



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

# Laboratorios de computación salas A y B

*Profesor:* M.I Edgar Tista García

*Asignatura:* Estructura de Datos y Algoritmos II

*Grupo:* 9

*No de Práctica(s):* 8-9

*Integrante(s):* Díaz Hernández Marcos Bryan

*No. de Equipo de  
cómputo empleado:* Equipo Personal

*No. de Lista o Brigada:* 9

*Semestre:* 2021-1

*Fecha de entrega:* 03 de diciembre del 2020

*Observaciones:*

**CALIFICACIÓN:** \_\_\_\_\_

## Objetivo de la practica

El estudiante conocerá e identificará las características de la estructura no-lineal árbol (árbol binario, binario de búsqueda y Árbol-B).

## Introducción

Esta practica se elaboró con la intención de poder comprender los árboles que en esta se elaboraron además de llevar de forma un poco más grafica la interpretación de estos, ya que esto ayuda a que reconozca con mayor facilidad las características de cada uno.

## Ejercicios de la practica:

- Ejercicio 1:

El primer ejercicio consistió en la implementación de las clases Nodo y Árbol Binario, que permiten la creación de un árbol, estas clases se proporcionaron para poder crear un árbol por medio de código concreto que inserta de nodo a nodo, y posteriormente modificar las clases, para realizar las operaciones de un árbol binario.

### a) Análisis de la implementación:

Lo más interesante de las clases fue como se manejaron los nodos, porque en algunas versiones que encontré en la investigación de como hacerlo, se manejaba dentro de una misma clase, pero si se realiza el análisis es adecuado porque el paradigma de Java permite la interacción entre clases y de esta forma no tener todo el código dentro de una misma clase.

```
public class Nodo {  
    int valor;  
    Nodo izq = null;  
    Nodo der = null;  
    Nodo padre=null; // $  
    // 1
```

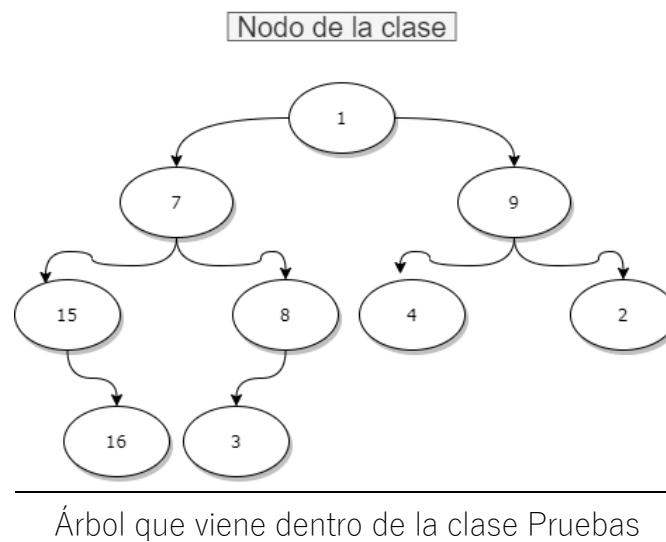
Atributos del nodo que permiten enlazar nodos.

Otro elemento que me resulto bastante curioso fue que solo se tuviera el atributo de raíz en la clase ArbolBin, porque igual pensé que de alguna forma los nodos sabrían cual es su padre, pero posteriormente encontré dos formas, crear un nodo que vaya apuntando al padre, o simplemente colocar el atributo. Además de la sobrecarga de los constructores que se tienen dentro de la clase.

```
*/  
public ArbolBin(int val){  
    root=new Nodo(val,null);  
}
```

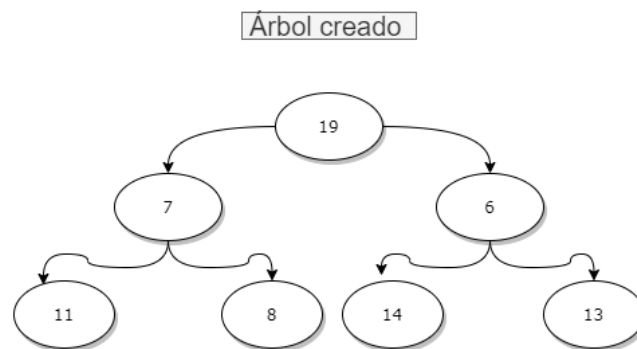
Este constructor es el que implemente.

b) Árbol que se realiza en la clase principal:



Lo que me resulta interesante es que se debe colocar las referencias a dos hijos en el constructor, en la implementación dentro de la clase Pruebas y que los nodos hijos no saben quién es su padre, lo que no resulta tan problemático en este caso.

c) Crea un segundo árbol e indica las ventajas de la implementación:



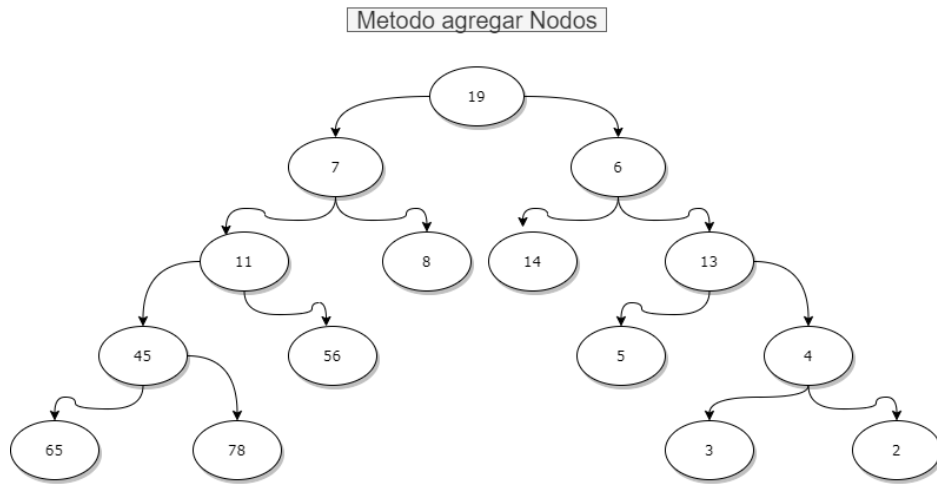
---

Árbol creado, este árbol lo borre en el programa pero el diagrama lo guarde.

Una ventaja de la implementación que se nos proporciona es que es bastante visible lo que realiza el programa, pero una desventaja dentro de la automatización de esta es que es forzoso el colocar la posición donde se va a colocar el nodo, en la derecha o izquierda, lo que es problemático, ya que el tipo de árbol tampoco tiene restricciones para decidir en que nodo se coloca sino que el usuario tendría que saber donde se va a colocar. O de lo contrario crear arboles nuevos, dentro de la forma en que lo implemente debido a que es bastante libre con respecto a donde se pueden insertar nodos.

d) Agrega un método de eliminación

Dentro de este método me centre en la elaboración del método que se indica y en la agregación de los nodos, lo que me causo demasiados problemas, porque buscaba hacerlo recursivo pero el no tener condiciones que me restringieran al árbol, no me daba ideas de como poder hacerlo, por lo que la implementación que se realiza en este inciso corresponde al siguiente diagrama:



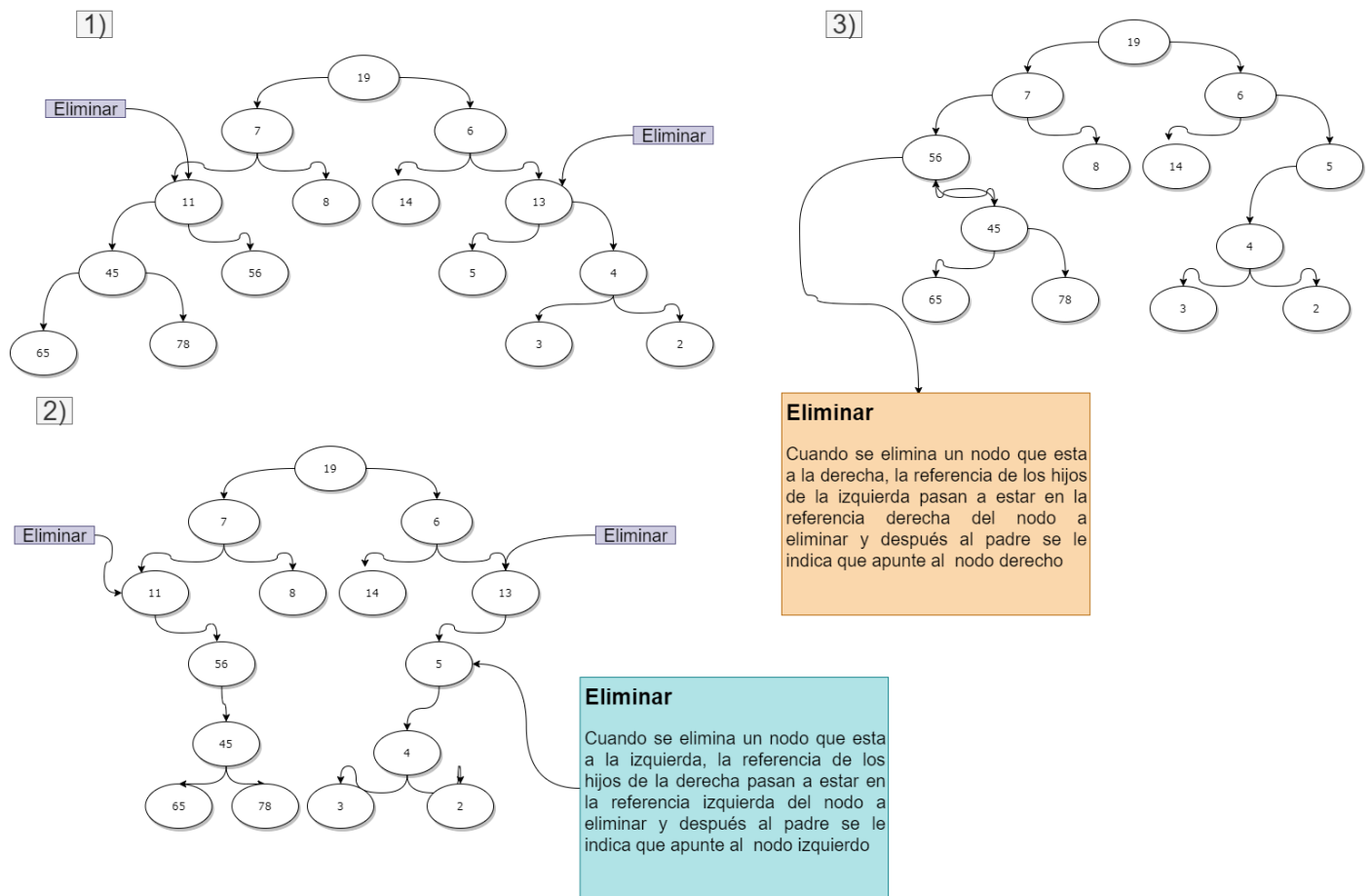
El árbol que se crea es de esta forma.

El diagrama muestra el árbol binario que se puede crear, donde las referencias del segundo nivel pueden existir las de la derecha, y de izquierda, pero cuando se pasa al siguiente nivel 3, se repite este proceso de solo poder indicar las referencias de la derecha e izquierda para el nodo.

En el caso del subArbol izquierdo solo se pude colocar un nodo derecho en cada nivel, porque el método de agregación busca el nodo mas a la izquierda, y en el caso del subArbol derecho solo deja colocar un nodo en izquierdo por nivel, ya que el método busca el mas a la derecha para colocar los nodos.

Eliminación:

Para el método de eliminación se toma la consideración de la forma de este árbol, para cuando los nodos que se van a eliminar son subArboles, y tienen hijos dependiendo si son derechos o izquierdos, se coloca el valor de los nodos hijos en la referencia donde ya no es posible agregar mas nodos por la inserción:



Se muestra como va eliminando, por lo que el método elimina de acuerdo a la forma específica que tienen los Árboles que se crean por medio de la agregación de los nodos.

e) Agrega un método de búsqueda, y que datos podrían ser de interés:

El método implementado devuelve un booleano que dentro de la selección se decide en cuanto al valor que se devuelva, en este caso se indica que existe o que no.

```
public boolean busqueda(int valor){
    Nodo buscar = root;
    Queue<Nodo> queue = new LinkedList();
    if(buscar!=null){
        queue.add(buscar);
        while(!queue.isEmpty()){
            buscar = (Nodo)queue.poll();
            if(buscar.valor==valor){
                return true;
            }
            if(buscar.izq!=null)
                queue.add(buscar.izq);
            if(buscar.der!=null)
                queue.add(buscar.der);
        }
    }
    return false;
}
```

Para realizar el método me base en las BFS, que visita todo el árbol entonces con la visita de todos los nodos, solo verifico el valor del nodo con el valor que se busca y si son iguales se regresa un true, y en caso contrario que no encuentre el valor se regresa un false. Otros datos relevantes podría ser el nivel de cada nodo, contar la copia, si fuera un árbol de expresión que se llevara a cabo la expresión y se regresara el valor de esta.

- Dificultades en el código

Lo más difícil de llevar a cabo la implementación de la creación y de la eliminación de los nodos, es que no hay elementos que condicionen al árbol, por lo que no hay como un límite que me dijera donde poder insertar valores, sino que era bastante general la implementación, por lo que decidí buscar una forma de implementarlo, ya que intente la recursividad pero no le veía forma para poder hacerlo de esta manera, entonces la elaboración quedo como se menciona en el inciso anterior, pero me hubiera gustado el determinar una manera de hacer que el usuario escogiera el nodo en donde insertar un nodo.

```

if(ladoA==1){
    Nodo nodoA = busquedaDer();//Se dirige
    System.out.println(nodoA.valor+" ");
    return nodoA;
}
if(ladoA==0){
    Nodo nodoA = busquedaIzq();//Se dirige
    System.out.println(nodoA.valor+" ");
    return nodoA;
}

```

Indica donde se guarda el nodo nuevo.

Por ello en el método y la opcion se le pide al usuario que se inserte donde se va a colocar el nodo, aunque se pudo haber pedido al padre del nodo y a partir de esto el comenzar a insertarlos, de hecho es una muy buena forma de poder implementarlo sin la necesidad de que el se tenga la referencia del padre.

Lo del padre surge porque en la eliminación es necesario saber cual es el padre del nodo, para que este ya no apunte a su hijo que se va a eliminar y que este modifique al árbol, además de que para poder eliminar un nodo se deben considerar varios casos con respecto a los hijos y además con respecto a las localidades que apunten a nada, en el caso de los hijos de la derecha e izquierda que no existan,

```

if(busqueda.izq==null){ //Nodo con un h:
    if(busqueda.padre.der==null){
        padre=busquedaInv(busqueda);
        busqueda.padre.izq=busqueda.der;
        retorno = new ArbolBin(padre);
        return retorno;
    }
}

```

Se consideran los hijos y localidades null

Lo que igual en el caso del Árbol que se crea es necesario que tenga valores por lo menos en los nodos derecho e izquierdo ya que los retornos del null, terminan rompiendo la ejecución del programa, y como no hago un manejo de excepciones estos rompen el programa.

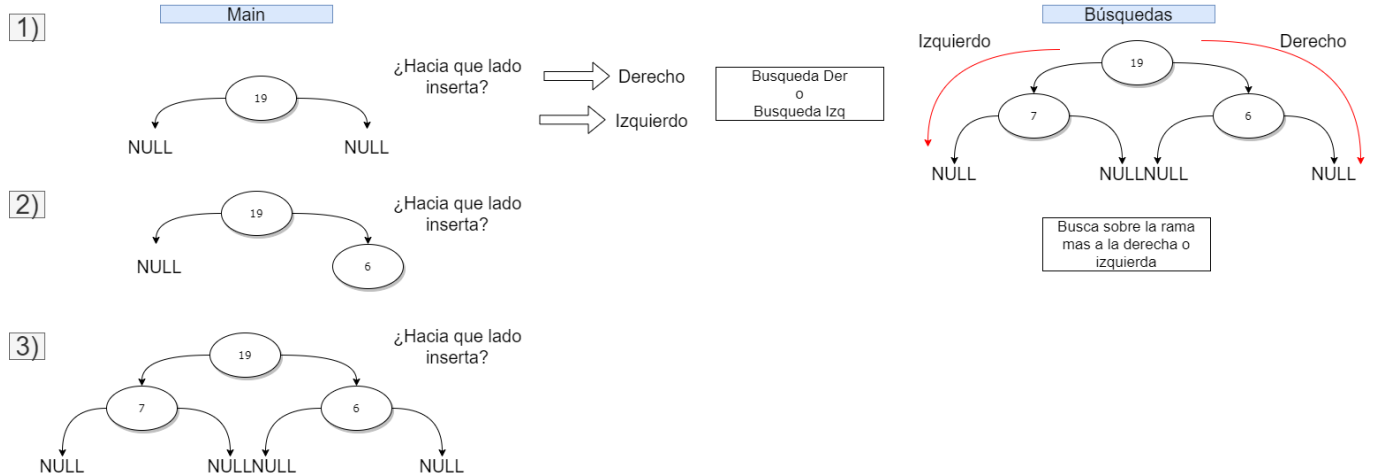
```

if (busqueda.izq != null && busqueda.der != null) {
    if (busqueda.padre.der.valor == busqueda.valor) {
        padre = busquedaInv(busqueda);
        busqueda.izq.izq = busqueda.der;
        busqueda.padre.der = busqueda.izq;
        retorno = new ArbolBin(padre);
        return retorno;
    }
}

```

Puede regresar un null, al buscar el valor de una localidad.

- Diagrama de funcionamiento:



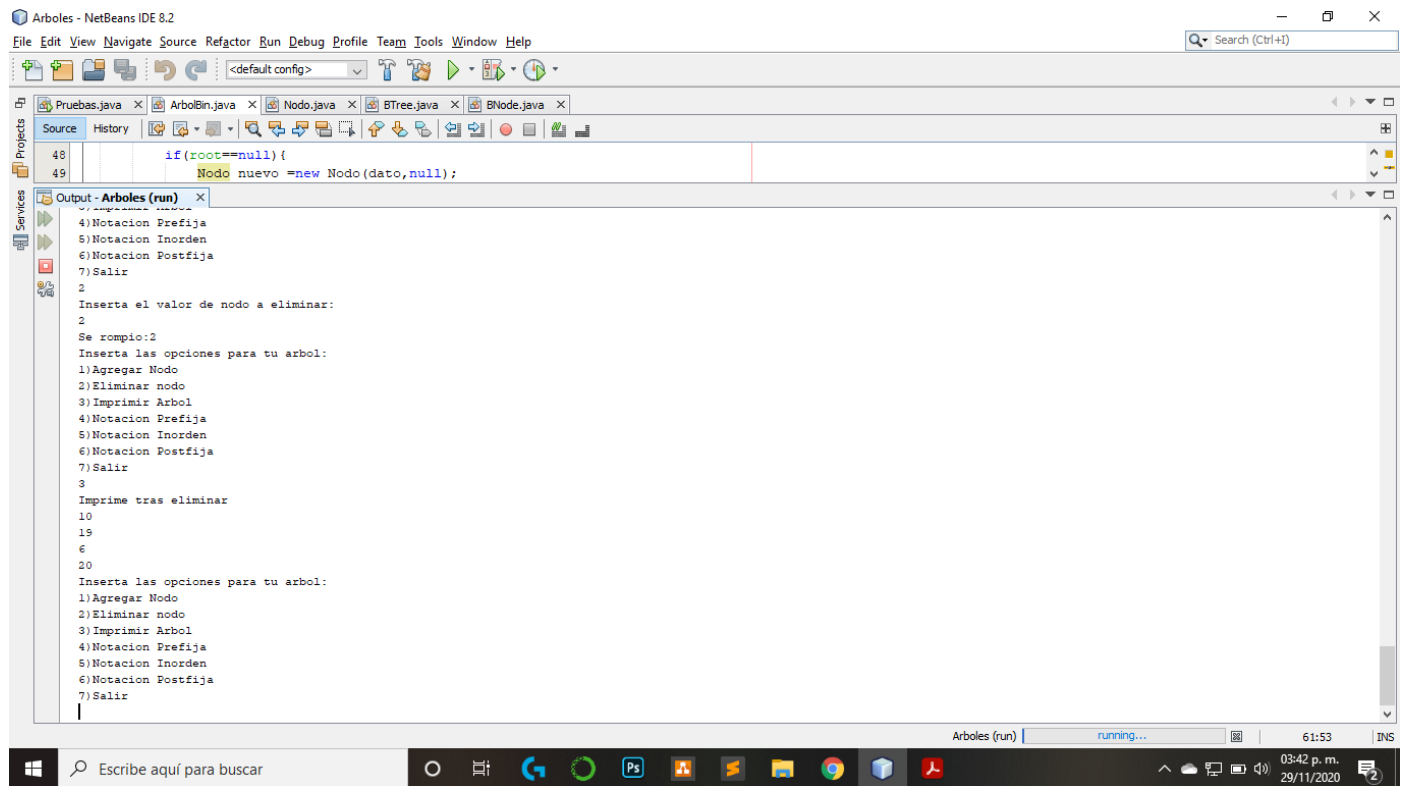
Como el diagrama lo muestra es necesario el indicar a que lado se va a insertar y en base a esto se busca el nodo mas a la derecha o izquierda y posteriormente se vuelve a pedir, hacia que lado se va a insertar derecho izquierdo. En caso de que no se inserte en la derecha o izquierda del segundo nivel en adelante, ya no se insertar en esas posiciones porque el programa va en busca de las hojas de cada subÁrbol y ahí es donde inserta.

- Relación con la teoría:

En este caso se utiliza la información que se nos dio en la clase de arboles binarios, ya que las consideraciones de la eliminación y de la inserción surgen de esa clase, y de eso depende la adecuada implementación de como saber eliminar un nodo y de como buscar en que nodo se debe agregar un nodo, pero como en este tipo de árbol no hay restricciones, no hay un limite con respecto a donde se agrega, a diferencia del árbol binario de búsqueda.

Esta libertad o la generalidad igual lo dificulta ya que no es posible limitarlo ya que por definición los valores de los nodos pueden estar en cualquier orden solo con cumplir con que los valores de los nodos no se repitan. Por lo que este y el tercer ejercicio están muy cargados de teoría.

- Evidencia de implementación



- Ejercicio 2:

En este ejercicio se busca la implementación de los recorridos en el árbol binario que se creó en el ejercicio anterior, además de que se puede implementar en los nodos nuevos, estas notaciones son las: preOrden, inOrden, postOrden.

- a) Prefija

Para el caso de la notación prefija, se toma en cuenta lo que vimos dentro de la clase ya que los valores de los nodos se comienzan a leer la raíz, después la lectura de los nodos de la izquierda, y al último la lectura de los nodos de la derecha.

- b) Infija

Para el caso de la notación infija, se toma en cuenta lo que vimos dentro de la clase ya que los valores de los nodos se comienzan a leer los nodos de la izquierda, después la lectura de la raíz, y al último la lectura de los nodos de la derecha.

- c) Postfija

Para el caso de la notación postfija, se toma en cuenta lo que vimos dentro de la clase ya que los valores de los nodos se comienzan a leer los nodos de la izquierda, después los de la derecha y al último de la raíz.

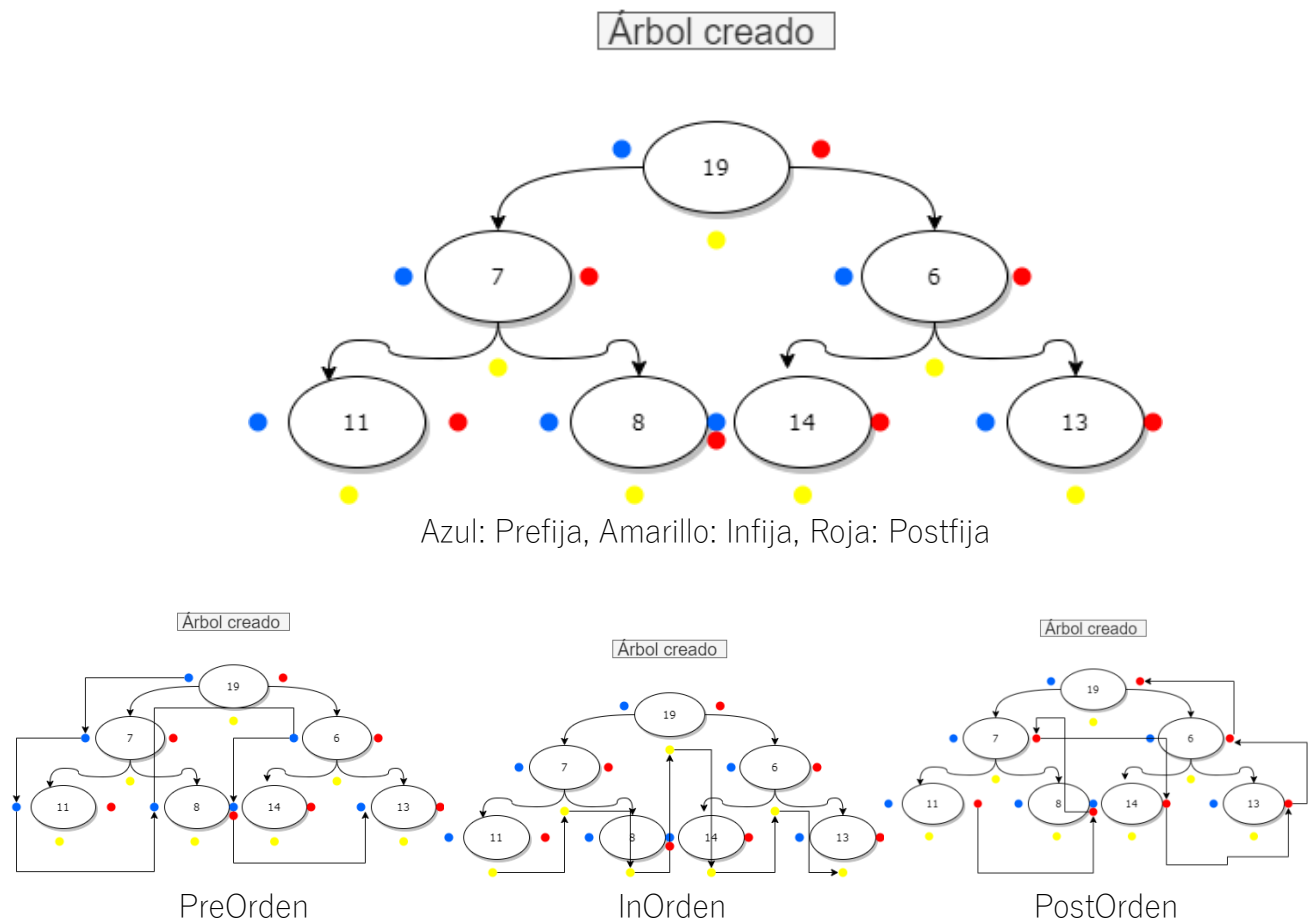
Todos los métodos son realizados por medio de la recursividad ya que solo se imprimen los datos y no se modifica ningún valor.



- Dificultades en el código:

En este caso no encontré dificultad con la impresión de las notaciones, no esta tan compleja ya se imprimen los datos de forma tal que se cumpla con las características de notación, además que en la investigación se mostraba la forma mas eficiente de implementarla, además que se reutilizan los métodos que ya existían en la clase.

- Diagrama de funcionamiento:



Como es posible ver las flechas indican la dirección de las notaciones, además para no ser redundante en la explicación de esta, que continua en la siguiente sección.

- Relación con la teoría:

Con respecto a la carga teórica del ejercicio se tiene que es necesario saber como se realiza el recorrido, en cada una de las notaciones, en la preOrden se toma en cuenta que no hay orden por decirlo de una manera, ya que se comienza desde la raíz, se recorre el subArbol izquierdo y después el derecho. Para la inOrden se tiene que se comienza en el orden del subArbol de la izquierda, después pasa por la raíz, y por ultimo el subArbol de la derecha y al final del subArbol de la izquierda, derecha y al final raíz.

- Evidencia de implementación

Arboles - NetBeans IDE 8.2

```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Pruebas.java x ArbolBin.java x Nodo.java x BTree.java x BNode.java x
Source History
81         }else{
82
83             arbol.inOrden(arbol.root);
84         }
85         break;
86
87         case 6:
88             if(contador>0){
89                 arbolNewA.postOrden(arbolNewA.root);
90             }else{
91                 arbol.postOrden(arbol.root);
92             }

```

Output - Arboles (run) #2

```

7) Salir
4
10
19
20
Vacio
Vacio
14
Vacio
Vacio
6
33
Vacio
Vacio
22
Vacio
Vacio
Inserta las opciones para tu arbol:
1) Agregar Nodo

```

Arboles (run) #2 running... 91:63 INS

PreOrden

Arboles - NetBeans IDE 8.2

```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Pruebas.java x ArbolBin.java x Nodo.java x BTree.java x BNode.java x
Source History
81         }else{
82
83             arbol.inOrden(arbol.root);
84         }
85         break;
86
87         case 6:
88             if(contador>0){
89                 arbolNewA.postOrden(arbolNewA.root);
90             }else{
91                 arbol.postOrden(arbol.root);
92             }

```

Output - Arboles (run) #2

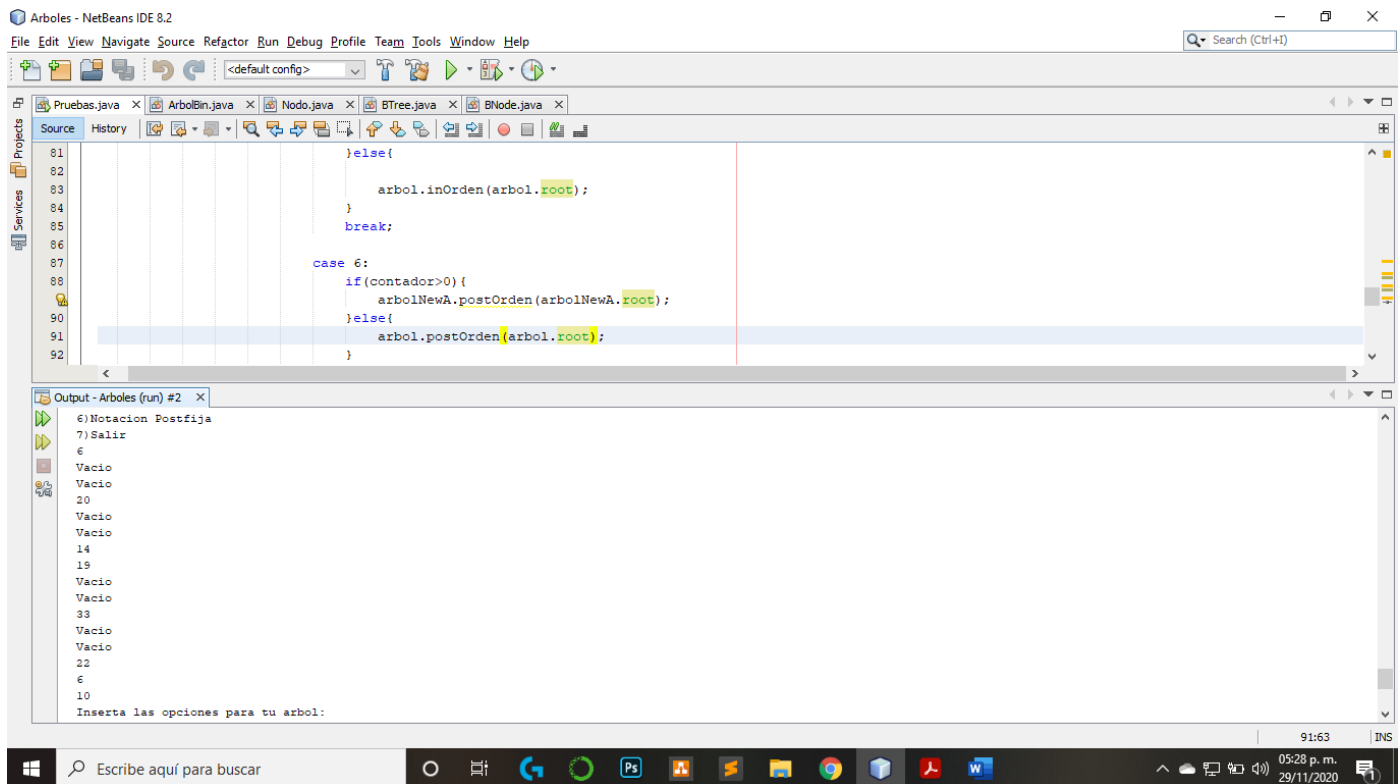
```

5) Notacion Inorden
6) Notacion Postfija
7) Salir
5
Vacio
20
Vacio
19
Vacio
14
Vacio
10
Vacio
33
Vacio
6
Vacio
22
Vacio

```

91:63 INS

InOrden



## PostOrden

- Ejercicio 3:

El tercer ejercicio consistía en la implementación de los árboles binarios de búsqueda, por medio de la herencia de los métodos del árbol binario, y mediante la sobreescritura crear los métodos correspondientes a los árboles binarios de búsqueda.

- Dificultades en el código

Durante la elaboración de este ejercicio tuve bastante información disponible para poder guiarme con respecto a cómo elaborar el ejercicio de forma recursiva, pero no lo pude hacer de esa forma por más que lo intenté, y fue a causa de que pensaba que el utilizar de forma directa la raíz iba a modificar los datos pero viéndolo desde el paradigma orientado a objetos eso es lo que hacen los métodos de forma normal, lo que me llevó a seguir un método de eliminación por medio de la recursividad pero siempre me salía la siguiente excepción:

```

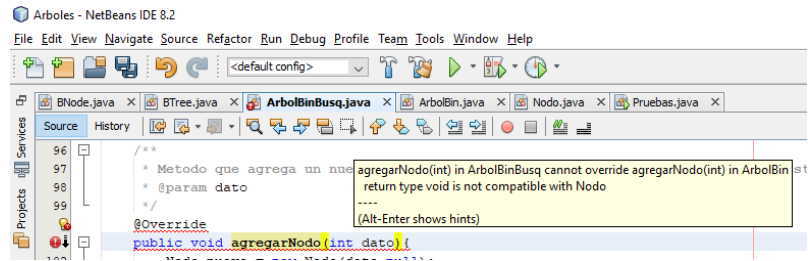
15
Exception in thread "main" java.lang.NullPointerException
    at ArbolBinBusq.agregarNodoBusq(ArbolBinBusq.java:115)
    at Pruebas.main(Pruebas.java:115)
C:\Users\Brain\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 11 seconds)

```

El error más común en este ejercicio.

Debido a esto decidí hacerlo por medio de ciclos que se rompieran bajo ciertas condiciones, en el caso del método de la creación y la eliminación de los nodos, fue lo que más me importó en la realización de los métodos, entonces comencemos con el de inserción de nodos.

Como mencionaba el principal problema que tuve dentro de la elaboración fue la creación de este método, más que la eliminación de los nodos, porque bien se explica en la practica que estos árboles son una especialización de los arboles binarios, por lo que se pide implementar la sobre escritura de los método pero en este caso lo que realice no sobrecargo los métodos debido a que marcaba el siguiente mensaje:



Error al hacer la sobreescritura.

En ese caso lo que decidí fue solo cambiarle el nombre al método y listo, para evitar ese tipo de errores de dentro de la VM, continuando con las complicaciones, el agregar nodo lo implemente de forma similar al método de inserción de un árbol, solo que se toma la consideración de que los valores deben estar ordenados por ello es muy importante el uso de la condicionales, y mediante un ciclo while se puede buscar de forma continua, tal y como lo hace el BFS, ya que me base demasiado en ese método para poder resolver los ejercicios del árbol binario y del árbol binario de búsqueda.

```
while(true){
    padre=viajero;
    if(dato<viajero.valor){
        viajero=viajero.izq;
        if(viajero==null){
            padre.izq=nuevo;
            nuevo.padre=padre;
            break;
        }
    }
}
```

Método que busca el nodo donde insertar.

Y en el caso de la eliminación de los nodos los problemas que tuve estaban relacionados con la lógica del la eliminación, porque en determinados casos, se tenía que verificar que el nodo no apuntara a un null, y esto porque para los distintos casos el nodo padre podría tener una o dos referencias a sus hijos, pero en caso de que se comprobara la posición y no existiera nada en uno de esto hijos se retornaba un null, y esto rompía el programa.

```
Nodo padre;
if(busqueda.der==null && busqueda.izq==null)
    if(busqueda.padre.der==null){
```

Regreso del null

En esos casos lo que tenía que hacer en el árbol era el repetir el código para cada posible retorno de un null, lo que hacia repetitivo el código, todo con la intención de saber la dirección en que estaba el nodo a eliminar, pero en este caso para la eliminación el mismo orden del árbol da la apertura de poder saber en que lado esta el nodo a eliminar, y para evitar esas comprobaciones utilice un booleano que se verifica para cada caso de eliminación tal como se hizo en el método del árbol binario.

- Análisis:

Dentro de la sección anterior comente los errores mas comunes que tuve, pero la parte de eliminación dentro de mi parecer es la mas importante, porque se verifica la integridad del árbol después de que se borra un miembro de este.

Entonces cuando se elimina una hoja, la raíz o un subÁrbol con un solo hijo, no hay mucho problema ya que solo se reemplaza el hijo, se elimina la hoja o se selecciona una nueva raíz, pero en el caso de eliminar un nodo con dos hijos, es lo que más complejidad tiene. Para ese caso se toma que es necesario el sustituir el nodo por el hijo mas derecho, pero la cercanía que tendría al valor del padre del nodo a eliminar en cuanto al valor, por lo que el método que se implementa para esta búsqueda esta basado en dos métodos que realice en la clase de árbol binario:

```
public Nodo busquedaDer(){
    Nodo derecha = root;
    Queue<Nodo> queue = new LinkedList();
    if(derecha!=null){
        queue.add(derecha);
        while(!queue.isEmpty()){
            derecha = (Nodo)queue.poll();
            if(derecha.der!=null){
                queue.add(derecha.der);
            }else{
                return derecha;
            }
        }
    }
}
```

Método de búsqueda del más derecho.

El método de la búsqueda del mas derecho se comporta como un nodo de búsqueda, pero el problema es que comienza desde la raíz del árbol, por lo que no comenzaría desde el nodo izquierdo que se marca como primer paso de búsqueda, así que por eso no implemente el método, ya que sería más código redundante, en ese caso solo hice un while que buscara a la derecha a partir del nodo que recibe, y si este es diferente al nodo del padre, que se recibe se comprueba que el nodo a eliminar tenía nietos y por ello sería necesario el conservarlos, colocándolos a la izquierda del nodo que regresa para sustituir el nodo a eliminar, y en base a sus posibles posiciones se indica la referencia hacia este, y al final se colocan los hijos de la derecha del nodo a eliminar en la derecha del nodo que sustituye.

```

17
public Nodo buscador(Nodo viajero){
    Nodo intercambio=viajero;
    Nodo viajeroI=viajero.izq;
    //busca hasta la derecha final
    while(viajeroI!=null){
        intercambio=viajeroI;
        viajeroI=viajeroI.der;
    }
    //Cuando el padre tiene hijos
    if(intercambio!=viajero.izq){
        intercambio.izq=viajero.izq;
    }
}

```

Busca el nodo más derecho.

En cuanto a la implementación de añadir nodos, se implementa un método similar al de buscador, únicamente que se condiciona a que se direcciona la búsqueda de acuerdo con el valor del dato, mediante un while infinito que se rompe en cuanto se encuentra la posición donde se va a colocar el nuevo nodo.

```

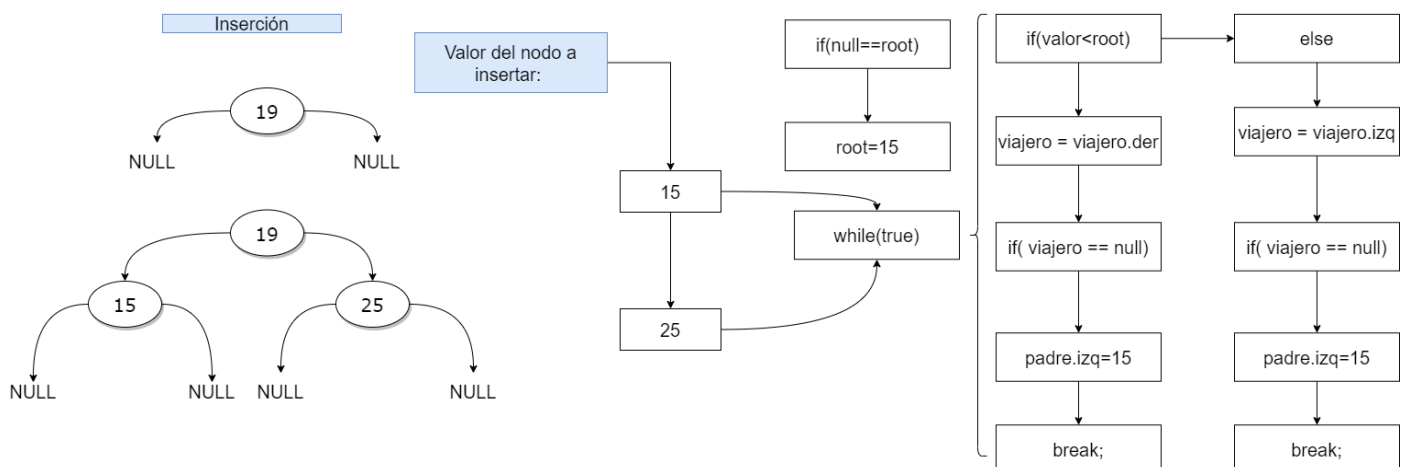
viajero=viajero.izq;
if(viajero==null){
    padre.izq=nuevo;
    nuevo.padre=padre;
    break;
}

```

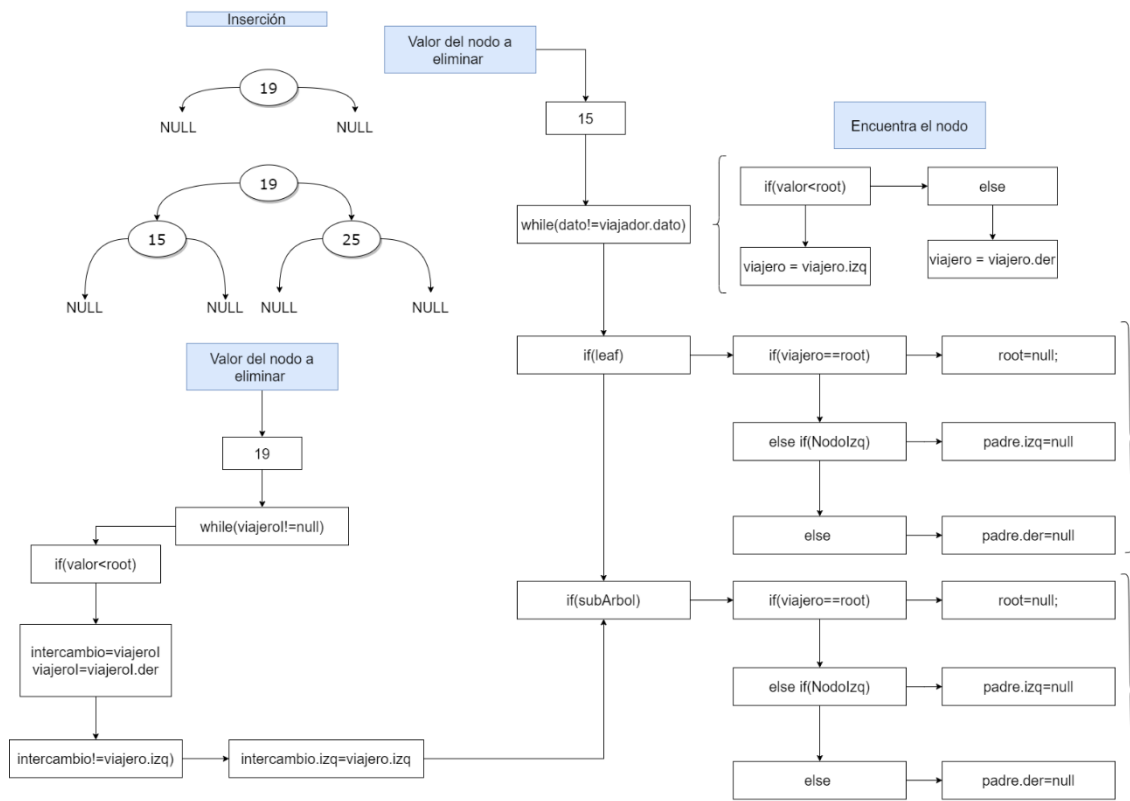
Indica el nodo donde se va a colocar el nuevo nodo.

El método es bastante eficiente en cuanto a la búsqueda del nodo, y volviendo a la implementación de los árboles binarios, me base en el método de BFS, para realizarlos todos, además de revisar un libro muy bueno de árboles, donde venía la versión recursiva pero no aplicaba a la forma de mis árboles a si que solo la utilice para poder comenzar a realizar la implementación mediante los while.

- Diagrama de funcionamiento:



En este diagrama se representa el como se verifica el valor y se continua con el proceso de verificación de forma continua hasta que se rompa el ciclo.



Representación de cómo funciona la eliminación de acuerdo con el nodo que se elimina y si este tiene hijos o no.

- Relación con la teoría:

La carga teórica de este ejercicio esta totalmente relacionada con las clases virtuales que se vieron, además de que el desarrollo de los métodos y de las clases permite identificar las herencias entre los diferentes tipos de arboles y como estos varían de acuerdo con la sobreescritura de los métodos.

- Evidencia de implementación

The screenshot shows the NetBeans IDE with the following components:

- Source Editor:** Contains Java code for a binary tree implementation, including methods for inserting, deleting, and searching nodes.
- Output Window:** Displays the execution of the program, showing the menu options and the state of the tree after several operations.

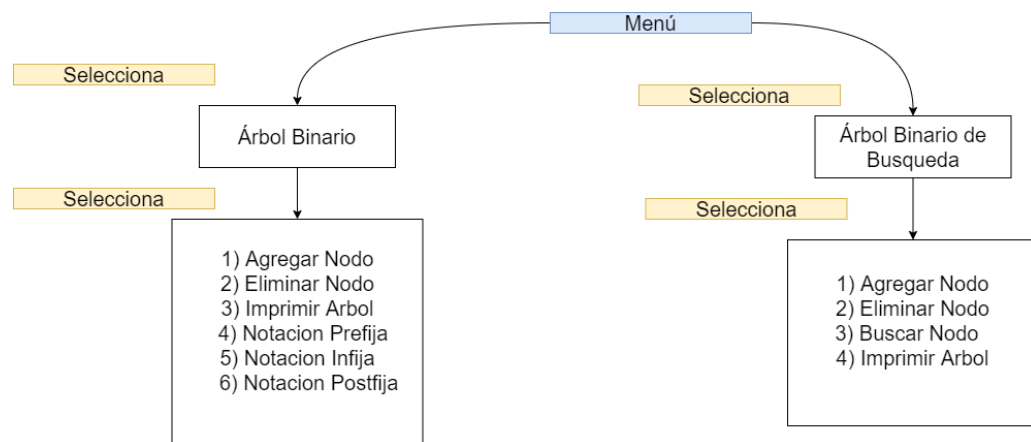
- Ejercicio 4:

El cuarto ejercicio consistió en crear un menú para poder trabajar con los dos tipos de arboles definidos con anterioridad, y de esta forma hacer el proceso de creación, añadir, buscar, eliminar, algo mas consecutivo y que se reflejaran las opciones en el árbol que se seleccione para trabajar.

- Dificultades en el código

En este caso no hubo dificultades en la programación sino que los fallos correspondían a la lógica de los ejercicios anteriores y en ellos describo las dificultades correspondientes.

- Diagrama de funcionamiento:



Básicamente eso es lo que es el ejercicio, donde cada opcion manda a llamar a los correspondientes métodos, y estos están descritos en los ejercicios anteriores, por lo que de igual forma al estar elaborando cada uno de los ejercicios fui elaborando el menú así que este ejercicio salió solo, porque la creación del menú desde el primer árbol facilitaba la implementación de las opciones de cada árbol.

En las opciones del árbol binario se tiene que cada una hace lo siguiente:

- a) Agregar Nodo: en esta opcion se llama al método que va a agregar un nodo, pero este será la raíz, es decir que añade y crea el árbol en la primera ejecución, y posteriormente va añadiendo más nodos de acuerdo con la forma prevista.
- b) Eliminar Nodo: eliminar los nodos de acuerdo con la posición de estos, si son raíces, sin son subÁrboles con uno o dos hijos, y si son hojas.
- c) Imprimir Árbol: utiliza el BFS, para visitar cada nodo e imprimir el contenido de este.
- d) Notación Pre: visita los nodos en comenzando por la raíz, subArbol izquierdo y por ultimo subÁrbol derecho.
- e) Notación In: visita los nodos en comenzando por subArbol izquierdo, raíz y por ultimo subÁrbol derecho.



- f) Notación Post: visita los nodos en comenzando por subArbol izquierdo, subÁrbol derecho y por último la raíz.

En las opciones de árbol binario de búsqueda:

- Agregar Nodo: en esta opción se llama al método que va a agregar un nodo, pero este será la raíz, es decir que añade y crea el árbol en la primera ejecución, y posteriormente va añadiendo más nodos de acuerdo con la forma de los nodos ya que este tipo de árbol permite el implementar de forma más eficaz el agregar nodos, debido a que los que se inserten deben de estar en una determinada posición para cuidar que los valores menores al padre van a la izquierda y los mayores a la derecha.
- Eliminar Nodo: eliminar los nodos de acuerdo con la posición de estos, si son raíces, si son subÁrboles con uno o dos hijos, y si son hojas, además de que se aprovecha el valor del nodo a eliminar y de esta forma buscar el nodo a eliminar por medio de estructura que se maneja en el árbol.
- Buscar Nodo: busca el valor en cada nodo de acuerdo si es mayor o menor, entonces verifica en cada nodo si el valor es mayor o menor, de esta forma se aprovecha la estructura.
- Imprimir Árbol: utiliza el BFS, para visitar cada nodo e imprimir el contenido de este.

- Evidencia de implementación

```
Arboles - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Pruebas.java ArbolBinBusq.java ArbolBin.java Nodo.java BTree.java BNode.java
Source History
130
131
132
133
} else {
    System.out.println("El valor existe en el arbol");
}
break;
Output - Arboles (run)
4) Eliminar nodo
3) Buscar
4) Imprimir BFS
5) Salir
4
Arbol
10
5
15
2
7
12
16
Inserta las opciones para tu arbol:
1) Agregar Nodo
2) Eliminar nodo
3) Buscar
4) Imprimir BFS
5) Salir
3
Inserta el valor del nodo:
10
El valor existe en el arbol
Inserta las opciones para tu arbol:
1) Agregar Nodo
2) Eliminar nodo
3) Buscar
4) Imprimir BFS
5) Salir
3
```

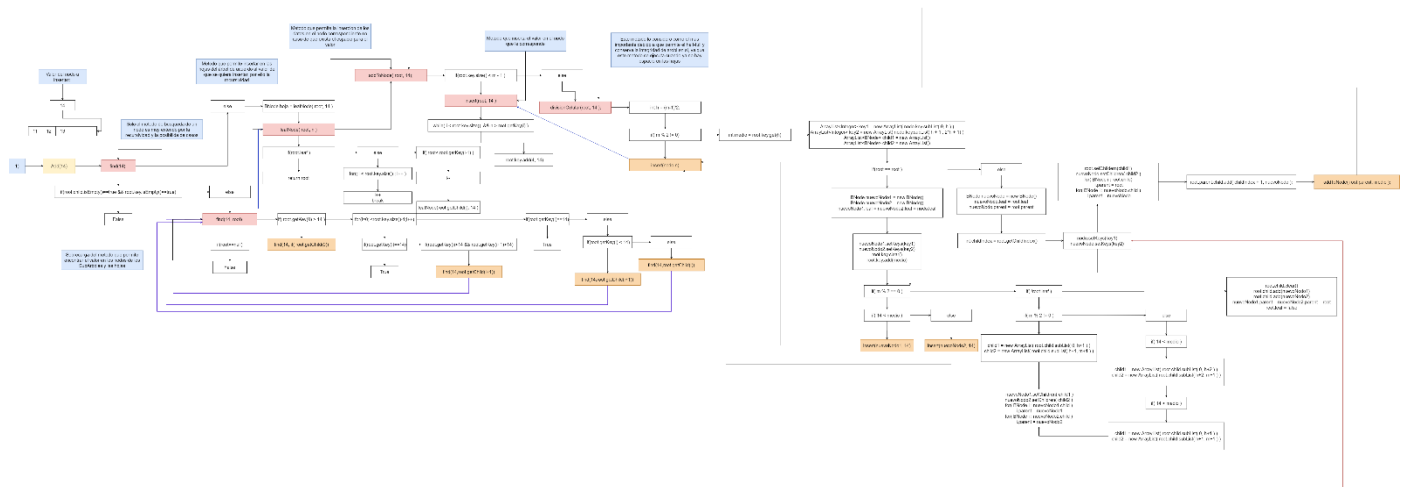
- Ejercicio 5:

El ejercicio consistió en elaborar un análisis de las dos clases proporcionadas para la implementación de los árboles B, donde se debe describir cada clase y método que se utiliza.

- Dificultades en el código

En este caso no hubo porque no modifique nada del código únicamente, lo analice e implemente.

- Diagrama de funcionamiento:



Estoy seguro de que no es perceptible lo que dice el diagrama, por ello le envié todos los diagramas creados y las respectivas imágenes, mas que nada se visualiza la ruta de los datos y como la recursividad se vuelve algo complejo por la gran cantidad de operaciones que se pueden llevar a cabo dentro de la ejecución del código o la implementación de este tipo de árbol.

Al análisis no lo he realizado escrito porque la verdad me resulto mas sensato el realizar un diagrama que mostrara la forma en que este trabaja y como se comunican los métodos, para que yo y quien lo viera lo pudiera interpretar de forma más fácil, además que dentro del código cada clase esta comentada y lleva comentarios con respecto a los elementos mas relevantes que me llamaron la atención o que son importantes para la ejecución de la creación del Árbol B.

- Relación con la teoría:

La teoría es por excelencia necesaria para poder desarrollar la implementación de esta clase, ya que dentro del código es comprensible lo que hace cada uno de los métodos pero el análisis de los métodos, de las clases, atributos necesarios para poder crear un árbol B, es bastante compleja en este caso, aunque en la investigación de esta también existen versiones con arreglos lo que vuelve más sencilla implementación. Pero en si es necesario saber los conceptos básicos para poder comprender la clase.

- Evidencia de implementación

The screenshot shows the NetBeans IDE 8.2 interface. The main editor displays a Java file named `ArbolBin.java` with the following code snippet:

```

130         }else{
131             System.out.println("El valor existe en el arbol");
132         }
133         break;

```

The Output window, titled "Output - Arboles (run)", shows the execution of a Java application. The output includes a menu of options for working with a binary tree:

```

Arbol
10
5
15
2
7
12
16
Inserta las opciones para tu arbol:
1)Agregar Nodo
2)Eliminar nodo
3)Buscar
4)Imprimir BFS
5)Salir
3
Inserta el valor del nodo:
10
El valor existe en el arbol
Inserta las opciones para tu arbol:
1)Agregar Nodo
2)Eliminar nodo
3)Buscar
4)Imprimir BFS
5)Salir
3

```

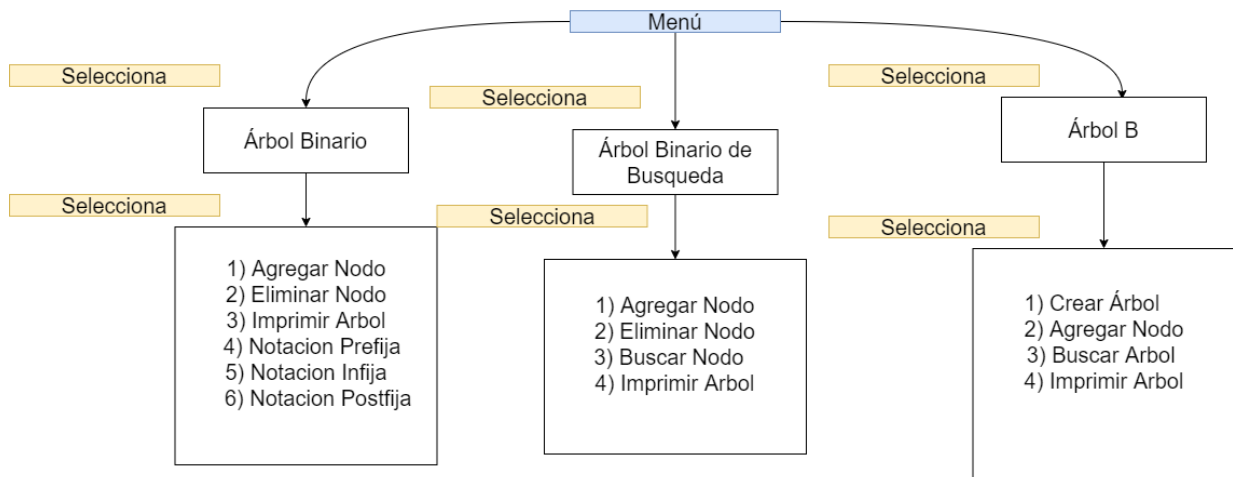
- Ejercicio 6:

El ejercicio consistió en añadir al menú de los arboles la opcion de poder trabajar con los arboles B, para este tipo de arboles se añaden las opciones de la creación, añadir, buscar e imprimir el árbol.

- Dificultades en el código

En este caso no hubo dificultades en la programación sino que los fallos correspondían a la lógica de los ejercicios anteriores y en ellos describo las dificultades correspondientes.

- Diagrama de funcionamiento:



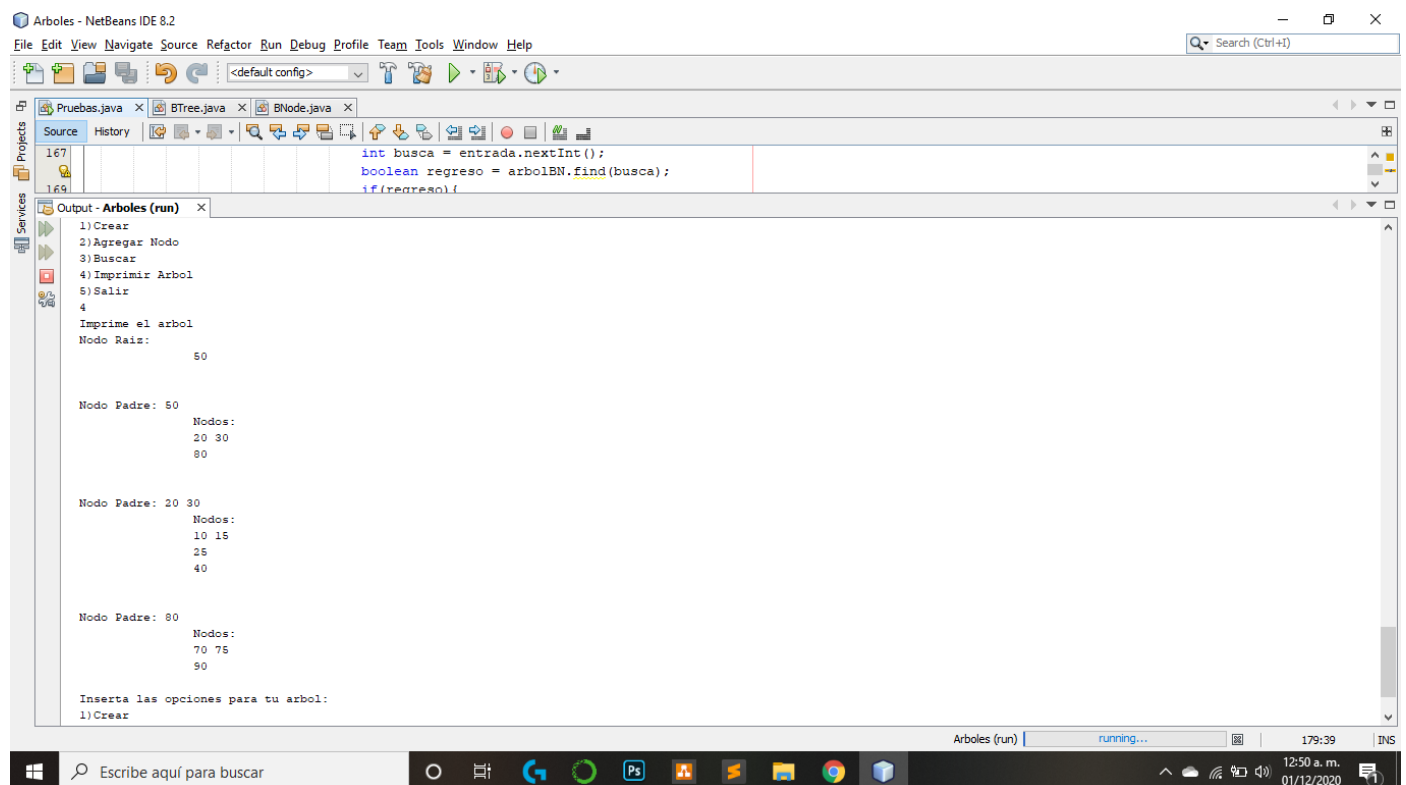
Básicamente se basa en esta forma el menú donde se agrega la opción de llevar a cabo la implementación de los Árboles B, de igual forma que el método anterior pues se estuvo desarrollando conforme se realizaba la implementación de cada método y eventualmente el menú surgió de esta forma.

\*Buscar nodo.

En las opciones del Árbol B:

- Crear árbol: en este caso se inicializa la raíz y se debe indicar el número de referencias que contendrá el árbol ya que este se define gracias a que contiene  $m-1$  elementos en cada nodo y que debe tener un mínimo de  $(m-1)/2$  elementos para poder ser half-full.
- Agregar Nodo: implementa una forma para la inserción de los elementos de un nodo, los cuales son, que se agregue a la raíz, a uno subÁrbol con uno o dos hijos, y a una hoja, para esto se tienen más consideraciones como que se añade en las hojas y se evalúa que exista una mínima cantidad de elementos y que no se exceda una máxima cantidad.
- Buscar Nodo: en este caso se busca el nodo y si lo encuentra regresa un booleano que se recibe y se trata para poder saber si existe o no.
- Imprimir árbol: se implementa una versión del BFS, en este caso la implementación va guardando los hijos de cada nodo, y va imprimiendo los valores de cada nodo, visitando cada nodo por lo menos una vez.

- Evidencia de implementación



```
Arboles - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Pruebas.java BTree.java BNode.java
Source History
167 int busca = entrada.nextInt();
168 boolean regreso = arbolBN.find(busca);
169 if(regreso){
Output - Arboles (run)
1) Crear
2) Agregar Nodo
3) Buscar
4) Imprimir Arbol
5) Salir
4
Imprime el arbol
Nodo Raiz:
50
Nodo Padre: 50
Nodos:
20 30
80
Nodo Padre: 20 30
Nodos:
10 15
25
40
Nodo Padre: 80
Nodos:
70 75
90
Inserta las opciones para tu arbol:
1) Crear
```

## Conclusiones

Esta ha sido una de las practicas mas interesantes que he realizado en todo el tiempo de cuarentena porque me mantuvo despierto por demasiado tiempo y me dejo con ganas de poder implementar un árbol del tipo B+, aunque tal vez lo haga, siento que el conocimiento que obtuve en esta practica es muy importante, además que me ayudo a poder repasar los subtemas para el examen.

Por otro lado siento que la mi implementación del árbol binario es carece de forma o no es tan general por la condición de la inserción de los nodos, por lo que en ese aspecto desearía mejorarla, en los demás no tengo esa duda con respecto a si están bien o no y creo igual a que es porque dentro de la elaboración de las practicas me encontré con varios topes, pero al final resulto bastante confortante poder terminirlas.

Por último el objetivo se cumplió dentro de mi percepción, ya que pude resolver los ejercicios, y analizarlos fue un elemento que me permitió el poder comprender mejor el funcionamiento, por lo que puedo concluir que obtuve el conocimiento necesario para decir que se cumplió con el objetivo de la práctica.

Referencia:

Joyanes L. (2008). Árboles. *ESTRUCTURAS DE DATOS EN JAVA*. Páginas: [367-396]. Madrid: MCGRAW-HILL.