UNIVERSIDAD CATÓLICA DE SANTA MARÍA
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO 08:

BINARY SEARCH TREE - BST

I
OBJETIVOS

- Analizar el comportamiento de los árboles binarios de búsqueda
- Analizar las diferentes formas de representar los árboles binarios de búsqueda
- Implementar recorridos en los árboles binarios: inorden, preorden y postorden
- Implementar las operaciones de los árboles binarios de búsqueda

II
TEMAS A TRATAR

- Definición de árbol binario
- Representación de un árbol binario
- Recorridos de un árbol binario
- Árbol de búsqueda binario (BST)
- Operaciones de los BST
- Aplicaciones de árboles binarios
- Ejercicios

III
MARCO TEÓRICO

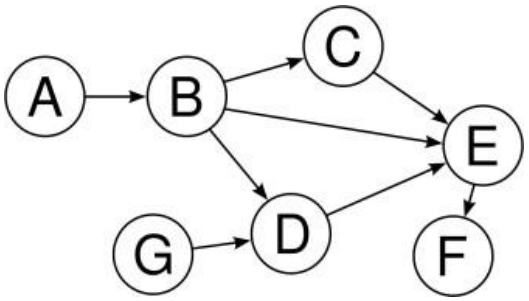
INTRODUCCIÓN

Los arreglos y las listas son ejemplos de estructuras de datos lineales. Cada elemento tiene:

- un predecesor único (excepto el primer elemento de la lista);
- un sucesor único (excepto el último elemento de la lista).

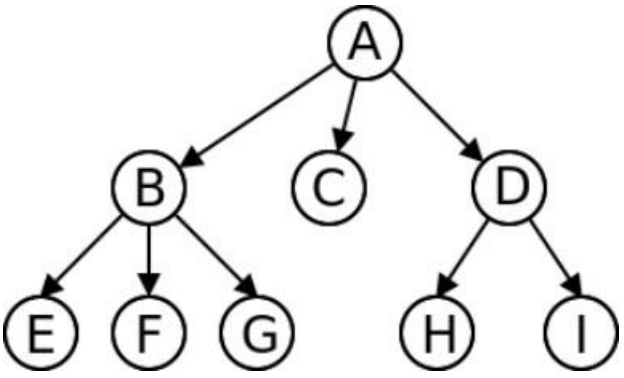
¿Existen otros tipos de estructuras?

Un grafo es una estructura de datos no lineal, ya que cada uno de los sus elementos, designados por nosotros, pueden tener más de una predecesor o más de un sucesor.



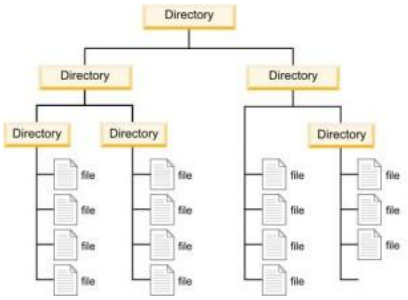
ÁRBOLES

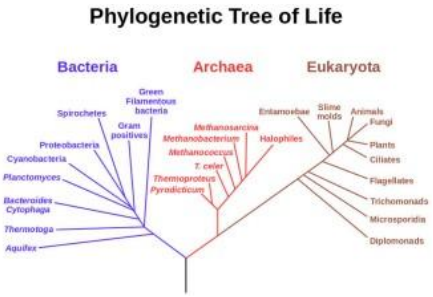
Un árbol es un tipo específico de grafo, cada elemento, designado por nodo, tiene cero o más sucesores, pero sólo un predecesor (excepto el primer nodo, que se llama raíz del árbol);
Un ejemplo de un árbol:



Los árboles son estructuras especialmente adecuadas para representar información organizada en jerarquías. Algunos ejemplos:

- La estructura de directorios (o carpetas) de un sistema de archivos
- Un árbol genealógico de una familia
- Un árbol de la vida





Qué es un Árbol Binario?

Un árbol binario es un tipo especial de árbol, en el cual ningún nodo de un árbol puede tener más de 2 nodos. Puede definirse como:

- Un árbol binario T, es una estructura vacía o
- T tiene un nodo especial llamado root (raiz)
- T tiene al menos uno de dos nodos, llamados subárbol izquierdo (Left Tree - LT) y subárbol derecho (Righ Tree - RT).

  - LT y RT son arboles binarios al mismo tiempo

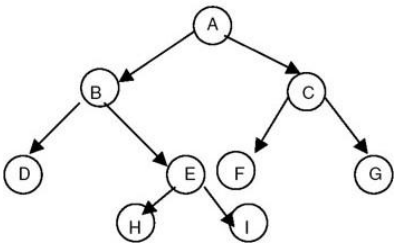
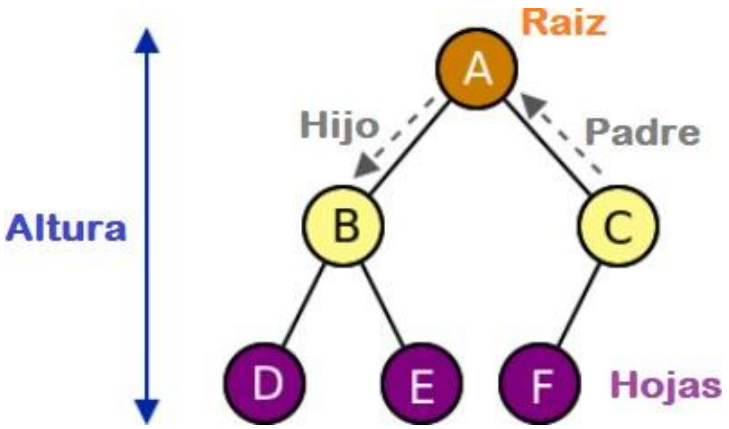


Figura 8.1. árbol binario

Terminología



- El predecesor (único) de un nodo se llama **padre**. Ejemplo: El padre de B es A; El padre de C también es A.
- Los sucesores de un nodo son sus **hijos**.Ejemplo: Los hijos de A son B y C
- El **grado** de un nodo es su número de hijos.Ejemplo: A tiene 2 hijos, C tiene 1 hijo
- Una **hoja** es un nodo sin hijos, es decir, de grado 0.Ejemplo: D, E y F son nodos hoja
- La **raíz** es el único nodo sin padre
- Un **subárbol** es un subconjunto de los nodos (vinculados) del árbol. Ejemplo: {B,D,E} son un subárbol
- Los arcos que conectan los nodos se llaman **ramas**.
- La secuencia de ramas entre dos nodos se llama **camino**.Ejemplo: A-B-D es el camino entre A y D

- La **longitud** de un camino es el número de ramas que contiene;Ejemplo: A-B-D tiene longitud 2
- La **profundidad** de un nodo es la longitud del camino desde la raíz hastaeste nodo (la profundidad de la raíz es cero);
  Ejemplo: B tiene profundidad 1, D tiene profundidad 2
- La **altura** de un árbol es la profundidad máxima de un nodo en el árbolEjemplo: El árbol de la figura tiene una altura de 2

Representación de un árbol binario

Los árboles binarios se pueden representar en memoria a través del:

- Uso de arreglos
- Uso de estructuras enlazadas

En el curso utilizaremos las estructuras enlazadas para representar los árboles, pues es la mejor forma y las más utilizada. Para definir la estructura de un nodo, se tendrá la clase ***Node***

```
class Node<T>{
   private T data;
   private Node<T> right;
   private Node<T> left;
   //otros métodos
};
```

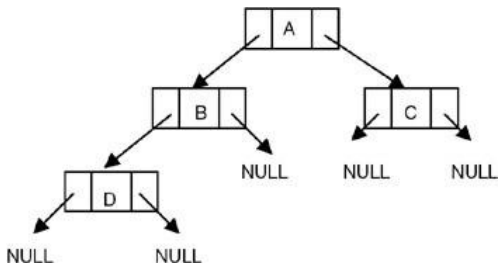


Figura 8.2. Estructura de un nodo

Recorridos de un árbol binario

- **InOrder**: Subárbol Izquierdo – Nodo (Raíz) – Subárbol Derecho
- **PostOrder**: Subárbol Izquierdo – Subárbol Derecho – Nodo (Raíz)
- **PreOrder:** Nodo (Raíz) – Subárbol Izquierdo – Subárbol Derecho

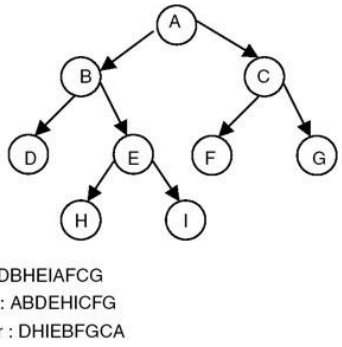


Figura 8.3. Recorridos de un árbol binario

BINARY SEARCH TREE (BST): Árbol de Búsqueda Binaria

Un BST es un árbol binario en el cual se debe cumple la siguiente regla:

- Cada dato de cualquier nodo del subárbol izquierdo es menor que el dato de la raíz.
- Un dato de cualquier nodo del subárbol derecho es mayor que el datos de la raíz.

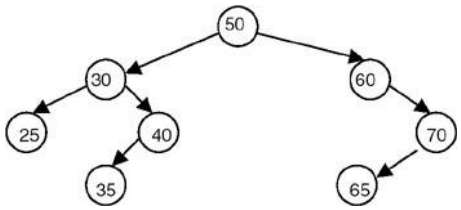


Figura 8.4. Árbol de búsqueda binaria

Operaciones de los BSTs

Las operaciones básicas son:

- **destroy()**: Elimina todos los elementos del BST dejándolo vacío
- **isEmpty()**: Verifica si el BST está vacío
- **insert(x):** Agrega el element '**x**' al BST.
- **remove(x):** Elimina el element '**x**' del BST.
- **search(x):** verifica la existencia del elemento '**x**' en el BST.
- **InOrder(), PostOrder(), PreOrder()**: recorren los elementos del árbol según corresponda al recorrido.

Adicionalmente se puede incluir otras operaciones que sean necesarias según sea el caso.

- **Búsqueda en un BST**

  Para buscar un elemento en el BST se debe de considerar la regla: **LEFT<Root<RIGHT**.

  En la figura 8.5, se observa el proceso de búsqueda de los elementos 4 y 18.

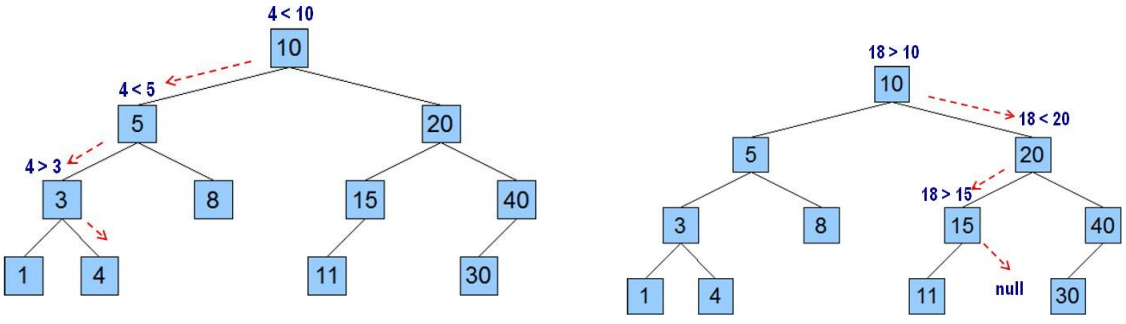


Figura 8.5. Proceso de búsqueda en un BST

- **Inserción en un BST**

  Para insertar un elemento se debe de considerar la regla del BST: **LEFT<Root<RIGHT**.

  Por ejemplo, se desea almacenar los números 8, 3, 1, 20, 10, 5 y 4 en un BST. La figura 5.6 nos muestra el proceso de inserción.

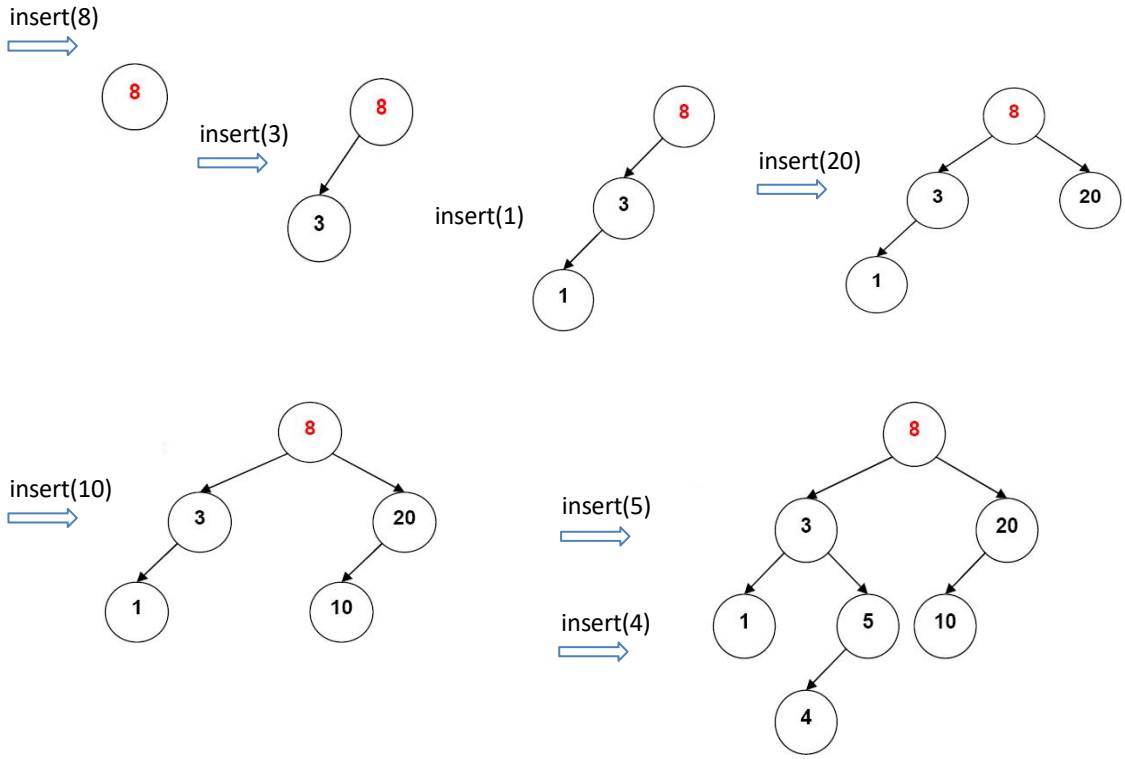


Figura 8.6. Proceso de inserción en un BST

- **Eliminación en un BST**

  En la eliminación de un elemento del BST, se identifican tres escenarios posibles:

  - **Caso 1**: Si se trata de una hoja se elimina directamente
  - **Caso 2**: Si tiene un único hijo se elimina el nodo haciendo que su nodo padre pase a referenciar a su nodo hijo
  - **Caso 3**: Si tiene dos hijos. Se puede proceder de alguna de las dos formas siguientes:
    a) Se sustituye el nodo por el menor elemento de su Subárbol Derecho (***sucesor en InOrden***)
    Ahora el objetivo es eliminar el nodo correspondiente a dicho menor elemento. Para esto se debe de volver a validar en que caso de los tres nos encontramos.
    b) O de forma simétrica, buscar el ***antecessor en Inorden***

  En la figura 8.7. podemos observar ejemplos de eliminación en un BST.

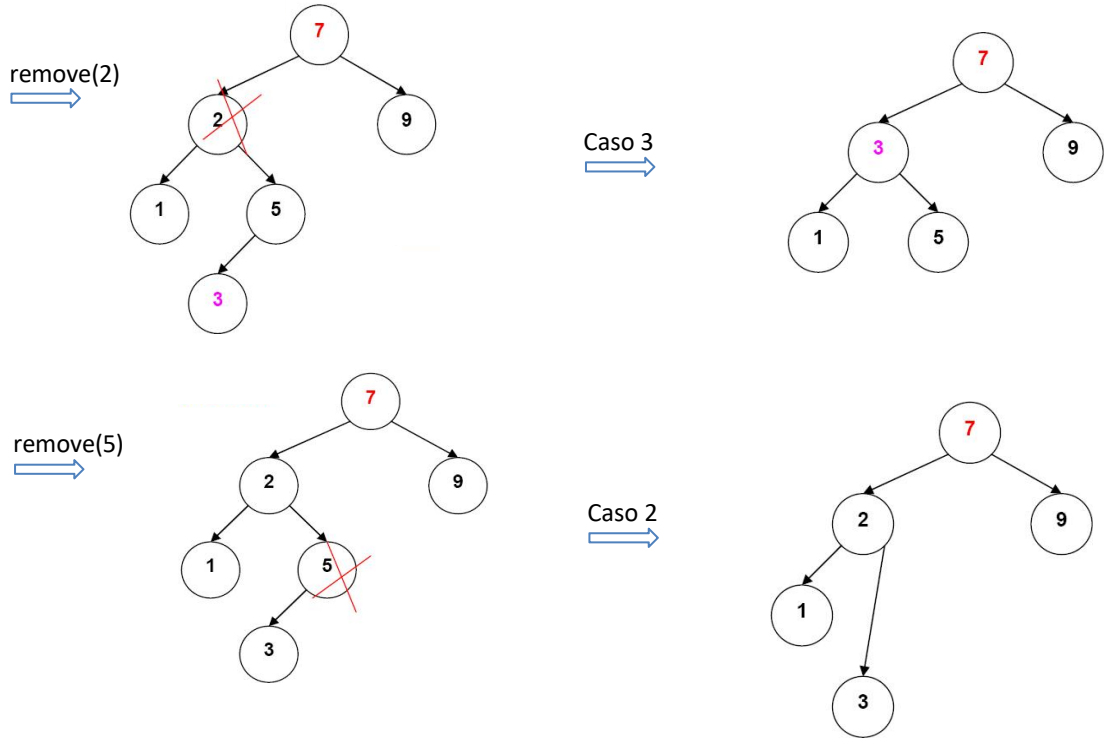
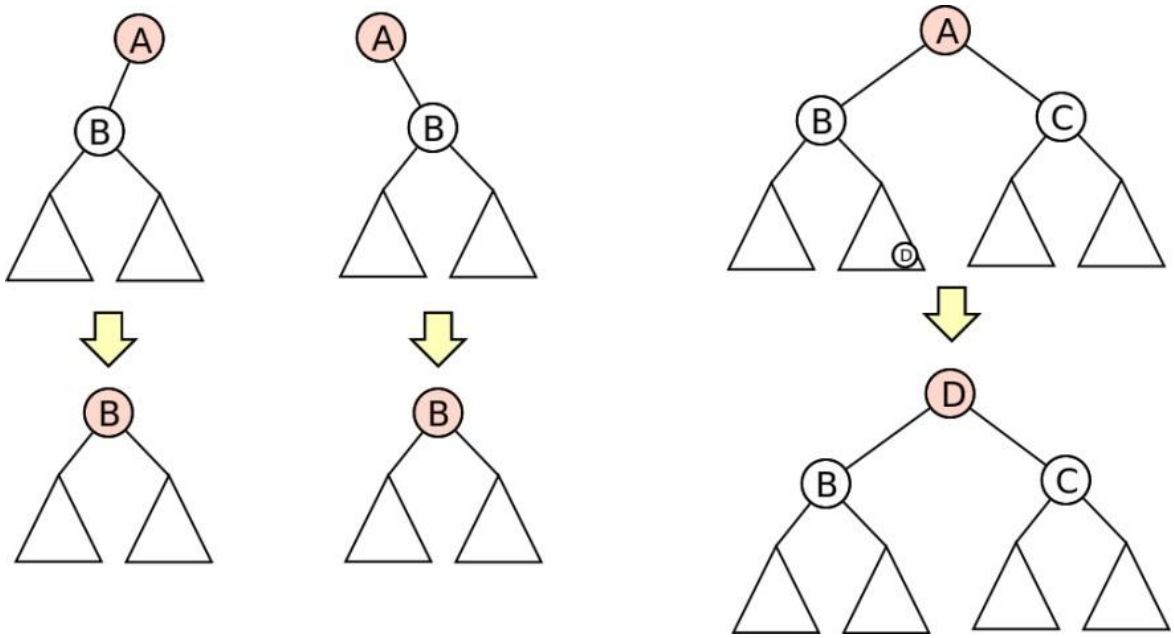


Figura 8.7. Eliminación del 2 y del 5 de un BST



IV

ACTIVIDADES

1. Aplicaciones de árboles binarios

Ingrese a las siguientes direcciones y pruebe cada una de las opciones de los applets y observe la forma como se realizan cada una de las operaciones en un BST.

a) http://btv.melezinek.cz/binary-search-tree.html
b) https://www.cs.usfca.edu/~galles/visualization/BST.html

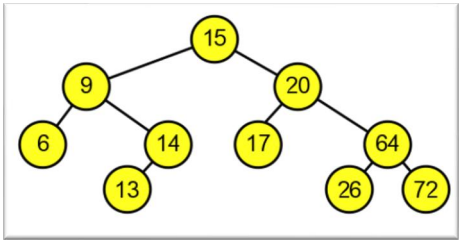
2. Representación Secuencial de un árbol binario en memoria

Vea el video del link siguiente, analice y comprenda como se almacena de manera secuencial en un arreglo lineal un árbol binario en memoria y la manera como se recupera para construir dicho árbol binario.

https://www.youtube.com/watch?v=8UwJPGUXMsQ&t=370s

Luego de ver, analizar y comprender el video, **realice en una hoja la representación de manera secuencial en memoria utilizando un arreglo lineal y un puntero, de los siguientes arboles binarios:**

i)



ii)

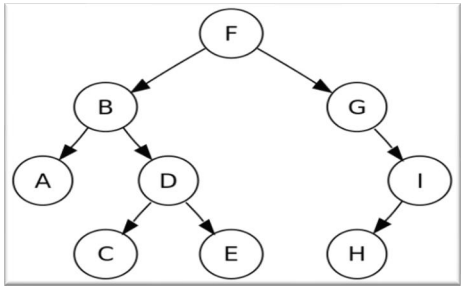


Figura 8.8. Árboles BST

3. Implementación del TAD BST Genérico

- Cree un nuevo Proyecto

3.1. Creando excepciones:

- Incorpore al proyecto el paquete de exceptions implementada en el laboratorio anterior.
- Agregue al paquete de exceptions, a la clase **ItemDuplicated** la misma que implememtará la excepción que ocurra cuando se ingrese a la estructura un elemento que ya existe en la misma.

```
public class ItemDuplicated extends Exception{
    public ItemDuplicated(String msg){
        super(msg);
    }
    public ItemDuplicated(){
        super();
    }
}
```

- En base al punto anterior, cree la clase ItemNoFound que gestionara la excepción que se presente cuando un elemento que será buscado no se encuentre en el BTS.

3.2. Creando la interface LinkBST genérica:

- Agregue al proyecto el paquete **treelink**, y en este paquete cree la interface **LinkBST** que contenga los siguiente métodos:

```
public interface LinkBST<E> {
   void insert(E x) throws ItemDuplicated;
   void remove(E x);
   E search(E x) throws ItemNoFound;
   boolean isEmpty();
}
```

3.3. Implementando el BST a través de una representación enlazada:

- Agregue al paquete *treelink* la clase **BSTree** cuya implememtación parcial se muestra a continuación. Complete los métodos que aún faltan implementar de manera progresiva. Esta clase debe de implementar la interface *LinkBST*

```
public class BSTree<E extends Comparable<E>> {
    class Node {
        protected E data;
        protected Node left, right;

        public Node(E data) {
            this (data,null,null);
        }
        public Node(E data, Node left, Node right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    private Node root;

    public BSTree(){ this.root = null; }
    public boolean isEmpty(){ return this.root == null; }

    //Metodo que inserta un elemento al BST.
    //Si el elemento ya existe en el árbol levanta la excepciÛn ItemDuplicated
    public void insert(E x) throws ItemDuplicated {
        //escriba su codigo aqui
    }

    //Metodo que busca un elemento y retorna su información.
    //Si no existe se levanta la excepción ItemNoFound
    public E search(E x) throws ItemNoFound {
        //escriba su codigo aqui
    }

    //Metodo que elimina un elemento del BST.
    //Si no existe se levanta la excepción ItemNoFound
    public void remove(E x) throws ItemNoFound {
        //escriba su codigo aqui
    }

    //Retorna la cadena que tiene toda la informaciÛn del BST.
    //Utiliza alguno de los recorridos
    public String toString() {
        //escriba su codigo aqui
    }
```

3.4. Eliminado un elemento del arbol:

- Adicione a la clase **BSTree** un método público **remove (x)** que permita eliminar un elemento 'x' cualquiera del árbol. Revise los tres casos posibles que se pueden dar.

3.5. Hallando el mínimo de un árbol:

Adicione a la clase **BSTree** un método privado **minRemove()** que retorne el nodo con el valor más bajo (mínimo) de un BST, y otro público, que retorne el dato correspondiente al nodo devuelto. Si no existe, se debe generar una excepción del tipo **ItemNoFound**.

V
EJERCICIOS

01. A la clase **BSTree** a gregue los siguientes métodos que son parte de la interface del árbol:
    a. Un método **countNodes()** que retorne el número de nodos no-hojas de un BST.
    b. Un método **height()** para retorne la altura de un nodo X cualquiera de un BST siempre y cuando exista en el árbol.
02. Agregue los siguientes métodos:
    a. Se define el área de un árbol binario como el número de nodos hojas multiplicado por la altura del árbol. Agregue un método público **areaBST()** a la clase **BSTree** que retorne el área del árbol binario de búsqueda.
    b. En la clase **Test** escriba el método **sameArea()** que retorne si dos árboles binarios diferentes tienen la misma área.
03. Adicione a la clase **BSTree** un método **iterativePreOrden()** (no recursivo) que recorra en PreOrden los nodos de un BST. Si requiere utilizar alguna estructura adicional, sólo puede usted utilizar las estructuras (TAD) que haya usted implementado en laboratorios previos.
04. Implemente un método recursivo que devuelva el número total de nodos de un árbol.

VI
BIBLIOGRAFÍA

- Weiss M., Data Structures & Problem Solving Using Java, Addison-Wesley, 2010. Chapter 18.
- Cutajar J. Beginning Java Data Structures and Algorithms, Packt Publishing, 2018. Chapter 3.
- Goodrich M., Tamassia R. and Goldwasser M., Data Structures & Algorithms, Wiley, 2014. Chapter 8

CALIFICACIÓN DEL LABORATORIO

CRITERIO A SER EVALUADO		Nivel				
		A	B	C	D	NP
R1	Correcta resolución de las actividades realizadas en las sesiones de laboratorio (excepto act. 1) (2 pto. cada actividad)	12.0 2.0(*)	1.5(*)	1.0(*)	0.5(*)	0.0(*)
R2	Correcta resolución de los ejercicios (2 pto. cada ejercicio)	8.0 2.0(*)	1.5(*)	1.0(*)	0.5(*)	0.0(*)
	Total	**20**				

(*) : de cada actvidad o ejercicio