	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	TISTA GARCÍA EDGAR
Asignatura:	ESTRUCTURA DE DATOS Y ALGORITMOS II
Grupo:	5
No de Práctica(s):	PRÁCTICA #8-9: ÁRBOLES
Integrante(s):	CARRILLO CERVANTES IVETTE ALEJANDRA
No. de Equipo de cómputo empleado:	TRABAJO EN CASA
No. de Lista o Brigada:	09
Semestre:	2022 - 1
Fecha de entrega:	16 NOVIEMBRE 2021
Observaciones:	

CALIFICACIÓN: _____

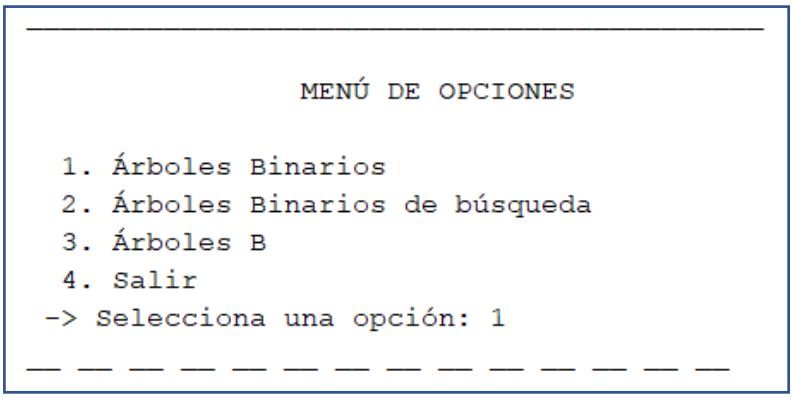
Práctica #8-9: Árboles

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

Objetivo de clase: El alumno analizará las implementaciones proporcionadas y conocerá la forma en la que se pueden implementar estas estructuras de datos.

ACTIVIDADES PARA EL DESARROLLO DE LA PRÁCTICA

Esta práctica tiene como objetivo implementar los diferentes tipos de árboles vistos en clase. Con el fin de hacer más sencillo el uso de este programa, se creó un menú de opciones para poder crear los diferentes tipos de árboles antes mencionados y descritos más a detalle a continuación.



Menú de opciones -> Principal.java

- ***Árboles Binarios***

Para la implementación de los árboles binarios, lo primero que se hizo fue crear las siguientes dos clases:

  - *Nodo*

Esta clase tiene como objetivo crear un nodo con su hijo derecho e izquierdo, tiene como atributos:

- valor. (variable de tipo int, el cual indicará el valor del valor que se le asigno al nodo creado)
- izquierdo. (variable de tipo Nodo, la cual indica el hijo que tendrá a lado izquierdo el nodo padre, o el nodo creado).
- derecha. (variable de tipo Nodo, la cual indica el hijo que tendrá a lado derecho el nodo padre, o el nodo creado).

Teniendo ya los atributos de esta clase, se implementaron los métodos de acceso para cada uno de ellos, para poder modificar el valor del nodo, así como su hijo derecho e izquierdo.

- *ArbolBin*

Esta clase tiene como objetivo crear el árbol binario el cual se ocupará para probar las diferentes operaciones que puede manejar un árbol, como son los que se describirán más adelante. Lo primero que se hizo fue inicializar los atributos de la clase:

- Objeto tipo Nodo. (el cual contendrá la raíz del árbol).
- Lista de Nodos. (en esta se almacenarán los nodos que se vayan creando y eliminando de cada árbol.

Ya teniendo los atributos de cada clase, lo siguiente que se implementó fueron los métodos constructores sobrecargados y los métodos de acceso correspondientes a cada uno de los atributos mencionados anteriormente. Asimismo, se crearon los siguientes métodos que más se utilizarán conforme vaya avanzando el programa:

- add. Tiene como parámetro el nodo padre, el nodo hijo que se quiera agregar, así como el lado en el que se quiere agregar el hijo (0 izquierda, 1 derecha). Dentro de este método se asigna el hijo al nodo padre, según el lado que se le indique, esto se logra con los métodos de acceso de la clase Nodo “setIzq()” y “setDer()”.
- visit. Tiene como parámetro el nodo que se visitará en la función “*breadhFrist*”, en este método se imprime el valor del nodo pasado como parámetro.
- breadthFirst. Tiene como función recorrer e imprimir el árbol binario mediante el método BFS, para ello se debe de obtener el nodo raíz del árbol binario creado y de ahí realizar el recorrido correspondiente.

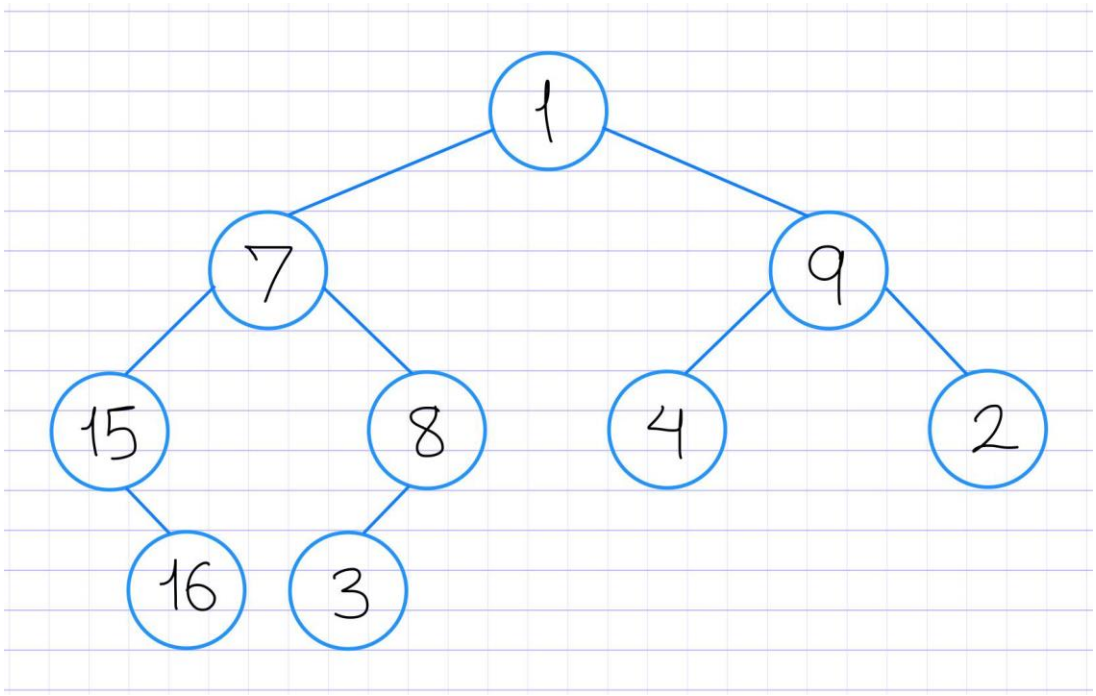
Para probar estos métodos y familiarizarnos con la implementación, se creó una nueva clase llamada “*pruebas*”, en donde se realizó manualmente la creación de dos árboles, es decir, sin ayuda de ningún ciclo de repetición o parecido. Para ello se crearon los nodos que se ocuparían en el árbol binario, y posteriormente se le asignaron uno con otros. Finalmente se imprimió cada árbol con ayuda del método breadthFirst, la salida es la siguiente:

```
ARBOL BINARIO 1
1 7 9 15 8 4 2 16 3

ARBOL BINARIO 2
16 6 1 24 26 7 2 31 9 36 3
```

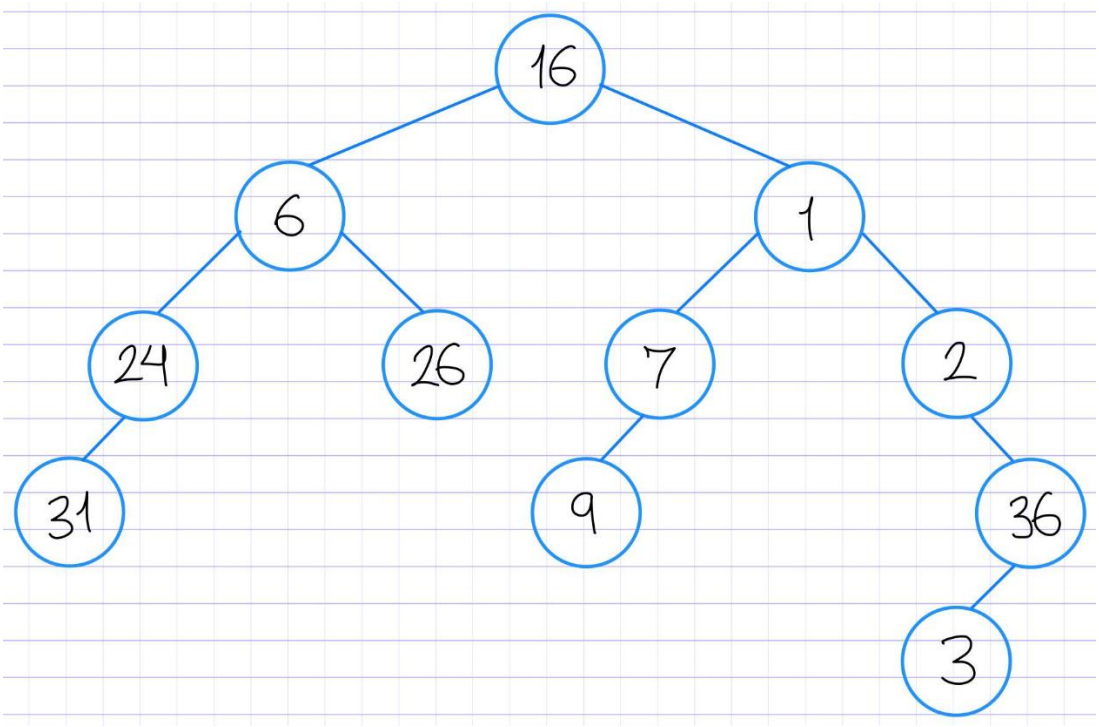
Recorrido primer árbol -> Prueba.java

El primer árbol creado, gráficamente se ve de la siguiente manera:



Árbol 1

El segundo árbol creado, gráficamente se ve de la siguiente manera:



Árbol 2

Ya familiarizados con la implementación de como crear los árboles binarios, se agregaron las operaciones correspondientes para ocupar en el árbol, tanto en la clase “ArbolBin”, como en un submenú de opciones en la opción **Árbol binario**:

```
. ____ . ____ . ____ . ____ . ____ .

        SUBMENU
  1. Agregar dato
  2. Eliminar dato
  3. Buscar elemento
  4. Imprimir árbol (BFS)
  5. Notación Prefija (preorden)
  6. Notación Infija (inorden)
  7. Notación Posfija (posorden)
  8. Salir
 -> Selecciona una opción: 4
```

Submenú de opciones “Árbol Binario” -> Principal.java

Para que se pudiera probar cada una de estas operaciones, se tuvieron que implementar los siguientes métodos en la clase “ArbolBin”

Para crear el árbol

- crearABinario. Tiene como parámetros la lista que contiene el árbol, en la cual se agregarán los valores que el usuario vaya ingresando, para lograr esto, se le pregunta al usuario cuantos nodos quiere agregar y con base al resultado se realiza un ciclo de repetición en el cual se le pide al usuario ingresar el valor de cada nodo, asimismo crea este y lo agrega a la lista que tiene el árbol como atributo. Posterior a ello, se crea un ciclo foreach el cual recorre toda la lista de nodos y va llamando al método “relacionar”, en cada iteración.
- relacionar. Tiene como parámetro el árbol, la lista que contiene este árbol y el nodo padre el cual se quiere agregar (en el método anterior, en el último ciclo for, este padre será el nodo “n” de la lista). En este método se le pregunta al usuario si que el nodo que creó tenga nodos hijos, en caso de, se creó un pequeño menú de opciones en el cual se le pregunta al usuario que dijo quiere que tenga su nodo (izquierda, derecha o ambos) y con base a eso, se le pide el valor de los nodos hijos y se modifica el valor de los hijos del padre correspondientemente.

```
— — — — — — — — — — — — — —

        ARBOLES BINARIOS

¿Cuantos nodos quieres que tenga tu árbol binario? 6
Ingresa el valor del nodo 1 (raíz): 16
Inserta el valor del nodo 2: 1
Inserta el valor del nodo 3: 2
Inserta el valor del nodo 4: 7
Inserta el valor del nodo 5: 36
Inserta el valor del nodo 6: 6

. __ . __ . __ . __ . __ . __ . __

Nodo: 16

* NOTA: Solo puedes ingresar nodos creados anteriormente *

  1. Si
  2. No
 -> ¿El nodo tiene hijos? 1

. . . . . .
```

Se repetirá por cada nodo

Crear Árbol

Opción 1

- agregarNodo. Tiene como parámetros el árbol en el cual se agregará el valor y así como la lista que tiene dicho árbol. Se buscará los nodos que tengan alguna referencia null como hijos y se imprimirán estos con el fin de que el usuario sepa a quien no se le puede agregar algún valor hijo, una vez encontrado el valor agregar en la lista de nodos se le pregunta al usuario si lo quiere agregar y en caso de ser sí se llama al método correspondiente para modificar el hijo.

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        AGREGAR DATO

Nodos en los que puedes agregar hijos:
 1  2  7  6

Elige el nodo en el cual quieras agregar el nodo hijo: 1

Solo tiene disponible el nodo Derecho
    1. Si
    2.No
 -> ¿Agregar?1
Ingresa el valor del nodo Derecho: 3
```

Opción 2

- eliminarNodo. Este método tiene como parámetro la lista que tiene el árbol binario y el nodo eliminar, en este método se va a recorrer todo el árbol tanto derecha como izquierda hasta encontrar el último valor que esté más a la derecha y se intercambiará este por el nodo que se quiere eliminar. Este método se hizo principalmente para eliminar un nodo de un árbol binario de búsqueda; sin embargo, funciona para ambos.

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        ELIMINAR DATO

Ingresa el nodo a eliminar: 36
```

Opción 3

- Imprimir árbol. Para implementar esta opción coma se hizo uso del método descrito anteriormente “breadthFirst”

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        IMPRIMIR ARBOL (BFS)

16 6 7 1 2 3
```

Opción 4, 5, 6

- Para estas 3 opciones se crearon 3 métodos similares, cada uno de ellos obtiene el valor raíz del árbol y se compara este valor, si es diferente a null, recorre tanto el lado izquierdo como derecho del árbol e imprime el nodo al principio, a la mitad y al final del método correspondientemente.

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        NOTACION PREFIJA (PREORDEN)

PreOrden: {
 16  6  7  1  3  2 .  ____  .  ____  .  __
```

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        NOTACION INFIJA (INORDEN)

InOrden: {
 6  16  3  1  7  2 .  ____  .  ____  .  _

. __ . __ . __ . __ . __ . __ . __ . __ .

        NOTACION POSFIJA (POSORDEN)

PosOrden: {
 6  3  1  2  7  16 .  ____  .  ____  .  _
```

- ***Árboles Binarios de búsqueda***

Al igual que el método anterior, para ver las operaciones de este tipo de árbol, se realizó un submenú:

```
.  ____  .  ____  .  ____  .  ____  .  ____  .

      SUBMENU
 1. Agregar dato
 2. Eliminar dato
 3. Buscar
 4. Imprimir árbol (BFS)
 5. Salir
-> Selecciona una opción:
```

Submenú de opciones "Árbol Binario de Busqueda" -> Principal.java

Para crear un árbol de busqueda binaria, a diferencia del árbol anterior, solo se le pide el valor de los nodos al usuario, puesto que se creó un método correspondiente para ordenar el árbol de busqueda y cumpliera con la regla que los hijos de la izquierda tuvieran un valor menor que el padre, y los de la derecha tuvieran un valor mayor que el padre.

```
__ __ __ __ __ __ __ __ __ __ __ __ __ __

      ARBOLES BINARIOS DE BUSQUEDA

¿Cuantos nodos quieres que tenga tu árbol binario? 6
Ingresa el valor del nodo 1 (raíz): 16
Inserta el valor del nodo 2: 6
Inserta el valor del nodo 3: 8
Inserta el valor del nodo 4: 22
Inserta el valor del nodo 5: 21
Inserta el valor del nodo 6: 24
```

Opción 1

- Agregar dato. Para esta opción se siguió la misma lógica que al crear el árbol de busqueda; sin embargo, en vez de que cada nodo del árbol se vaya creando en un ciclo for, se omitió esa parte y se crea directamente

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        AGREGAR DATO

Ingresa el nodo por agregar: 9
```

Opción 3

- Se realizó la busqueda comparando cada una de las claves, si el numero a buscar era mayor se iba a lado derecho, y si era menor a lado izquierdo, pero en dado caso de no encontrar dicho número, se quedaba en false.

```
· __ · __ · __ · __ · __ · __ · __ · __ ·

        BUSCAR

¿Qué nodo quieres buscar? 24
¿Se encuentra? true
```

Opción 2 y 4

- Se implementó el método eliminar descrito anteriormente

```
· __ · __ · __ · __ · __ · __ · __ · __ ·

        IMPRIMIR ARBOL (BFS)

16 6 22 8 21 24 9
```

- ***Árboles B***

Al igual que el método anterior, para ver las operaciones de este tipo de árbol, se realizó un submenú:

```
·  ____  ·  ____  ·  ____  ·  ____  ·  ____  ·

      SUBMENU
 1. Agregar un valor
 2. Buscar valor
 3. Imprimir árbol
 4. Salir
-> Selecciona una opción:

 __ __ __ __ __ __ __ __ __ __ __ __ __ __

        ARBOLES B

 ¿De qué orden es tu árbol? 4
 ¿Cuantos nodos quieres crear? 6
 Ingresa el valor del nodo 0: 1
 nodo key size0
 Ingresa el valor del nodo 1: 4
 nodo key size1
 Ingresa el valor del nodo 2: 6
 nodo key size2
 Ingresa el valor del nodo 3: 5
 Ingresa el valor del nodo 4: 7
 nodo key size2
 Ingresa el valor del nodo 5: 2
 nodo key size1
 · __ · __ · __ · __ · __ · __ · __ · __ ·

        IMPRIMIR ARBOL

 Nodo Raiz:
                4


 Nodo Padre: 4
                Nodos:
                1 2
                5 6 7
```

```
. __ . __ . __ . __ . __ . __ . __ . __ .

        BUSCAR VALOR

Ingresa el valor que quieres buscar: 6
El valor 6 si se encontró en el árbol (:
```

Conclusiones

Considero que se cumplieron los objetivos de esta práctica, ya que se vio las principales características de una estructura no lineal, en este caso un árbol binario normal, un árbol binario de búsqueda, así como un árbol B punto. Asimismo, se implementó cada una de estas estructuras en una clase correspondiente.

Personalmente me costó mucho trabajo implementar los métodos de eliminar nodos para el árbol binario de busqueda; sin embargo, dado que empecé con este método, lo pude implementar también en el árbol binario normal. Al principio de la práctica también me costó mucho conflicto el hecho de crear el árbol, ya que al ingresar los datos mediante un ciclo for no se guardaban los nodos correctamente (no se “vinculaban” bien, unos a otros), por lo que decidí tener una lista como atributo del árbol binario para que ahí se vayan guardando los nodos creados, así como los eliminados y no se presente algún otro conflicto en lo que resta del programa.

Con respecto a los árboles B, me costó mucho trabajo entender la teoría ya que me confundía un poco diferenciar las operaciones de los diferentes tipos de árboles; no obstante, con esta práctica ya me quedo mucho más en claro la implementación, así como la teoría de en que consiste cada uno de los árboles vistos en clase.

Considero que esta práctica ha sido una de las más complicadas de implementar, ya que en lo personal me costó un poco de trabajo entender como se implementa la estructura de un nodo en la programación orientada a objetos; sin embargo, al recordar las clases de EDA 1, note que existe una cierta comparación la implementación de los nodos dados en clase que con las listas ligadas hechas con estructuras.

A mi parecer, consideró que los ejercicios planteados en esta práctica estuvieron muy bien para entender mejor este tema, ya que tiene una gran importancia.