empezando en el nodo raíz y bajando hasta una hoja. Una hoja es un nodo que no tiene hijos. Decimos que un árbol vacío no contiene caminos desde la raíz hasta una hoja.

Como un ejemplo, el siguiente árbol tiene 4 caminos desde la raíz hasta una hoja:



Caminos desde la raíz hasta una hoja:

- Camino 1: 5 4 11 7
- Camino 2: 5 4 11 2
- Camino 3: 5 8 13
- Camino 4: 5 8 4 1

Para este problema nos interesan las sumas de los elementos en los nodos de esos caminos, por ejemplo, la suma del camino [5,4,11,7] es 27.

Dada la raíz de un árbol binario `Nodo a` y un entero `int suma`, decir si existe un camino desde la raíz hasta una hoja tal que al sumar los valores de los nodos de ese camino la suma sea igual al parámetro `suma`.

Retorne falso si no se puede encontrar un camino con esa condición. Utilice la definición de `Nodo` del punto 1.

Desafortunadamente, al código que hizo Dayla le faltan unas líneas y Kefo está de vacaciones.

```
01 boolean sumaElCamino(Nodo a, int suma) {  
02 if (a == null)  
03 return _____;  
04 if (a.izq == null && a.der == null)  
05 return suma == _____;  
06 else  
07 return sumaElCamino(_____, _____)  
08 || sumaElCamino(_____, _____); }
```

a) Complete el espacio de la línea 03

return false

b) Complete el espacio de la línea 05

return a.dato

c) Complete los espacios de la línea 07

a.der, suma - a.dato

d) Complete los espacios de la línea 08

a.izq, suma - a.dato

4. Considere la siguiente definición de árbol binario:

```
01 class Node {  
02 public Node left;  
03 public Node right;  
04 public String data;  
05 public Node(String d) {  
06 data = d; }  
07 }
```

El siguiente algoritmo imprime todos los valores de un árbol en pre orden.

```
01 private void printAUX(Node node) {  
02 if (node != null) {  
03 System.out.println(node.data);  
04 printAUX(node.left);  
05 printAUX(node.right);  
06 }  
07 }  
08 public boolean print() {  
09 printAUX(root);  
10 }
```

4.1. ¿Cuál ecuación de recurrencia que describe el número de instrucciones que ejecuta el algoritmo print en el peor de los casos? La variable n representa el número de elementos del árbol.

- a) $T(n) = T(n-1) + C$
- b) $T(n) = 2 * T(n-1) + C$
- c) $T(n) = 2 * T(n/2) + C$
- d) $T(n) = T(n/2) + C$
- e) $T(n) = T(n+1) + C$

4.2. ¿Cuál es su complejidad asintótica en el peor de los casos del algoritmo print? La variable n representa el número de elementos del árbol.

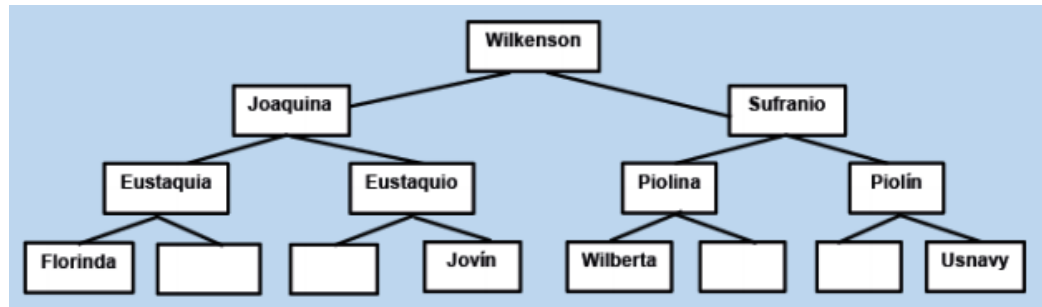
- a) $O(n)$
- b) $O(n^2)$
- c) $O(\log n)$
- d) $O(n * m)$
- e) $O(1)$

4.3. ¿Cuál es la salida del algoritmo print para el siguiente árbol?

Tenga en cuenta que la raíz es Wilkenson. Como Wilkenson es la raíz, para poder entender el árbol, recomendamos rotarlo 180 grados.

Después de hacer la rotación, usted entenderá que realmente las mujeres van a la izquierda y los hombres a la derecha.

Tenga en cuenta que, en un árbol genealógico, para algunas personas, no se conoce la mamá o el papá.



- a) Wilkenson, Sufranio, Piolín, Usnavy, Piolina, Wilberta, Joaquina, Eustaquio, Florinda, Eustaquia, Yovín.
- b) Sufranio, Piolina, Wilberta, Piolín, Usnavy, Joaquina, Estaquia, Florinda, Wilkenson, Yovín, Eustaquio.
- c) Wilkenson, Yovín, Eustaquio, Sufranio, Piolina, Wilberta, Piolín, Usnavy, Joaquina, Estaquia, Florinda.
- d) Wilkenson, Joaquina, Estaquia, Florinda, Yovín, Eustaquio, Sufranio, Piolina, Wilberta, Piolín, Usnavy.**
- e) Sufranio, Piolina, Wilberta, Piolín, Usnavy, Florinda, Wilkenson, Yovín, Eustaquio, Joaquina, Estaquia.

4.4. ¿Qué modificación hay que hacer algoritmo print para que arroje la siguiente respuesta para el árbol anterior?

Usnavy, Piolín, Wilberta, Piolina, Sufranio, Florinda, Eustaquio, Yovín, Eustaquia, Joaquina, Wilkenson.

Tenga en cuenta que la raíz es Wilkenson. Como Wilkenson es la raíz, para poder entender el árbol, recomendamos rotarlo 180 grados.

Después de hacer la rotación, usted entenderá que realmente las mujeres van a la izquierda y los hombres a la derecha.

Tenga en cuenta que, en un árbol genealógico, para algunas personas, no se conoce la mamá o el papá.

- a) Cambiar el orden de las líneas 03, 04 y 05 por 05, 04, 03.
- b) Cambiar el orden de las líneas 03, 04 y 05 por 04, 05, 03.
- c) Cambiar el orden de las líneas 03, 04 y 05 por 03, 05, 04.
- d) Cambiar el orden de las líneas 03, 04 por 06, 07.
- e) Intercambiar la línea 03 y la línea 04.

5. Luis escribió un programa para insertar un número en un árbol binario de búsqueda. En dicho árbol, él quiere tener los números menores o iguales a la raíz a la derecha y los mayores a la raíz a la izquierda.

Ayúdele a completar su código. El algoritmo recibe la raíz de un árbol p y un número a insertar $toInsert$, y retorna la raíz del árbol con el elemento insertado donde corresponde.

```
01 private Node insert(Node p, int toInsert){
02 if (p == null)
03 return new Node(toInsert);
04 if (_____)
05 return p;
06 if (_____)
07 p.left = insert(p.left, toInsert);
08 else
09 p.right = insert(p.right, toInsert);
10 return p;
11 }
```

- a) Complete, por favor, la línea 04 con la condición que corresponde.

$toInsert == p.dato$

- b) Complete, por favor, la línea 06 con la condición que corresponde.

$toInsert > p.dato$