



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

Número de Práctica(s): Guía práctica de estudio 8: Árboles

Integrante(s): MURRIETA VILLEGAS ALFONSO | VALDESPINO MENDIETA JOAQUÍN

*Núm. De Equipo de
cómputo empleado* : 38

Semestre : 2019 - 1

Fecha de entrega: 9 DE OCTUBRE DE 2018

Observaciones:

CALIFICACIÓN: _____

ÁRBOLES. PARTE I

OBJETIVOS DE LA PRÁCTICA

- El estudiante conocerá e identificará las características de la estructura no lineal árbol.

INTRODUCCIÓN

En las estructuras de datos lineales como las pilas o las colas, los datos se estructuran en forma secuencial es decir cada elemento puede ir enlazado al siguiente o al anterior. En las estructuras de datos no lineales o estructuras multi-enlazadas se pueden presentar relaciones más complejas entre los elementos; cada elemento puede ir enlazado a cualquier otro, es decir. puede tener varios sucesores y/o varios predecesores. Ejemplos de estructuras de datos no lineales son los grafos y árboles.

Un árbol es una colección de elementos llamados nodos, uno de los cuales se distingue como raíz, junto con una relación(rama) que impone una estructura jerárquica entre los nodos. Los árboles genealógicos y los organigramas son ejemplos de árboles.

Un árbol puede definirse formalmente de forma recursiva como sigue:

- Un solo nodo es, por sí mismo un árbol. Ese nodo es también la raíz de dicho árbol.
- Suponer que n es un nodo y que A_1, A_2, \dots, A_m son árboles con raíces, n_1, n_2, \dots, n_m , respectivamente. Se puede construir un nuevo árbol haciendo que se constituya en el padre de los nodos, n_1, n_2, \dots, n_m . En dicho árbol, n es la raíz y A_1, A_2, \dots, A_m son subárboles de la raíz.

La definición implica que cada nodo del árbol es raíz de algún subárbol contenido en el árbol principal. Un árbol vacío o nulo es aquel que no tiene ningún nodo.

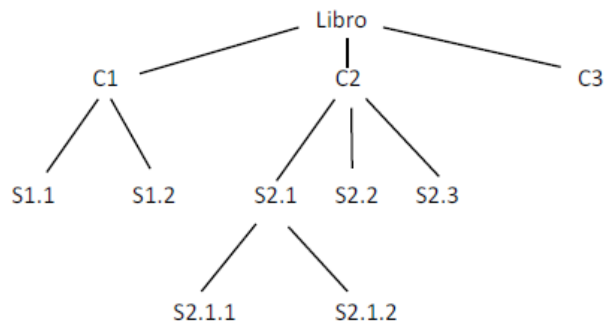


Imagen 1: Ejemplo de un árbol

TÉRMINOS Y CONCEPTOS DE ÁRBOLES

Grado de un nodo: Es el número de subárboles o hijos que tienen como raíz ese nodo.

Nodo terminal u hoja: Nodo con grado 0, no tiene subárboles.

Grado de un árbol: Grado máximo de los nodos de un árbol.

Hijos de un nodo: Nodos que dependen directamente de ese nodo, es decir, las raíces de sus subárboles. Si un nodo x es descendiente directo de un nodo y, se dice que x es hijo de y.

Padre de un nodo: Antecesor directo de un nodo, nodo del que depende directamente. Si un nodo x es antecesor directo de un nodo y, se dice que x es padre de y.

Nodos hermanos: Nodos hijos del mismo nodo padre.

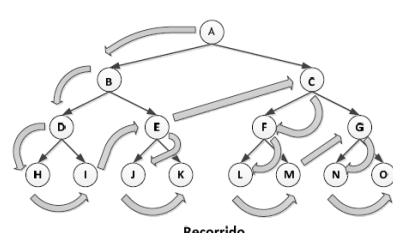
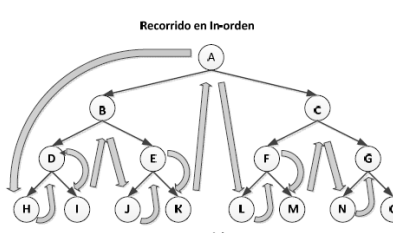
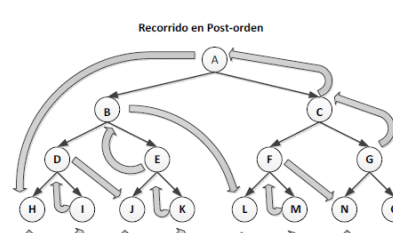
RECORRIDOS

El recorrido de una estructura de datos es el acceso sistemático a cada uno de sus miembros. Hay dos formas básicas de recorrer un árbol; en profundidad y en amplitud

RECORRIDOS POR PROFUNDIDAD

Siempre empieza accediendo a la raíz. Cuando es en profundidad se trata de alejarse en todo lo posible de la raíz hasta alcanzar un nodo hoja. Una vez alcanzado se da un paso atrás para intentar alejarse por un camino alternativo. Este esquema implica que por nodo se pasa varias veces: cuando se accede por primera vez y cuando se regresa de cada uno de sus hijos para acceder al siguiente o volver al padre.

El tratamiento del nodo se puede hacer en cualquiera de esas ocasiones, lo que da lugar, según el momento que se elija a tres variantes: los recorridos en **preorden**, **inorden**(simétrico) y **postorden**

PRE_ORDEN	IN_ORDEN	POST_ORDEN
Se trata primero la raíz del árbol y, a continuación, se recorren en preorden sus subárboles de izquierda a derecha (raíz - subárbol izq. - subárbol der).	Se recorre en inorden el primer subárbol, luego se trata de la raíz y, a continuación se recorre en inorden el resto de los subárboles (subárbol izq. - raíz - subárbol der).	Se recorren primero en postorden todos los subárboles, de izquierda a derecha, y finalmente se trata la raíz (subárbol izq. - subárbol der. - raíz).
 <p>Recorrido A,B,D,H,I,E,J,K,C,F,L,M,G,N,O</p>	 <p>Recorrido H,D,I,B,J,E,K,A,L,F,M,C,N,G,O</p>	 <p>Recorrido H,I,D,J,K,E,B,L,M,F,N,O,G,C,A</p>

RECORRIDO POR EXPANSIÓN

También conocida como recorrido por niveles o en amplitud. Se explora el árbol nivel a nivel, de izquierda a derecha y del primero al último

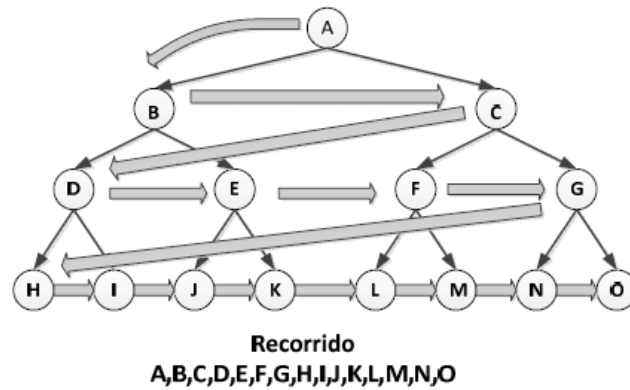


Imagen 2: Recorrido de un árbol mediante expansión

ÁRBOLES BINARIOS

Un árbol binario es un árbol de grado dos, esto es, cada nodo puede tener dos, uno o ningún hijo. En los árboles binarios se distingue entre el subárbol izquierdo y el subárbol derecho de cada nodo.

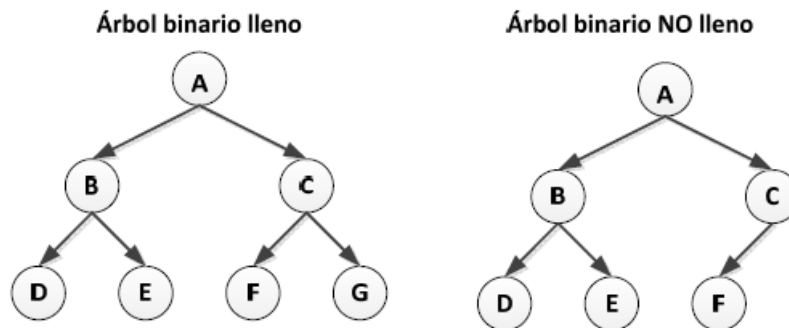
Se puede definir al árbol binario como un conjunto finito de nodos (≥ 0), tal que:

- 1- Si $=0$, el árbol está vacío
- 2- Si >0
 - a. Existe un nodo raíz
 - b. El resto de los nodos se reparte entre dos árboles binarios, que se conocen como subárbol izquierdo y subárbol derecho de la raíz.

A continuación, se describen algunas características de los árboles binarios.

ÁRBOL BINARIO LLENO

Es un árbol binario lleno si cada nodo es de grado cero o dos. O bien, si es un árbol binario de profundidad que tiene $2^n - 1$ nodos (es el número máximo de nodos).



ÁRBOL BINARIO COMPLETO

Un árbol binario es completo si todos los nodos de grado cero o uno están en los dos últimos niveles, de forma que las hojas del último nivel ocupan las posiciones más a la izquierda de dicho nivel. En un árbol binario completo de altura h , todos los niveles, excepto posiblemente el nivel h están completamente llenos. Si un árbol está lleno también está completo.

APLICACIONES DE LOS ÁRBOLES BINARIOS

Una aplicación es la representación de otro tipo de árboles. Esto es importante porque resulta más complejo manipular nodos de grado variable (número variable de relaciones) que nodos de grado fijo. Entonces es posible establecer una relación de equivalencia entre cualquier árbol no binario y un árbol binario, es decir obtener un árbol binario equivalente.

Otra aplicación de los árboles binarios es hallar soluciones a problemas cuyas estructuras son binarias, por ejemplo, las expresiones aritméticas y lógicas.

Actividad 1. Ejercicios del Manual

Para este apartado se pidió codificar el algoritmo de árboles presentado en el manual de prácticas, para ello a continuación se presenta la descripción y salida del programa.

CONOCIMIENTOS PREVIOS

La programación orientada a objetos es un paradigma donde todo se lleva a cabo mediante la abstracción de objetos o modelos a través de lo que se conoce como clases, donde además influyen otras características más que a continuación serán explicadas:

Clases: Las clases son modelos o plantillas sobre los cuales se construirán objetos, estos están formados por atributos y métodos. En Python una clase se define con la palabra reservada `class`.

Atributos: Son características propias de un objeto y modifican el estado de este, existen principalmente 2 tipos de variables o atributos las de tipo instancia y las de tipo clase o global.

Métodos: Los métodos o también conocidos como funciones son aquellos encargados de llevar acciones propias de los objetos.

DIAGRAMA DE CLASES - UML

Para llevar a cabo el algoritmo de árboles en Python fue necesario llevar a cabo el uso de clases dentro de este, es por ello que, mediante los conocimientos previos de POO, se tuvo primero que llevar a cabo un diagrama UML de clases para poder considerar los requisitos que debía tener el programa, a continuación, se muestra el diagrama:

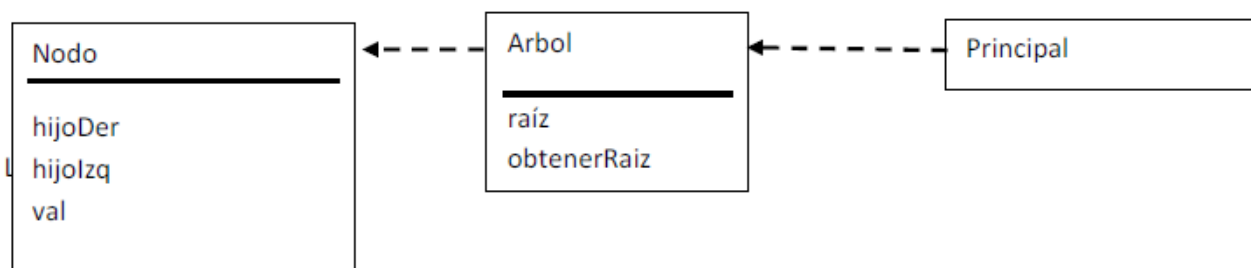


Imagen 3: Diagramas UML – Diagrama de Clases del código previo del algoritmo DFS en Python

ANÁLISIS DEL CÓDIGO

En este ejercicio se dio previamente el código de árbol, a continuación, se explican brevemente cada uno de los métodos:

CLASE NODO

Esta clase sirve para la construcción de objetos del tipo nodo donde las variables que tiene para llevar a cabo el algoritmo de árbol son (Esto dentro del método constructor de la clase):

- 1) Un nodo hijo para la parte derecha
- 2) Un nodo hijo para la parte izquierda
- 3) Un variable que sirviera para guardar el valor del nodo

CLASE ÁRBOL

Dentro de esta clase solamente hay dos métodos:

Método constructor: Solamente se da la asignación de un nodo como raíz dentro del árbol (Esto con el fin de saber de dónde parte o comienza el árbol)

Método obtener raíz: Es un método que regresa el nodo raíz del árbol.

MÉTODOS

1] Método agregar

Es el método que se encarga de dividir cual será el nodo raíz y cual serán todos los demás nodos, para ello simplemente se declara mediante el método raíz el nodo raíz en caso de que ya existe un nodo raíz (Caso del else) se hace llamada al método agregarNodo

2] Método agregarNodo

Es realmente el método encargado de la asignación de los nodos tanto derechos como izquierdos (Es un árbol del tipo binario) , a través de simplemente condiciones if y else

NOTA: Como se puede ver en la siguiente captura de pantalla la asignación de valores se da a través del concepto de nodo derecho e izquierdo para de esa forma decir análogamente si estará en la parte izquierda o derecha.

```
if(val < nodo.val):
    #Si hay hijo izquierdo
    if(nodo.hijoIzq != None):
        self.agregarNodo(val, nodo.hijoIzq)
    else:
        #Si no hay hijo izquierdo se crea un nodo con el valor
        nodo.hijoIzq = Nodo(val)
    #Si el valor a agregar es mayor al valor que tiene el nodo actual
    #Se revisa hijo derecho
    else:
        if(nodo.hijoDer != None):
            self.agregarNodo(val, nodo.hijoDer)
        else:
            nodo.hijoDer = Nodo(val)
```

Imagen 4: Líneas de Código encargadas de la asignación de los nodos derecho e izquierdo.

3] Método preorden y Método imprimepreorden

Son los métodos encargados de llevar la impresión y recorrido del árbol, lo primero que se puede destacar es que realmente el método imprimepreOrden es el que llama al método preorden el cual realmente es el encargado de las impresiones.

MÉTODOS DE BÚSQUEDA

Por último, en el apartado 2 de la práctica se pidió realizar la búsqueda de elementos dentro del árbol es por ello lo primero que se realizó fue llevar a cabo la búsqueda mediante 2 métodos:

1] Método búsqueda

Es el que simplemente divide dentro de la búsqueda a los nodos hijos del nodo padre obviamente en caso de que el nodo que se está buscando sea el padre por ello tiene el apartado de la condición if y else.

2] Método depthSearch

Como lo dice su nombre, realmente es el método encargado de la búsqueda dentro de las ramas y hojas del árbol, para ello lo que se hace es declarar los dos casos dentro de cada recorrido, por un lado, si es mayor o es menor (Esto con el fin de buscar dentro de los nodos derechos e izquierdos)

NOTA: Por otro lado, para el uso de todos los anteriores métodos se dio uso del **método main** donde a su vez se declaró un árbol y la inserción de nodos dentro de este.

SALIDA DEL PROGRAMA

A continuación, se muestran la salida del programa empleando el código dado en la práctica y a su vez los métodos que se tuvieron que llevar a cabo para hacer la búsqueda de elementos del siguiente árbol binario.

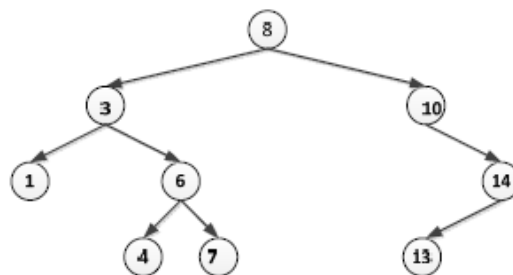


Imagen 5: Esquema del Árbol Binario que será empleado para probar los algoritmos previos.

```
In [6]: runfile('C:/Users/pon_c/OneDrive/Documentos/
Universidad/3º Semestre/EDA II/Prácticas_LAB/8_practice/
Manual_codes/Arbol.py', wdir='C:/Users/pon_c/OneDrive/
Documentos/Universidad/3º Semestre/EDA II/Prácticas_LAB/
8_practice/Manual_codes')
Arbol Binario
8
3
1
6
4
7
10
14
13
Búsqueda en el árbol Binario
[ 3 ] si está en el árbol
[ 19 ] NO está en el árbol
```

Imagen 6: Salida del programa empleando el algoritmo de árbol hecho en Python, En la última parte a su vez se muestra la salida del método de búsqueda de elementos dentro del árbol

ACTIVIDAD 1. IMPLEMENTACIÓN DE ÁRBOL BINARIO

1.A Compilar y ejecutar proyecto

Para este apartado simplemente se creó un proyecto en NetBeans donde la clase principal “Murrieta,Valdepino_practica9” la cual a través de su método main fuera la encargada de llamar a a través de instancias o directamente mediante métodos las distintas clases que a continuación se describirán:

DEFINICIÓN DE CLASES:

1] CLASE ArbolBin

Es la clase dedica a los árboles binarios, primero y como variable global se declaró una clase del tipo nodo (Mencionada como root).

MÉTODOS CONSTRUCTORES

Posteriormente y por convención se declararon los constructores respectivos de la clase donde:

- 1) El primero no requiere ningún parámetro y se define un nodo en su forma nula.
- 2) El segundo constructor se le pasa un valor entero el cual es directamente asociado y pasado al valor del nodo raíz.
- 3) El último a diferencia del método anterior en vez de pasarse el valor del nodo raíz, directamente se le pasa directamente un nodo

MÉTODOS DADOS (Los de la práctica)

1) Visitados

Es un método realmente auxiliar el cual simplemente se encarga de sacar o dar como salida el mensaje de si se ha visitado o no un nodo, esto es a través de pasarle un nodo.

NOTA: Es un método del tipo void debido a que no se cambia nada con este método solo es para impresión (Salida).

2) BFS

Al igual que en práctica anteriores (Las de Grafos) a través del método o algoritmo de recorrido de grafos BFS (breadth First) es como se llevó a cabo el recorrido del árbol en esta práctica

Primero lo que se consideró fue hacer un método nuevamente del tipo void en el cual no se le pasa nada (Esto debido a que directamente el valor del nodo raíz es una variable global), por lo que inmediatamente se asignó el nodo a el nodo “r”.

NOTA: Es importante destacar que en este punto es cuando a través de una lista ligada (LinkedList) del tipo cola o Queue de valores nodo es como se llevó a cabo toda la lógica del programa

Posteriormente, y dentro de un ciclo while es como se fue recorriendo la lista previamente mencionada, donde a través del método “visit” – visitados es como se determinaba si ya se había visitado o no un nodo de la lista.

Recordemos que este algoritmo se hace a través de la expansión del grafo (Que en este caso es un árbol)

2] CLASE Nodo

Como se mencionó previamente, un árbol está constituido de varios nodos los cuales a su vez tienen un valor asignado, además y como bien se sabe en esta estructura se debe considerar algunas consideraciones como el caso donde cada nodo hijo debe tener un nodo izquierdo, derecho y padre.

Dicho lo anterior, fue como se declararon las siguientes variables globales:

1. Una variable entera que guardara el valor del nodo
2. Un nodo dedicado al nodo izquierdo del padre
3. Un nodo dedicado al nodo derecho del padre
4. Un nodo padre el cual sirve para diferenciar de cada uno de sus respectivos hijos

También se declaró un valor extra que posteriormente se pide en la práctica el cual es la altura del nodo.

NOTA: Cabe destacar que en este apartado no se hace la diferencia entre si es un nodo hijo o padre, cabe destacar también que realmente solo el nodo raíz es el único que no tiene padre por lo que directamente no fue necesario hacer esta consideración

Dicho lo anterior, a través de 3 métodos constructores es como se pudo realizar la instanciación de esta clase:

- 1) El primer constructor simplemente declara los 3 nodos mencionados anteriormente en null.
- 2) El segundo constructor a través del valor entero asigna directamente este al valor del nodo mientras que los nodos hijos se declaran nulos.
- 3) El tercer constructor a través de un valor entero y de pasarle 2 nodos es como hace la asignación del valor del nodo que se trata y a su vez del nodo hijo derecho e izquierdo.

Por otro lado, y como convención y aplicación de conocimiento de POO, se realizó los respectivos setters de cada una de las variables globales

NOTA: Nuevamente en este caso se agregó otro setter dedicado a la variable global el tipo nodo para el nodo padre ya que posteriormente será necesaria para la práctica.

3] CLASE practica8 (La de la prueba o menú)

Esta clase realmente es la encargada de hacer la instanciación de los árboles de ejemplo (Propuestos por el profesor) , y se lleva a cabo de simplemente un método main en el cual se hizo la asignación y declaración de los elementos necesarios de cada uno de los árboles.

A continuación, se muestra el diagrama o esquema del árbol propuesto por el profesor:

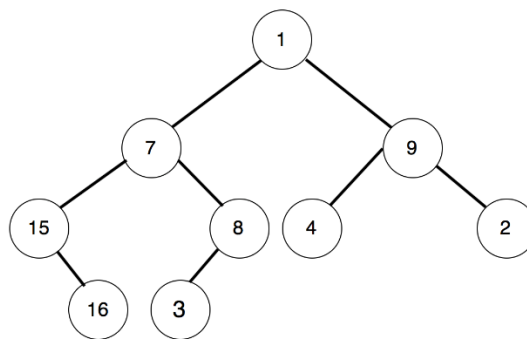


Imagen 7: Esquema del árbol propuesto en para esta actividad

1.B Método de búsqueda para la clase Árbol Binario

Para este apartado se pidió agregar un método para realizar búsqueda de elementos dentro de la clase arbolBin, a continuación, se describe el método encargado:

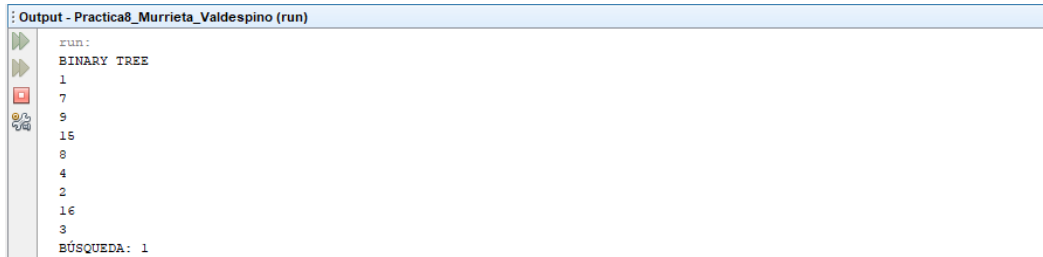
Para este método lo que se realizó fue simplemente mandar el valor del nodo que se quiere buscar, con base al método BFS, simplemente lo único que se realizó fue hacer 2 casos, el primero que es aquel donde el valor se encuentra en el árbol obviamente buscándolo en 3 diferentes casos, ya sea siendo el nodo padre, o alguno de sus nodos hijos, el derecho o izquierdo.

NOTA: Para llevar lo anterior acabo s a través de la restricción de que los nodos no sean nulos y que además la lista donde se encuentran los nodos no esté vacía ya que si es el caso el valor no se encuentra en el árbol.

En caso de que no se encuentre dentro del árbol simplemente se regresa el valor -1.

1.C Salida del programa

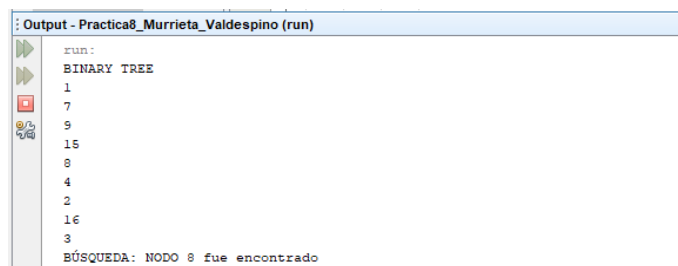
Por último y como comprobación del programa, en este apartado se mostrará la salida de cada uno de los métodos anteriores incluyendo el que se creó para la búsqueda de elementos.



```
run:
BINARY TREE
1
7
9
15
8
4
2
16
3
BÚSQUEDA: 1
```

Imagen 8: Salida del programa donde se muestra en primera instancia el recorrido a través de BFS del árbol binario, posteriormente se indica si es falso o verdadero la búsqueda de un elemento.

NOTA: Se realizó unos pequeños cambios al método de búsqueda a continuación se muestra la salida correspondiente:



```
run:
BINARY TREE
1
7
9
15
8
4
2
16
3
BÚSQUEDA: NODO 8 fue encontrado
```

Imagen 9: Salida del programa que indica si es falso o verdadero la búsqueda de un elemento.

ACTIVIDAD 2. MENÚ DE USUARIO

En este apartado se pidió realizar un menú para el programa donde se consideró las siguientes características:

- 1) Árbol Binario
 - a. Agregar
 - b. Eliminar (Este apartado es por puntos extras)
 - c. BFS
 - d. Notación pre-fija
 - e. Notación In-fija
 - f. Notación post-fija
- 2) Árboles Binarios de Búsqueda
 - a. Agregar
 - b. Eliminar
 - c. Buscar
 - d. Imprimir árbol (BFS)

Dicho lo anterior, a continuación, se muestra la salida correspondiente a este apartado:

```
Output - Practica8_Murrieta_Valdespino (run)
[MENU]
1)Árbol Binario
2)Árbol Binario de búsqueda
3)Salir
INGRESE OPCIÓN:
1
BINARY TREE
1)Agregar
2)BFS
3)Notación prefija
4)Notación Infija
5)Notación Posfija
6)Salir
INGRESE OPCIÓN:
6
---->
1)Árbol Binario
2)Árbol Binario de búsqueda
3)Salir
INGRESE OPCIÓN:
2
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
|
```

Imagen 10: Salida del programa donde se muestra el correspondiente menú de todo el programa, como se puede apreciar para cada caso se tiene una opción alterna de salida

ACTIVIDAD 3. OPERACIONES DE ÁRBOLES BINARIOS

3.A Desarrollo de los métodos

Para este apartado se pidió realizar cada uno de los métodos del recorrido de árboles dinarios en su notación prefija, infija y postfija, además y como apartado extra se dio la opción de agregar un método de eliminación de nodos para el árbol binario donde se debe preguntar que nodo se va a eliminar.

NOTA: El dato ingresado por el usuario se debe asumir que existe.

Dicho lo anterior, a continuación, se describirán y mostrarán cada uno de los métodos solicitados:

1) Método de notación prefija

Para esta notación se realizaron 2 métodos principalmente, el primero llamado “preorden” el cual solamente llama al segundo método preordenUTIL a través de pasarle el nodo raíz.

Por otro lado, el método preordenUTIL realmente el que lleva toda la lógica del algoritmo, con base a BFS es como se partió para el desarrollo de este método.

A través de una lista ligada de tipo cola es como se llevó a cabo el recorrido del árbol, cabe destacar que para poder llevar el recorrido simplemente se condicionó mediante un ciclo while el recorrido de todo el árbol

La parte esencial del código es realmente el apartado de las condiciones dentro del ciclo while, donde mediante la recursividad es como se lleva a cabo el recorrido tanto por el lado izquierdo y derecho de los nodos hijos de un nodo padre.

NOTA: La parte importante es el marcado de los nodos durante el recorrido en este caso se hace antes de empezar el recorrido de los hijos izquierdos y derechos

2] Método de notación infija

Para esta notación se realizaron 2 métodos principalmente, el primero llamado “inorden” el cual solamente llama al segundo método inordenUTIL a través de pasarle el nodo raíz.

Por otro lado, el método inordenUTIL realmente el que lleva toda la lógica del algoritmo, con base a BFS es como se partió para el desarrollo de este método.

A través de una lista ligada de tipo cola es como se llevó a cabo el recorrido del árbol, cabe destacar que para poder llevar el recorrido simplemente se condicionó mediante un ciclo while el recorrido de todo el árbol

La parte esencial del código es realmente el apartado de las condiciones dentro del ciclo while, donde mediante la recursividad es como se lleva a cabo el recorrido tanto por el lado izquierdo y derecho de los nodos hijos de un nodo padre.

NOTA: La parte importante es el marcado de los nodos durante el recorrido en este caso se hace después de recorrer los hijos izquierdos.

3] Método de notación post-fija

Para esta notación se realizaron 2 métodos principalmente, el primero llamado “postorden” el cual solamente llama al segundo método postordenUTIL a través de pasarle el nodo raíz.

Por otro lado, el método postordenUTIL realmente el que lleva toda la lógica del algoritmo, con base a BFS es como se partió para el desarrollo de este método.

A través de una lista ligada de tipo cola es como se llevó a cabo el recorrido del árbol, cabe destacar que para poder llevar el recorrido simplemente se condicionó mediante un ciclo while el recorrido de todo el árbol

La parte esencial del código es realmente el apartado de las condiciones dentro del ciclo while, donde mediante la recursividad es como se lleva a cabo el recorrido tanto por el lado izquierdo y derecho de los nodos hijos de un nodo padre.

NOTA: La parte importante es el marcado de los nodos durante el recorrido en este caso se hace una vez que se recorre tanto parte izquierda como derecha.

3.B Salida de los métodos correspondientes

Por último y como comprobación del programa, en este apartado se mostrará la salida de cada uno de los métodos anteriores de recorrido de árboles:

```
Output - Practica8_Murrieta_Valdespino (run)
RECORRIDOS:
PRE-ORDEN
1
7
15
16
8
3
9
4
2
IN-ORDEN
15
16
7
3
8
1
4
9
2
POST-ORDEN
15
16
3
8
7
4
2
9
1
```

Imagen 11: Salida del programa donde se muestran las diferentes notaciones de un mismo árbol preorden, inorden y postorden.

ACTIVIDAD 4. OPERACIONES DE ÁRBOLES BINARIOS DE BÚSQUEDA

4.A Desarrollo de los métodos

Para esta actividad se partió de la clase previamente desarrollada de árboles binario donde a través de mínimos cambios (Que serán mencionados) se desarrollaron los métodos siguientes:

- 1) Un método para eliminar elementos del árbol binario de búsqueda
- 2) Un método encargado de verificar la altura, número de nodos y si es o no balanceado.

A continuación, se explican y muestran los métodos correspondientes de cada uno de los métodos mencionados y de las modificaciones que se realizó a la clase:

NOTA IMPORTANTE:

Para el desarrollo de esta clase y de sus respectivos métodos se partió de la clase de árbol binario, donde los siguientes métodos fueron directamente utilizados sin ninguna modificación

1. Constructores: En sus 3 versiones previamente mencionadas.
2. Método BFS (breadthFirst) y su método auxiliar visit (visitado) // Asociados con el recorrido e impresión
3. Método de búsqueda
4. Método de add (Agregado de nodos)

MÉTODO DE BÚSQUEDA ALTERNO

1] Métodos de búsqueda de nodos

Realmente este método contiene la misma lógica y líneas de código que los métodos previamente mencionados para la búsqueda de un valor dentro del árbol (O sea los métodos búsqueda y busquedaUTIL), el único cambio realmente es el tipo de método y lo que se va a realizar para la búsqueda ya que en este caso en vez de buscar el valor del nodo se está buscando directamente el nodo asociado. La razón de este método es para poder llevar a cabo la eliminación de nodos dentro del árbol.

NOTA: Realmente este método es innecesario, pero lamentablemente el poder utilizar el otro método realmente no fue posible.

MÉTODOS PARA LA CANTIDAD DE NODOS

1] Métodos de cantidad de nodos

Este método que realmente está totalmente basado en BFS donde las únicas modificaciones que tiene es un valor entero el cual va incrementando con forme se va recorriendo el árbol, es por ello que para el apartado del recorrido dentro del ciclo while para cada iteración simplemente se iba incrementa en uno el valor de auxiliar (n) el cuál almacenará la cantidad de nodos.

Cabe destacar que en dado caso de que no entre en el ciclo while al no tener nodos simplemente se regresa 0.

MÉTODOS PARA DETERMINAR LA ALTURA Y SI ES O NO BALANCEADO

1] Métodos de altura y Método alturaUTIL

El primer nodo es el encargado de llamar al método alturaUTIL el cual es realmente el encargado de llevar a cabo la determinación de la altura del método para este caso lo que realmente se debe realizar es el recorrido tanto del lado derecho como izquierdo del árbol considerando obviamente que cada vez que se hace una revisión se debe aumentar el valor de la profundidad (altura) este método realmente está basado en BFS solamente que no se considera el descarte de los recorridos.

NOTA IMPORTANTE:

Al igual que el método de búsqueda, se tuvo que realizar nuevamente estos 2 métodos para poder llevar de manera auxiliar el apartado de la eliminación de nodos, es por ello que en vez de buscar de considerar los valores se realiza respecto a los nodos.

NOTA: Realmente los métodos alturaNODO y alturaNUTIL son métodos no encargados para la determinación de la altura sino para auxiliar la eliminación de nodos

NOTA 2: Estos 2 métodos realmente necesitan de los 2 métodos anteriores debido a que estos necesitan de la altura del árbol.

MÉTODOS PARA DETERMINAR BALANCEADO

1] Métodos balanceado

Es un método de tipo booleano donde realmente a través de 3 variables enteras destinadas a la altura del lado derecho, izquierdo y otro para la diferencia (Este valor es realmente para determinar si es o no balanceado), es por ello por lo que a través del método previo alturaNODO es como se lleva la asignación de los valores de tanto izquierdo como derecho, ya para determinar si es o no balanceado es como a través de la diferencia de izquierdo y derecho es como se puede determinar esta propiedad.

Si son diferentes regresamos un falso y si son iguales se regresa verdadero

MÉTODOS PARA ELIMINAR NODOS

NOTA: Para la eliminación de nodos se tuvo que realizar un algoritmo a través de la metodología bottom up donde se tuvo que partir el problema desde casos particulares de la eliminación y asignación de cada nodo.

1] Método de eliminación

El método principal de eliminación es el encargado de llevar a cabo primero la búsqueda del nodo y posteriormente de los 3 casos que existen al eliminar un nodo:

- 1) El nodo que se quiere borrar no tiene ningún nodo hijo
- 2) El nodo que se quiere borrar solamente tiene un nodo hijo
- 3) El nodo que se quiere borrar tiene tanto el nodo izquierdo y derecho

A continuación, se muestra la forma en que se abordaron los 3 casos anteriores:

```
241 | if(n!=null){
242 |     if(n.isq==null&& n.der==null){//Si no tiene hijos
243 |         eliminarWITHOUT(n);
244 |     }else if (n.isq!=null&& n.der==null){ //si tiene un hijo
245 |         eliminarWITHONE(n);
246 |     }else if (n.isq==null&& n.der!=null){
247 |         eliminarWITHONE(n);
248 |     }else if(n.isq!=null&& n.der!=null){ //si tiene dos hijos
249 |         eliminarWITHTWO(n);
```

Imagen 13: Líneas encargadas de la distribución de los 3 casos anteriores.

2] Métodos eliminarWITHONE, eliminarWITHTWO y eliminarWITHOU

Son los métodos destinados a cada uno de los casos que existen para los nodos, para cada uno de estos métodos lo que se realizó fue partir a través del recorrido del árbol para de esa forma poder determinar los nodos hijos que hay del nodo que se quiere realizar.

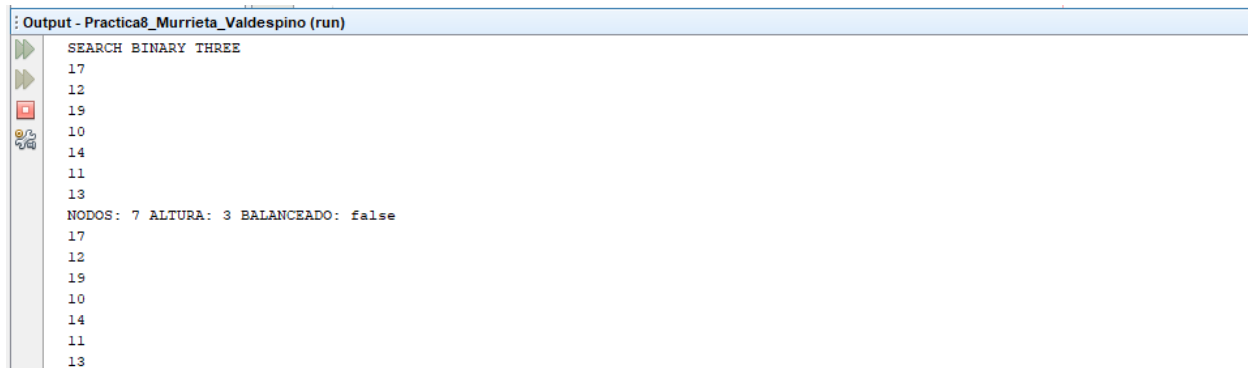
Realmente el primer caso que es donde no hay ningún nodo es el más sencillo ya que simplemente se borra el nodo sin que realmente haya necesidad de volver asignar los nodos hijos.

Sin, embargo, para los otros 2 casos necesariamente se debe realizar el apartado mencionado anteriormente, es por ello por lo que primero se debe encontrar y determinar los nodos hijos y posteriormente hacer la asignación.

NOTA: La eliminación en el caso de que haya 2 nodos hijos, se tuvo que realizar otro método externo para ese caso.

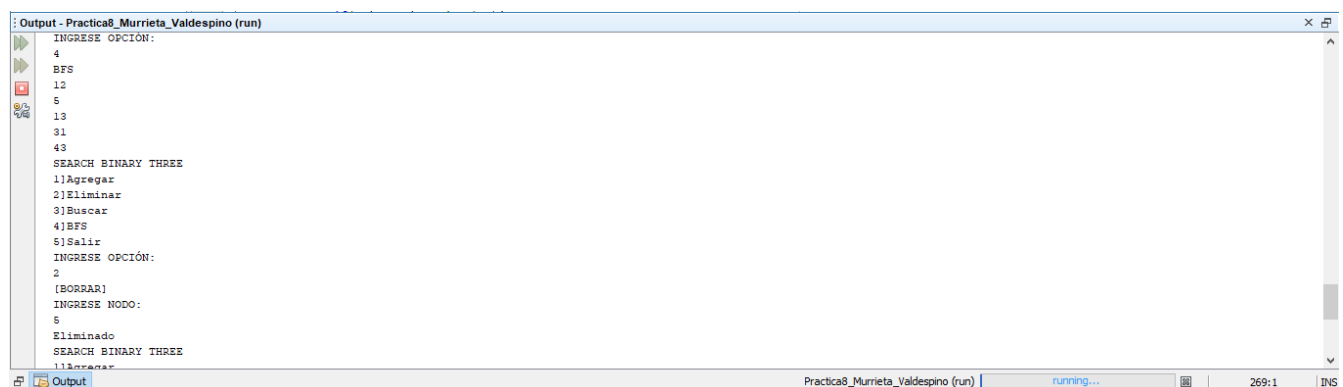
4.B Salida del programa

Por último y como comprobación del programa, en este apartado se mostrará la salida de cada uno de los métodos anteriores:



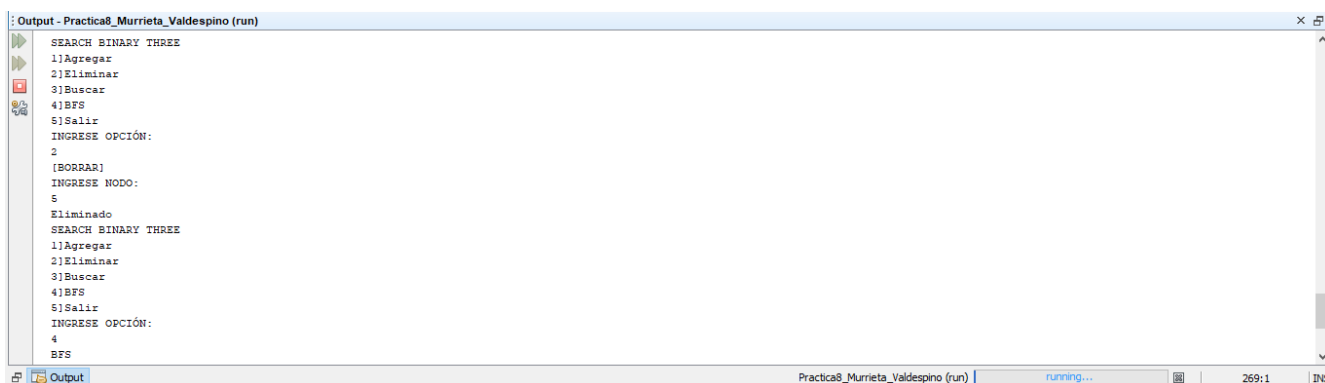
```
Output - Practica8_Murrieta_Valdespino (run)
SEARCH BINARY THREE
17
12
19
10
14
11
13
NODOS: 7 ALTURA: 3 BALANCEADO: false
```

Imagen 13: Salida del programa donde se muestran los métodos anteriores, en primera instancia la impresión del árbol binario de búsqueda, posteriormente se muestra la cantidad de nodos, la altura correspondiente del árbol y por último si es o no balanceado.



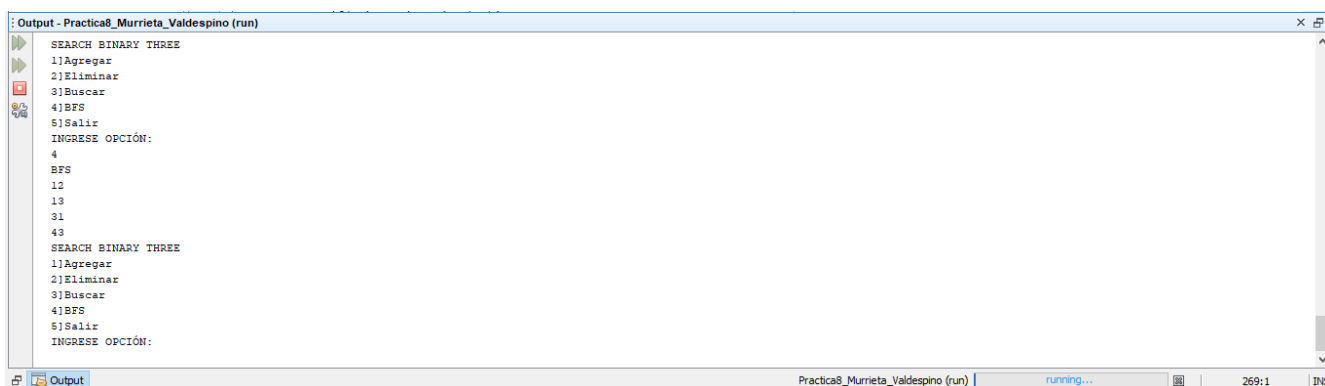
```
Output - Practica8_Murrieta_Valdespino (run)
INGRESE OPCIÓN:
4
BFS
12
5
13
31
49
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
2
(BORRAR)
INGRESE NODO:
5
Eliminado
SEARCH BINARY THREE
1)Agregar
```

Imagen 14: Salida del programa donde se muestra a través de BFS el árbol que se ha creado.



```
Output - Practica8_Murrieta_Valdespino (run)
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
2
[BORRAR]
INGRESE NODO:
5
Eliminado
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
4
BFS
```

Imagen 15: Salida del programa donde se muestra que ha sido borrado el elemento 5



```
Output - Practica8_Murrieta_Valdespino (run)
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
4
BFS
12
13
31
43
SEARCH BINARY THREE
1)Agregar
2)Eliminar
3)Buscar
4)BFS
5)Salir
INGRESE OPCIÓN:
```

Imagen 16: Salida del programa donde se muestra el nuevo árbol a través de BFS (Obviamente considerando que ha sido borrado el elemento anterior)

CONCLUSIONES

MURRIETA VILLEGAS ALFONSO

Sin duda alguna, a lo largo de esta práctica se fueron recopilando y empleando todos los conocimientos previamente vistos en las 3 últimas prácticas como son el recorrido de Grafos mediante BFS, la creación de árboles y sobre todo la implementación de árboles binarios donde además se consideraron las características correspondientes para los árboles binarios de búsqueda, sin duda alguna una de las prácticas que realmente han implicado muchísimos conocimientos previos.

Por otro lado, el desarrollo del árbol binario realmente no resultó complicado salvo por el desarrollo no logrado del método de eliminación y es que realmente no fue posible tanto por el tiempo como por la estructura el poder desarrollar este apartado, sin embargo, lo demás no resultó nada complicado el desarrollo de BFS y de los recorridos en sus distintas notaciones no fue ningún problema.

Como apartado más pesado fue realmente el de árboles binarios de búsqueda, en este caso cabe destacar que fue totalmente basado en la clase de árbol binario como fue mencionado en el apartado de métodos, sin embargo, el poder llevar a cabo del desarrollo del método de eliminación resultó algo complejo pero no imposible ya que en este caso la ventaja era simplemente a través de las condiciones que tiene el mismo árbol fue como se pudo desarrollar la eliminación de nodos incluso cuando este tenía 2 hijos.

Por último, sin duda esta ha sido la práctica más complicada hasta el momento el realmente ver que tan difícil ha resultado esta práctica solamente me hace pensar en que tan difícil será llevar a cabo lo que se pedirá para la práctica.

VALDESPINO MENDIETA JOAQUÍN

En conclusión de esta práctica se han logrado los objetivos, se ha podido comprender la estructura de un árbol binario y un árbol binario de búsqueda, la gran ventaja que otorga un árbol de búsqueda es como su nombre lo menciona, permite una búsqueda de manera eficiente, ya que la misma estructura favorece y delimita el rango de búsqueda, a comparación de un árbol binario, sin embargo la lógica para construir un árbol binario de búsqueda es más compleja de armar ya que tiene una estructura definida y ese proceso para elaborarlo requirió un análisis más profundo para una implementación, este tema de árboles está ligado con grafos por un lado un árbol es un caso de un grafo, por otro lado y a los recorridos sobre grafos ayudó a implementar de manera rápida los recorridos sobre árboles.

Por último, un árbol binario nos permite manejar datos de manera más simple, sin embargo la implementación requiere un cierto nivel de abstracción del concepto y estructura, además del uso de conceptos previos.

REFERENCIAS

Mark J. Guzdial, Barbara Ericson. (2015). *Introducción a la computación y programación con Python*. 3ra Edición. Madrid España.

Bradley N. Miller y David L. Ranum, Franklin, Beedle & Associates. (2011). *Problem Solving with Algorithms and Data Structures using Python*. Segunda Edición.

Elba Karen Saenz García. (2017). *Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos II*. UNAM, Facultad de Ingeniería.