

INSTITUTO TECNOLÓGICO DE MEXICO

CAMPUS ORIZABA

NOMBRE DE LA ASIGNATURA:
ESTRUCTURA DE DATOS

TEMA:
REPORTE DE PRACTICA TEMA 4 ESTRUCTURAS NO LINEALES

NOMBRE DE LOS INTEGRANTES:
HERNÁNDEZ DÍAZ ARIEL - 21010195
PELACIOS MENDEZ XIMENA MONSERRAT – 21010209

NOMBRE DE LA PROFESORA:
MARIA JACINTA MARTINEZ CASTILLO

HORARIO OFICIAL DE LA CLASE:
15:00 – 16:00 HRS

FECHA DE ENTREGA:
02 DE JUNIO DE 2023

INTRODUCCIÓN

En esta unidad se debe conocer, identificar y aplicar las estructuras no lineales en la solución de problemas del mundo real. Para poder aprender de ello se necesita consultar en las fuentes bibliográficas la terminología sobre árboles como también practicar ejercicios y buscar información ayudara para su mayor comprensión. Utilizando un lenguaje de programación podemos implementar las operaciones básicas (insertar, alturas, buscar) en un árbol binario de búsqueda, así como los recorridos en PreOrden, InOrden y PostOrden.

En una estructura lineal, cada elemento sólo puede ir enlazado al siguiente o al anterior. A las estructuras de datos no lineales se les llama también estructuras de datos multienlazadas. Cada elemento puede estar enlazado a cualquier otro componente. Se trata de estructuras de datos en las que cada elemento puede tener varios sucesores y/o varios predecesores. En estas estructuras cada elemento puede tener varios elementos “siguientes”, lo cual introduce el concepto de estructuras de ramificación. Estas estructuras de datos de ramificación son llamadas grafos y árboles

En si podemos decir que aprenderemos conceptos y aplicaciones de por ejemplo: Concepto de árbol y su clasificación de árboles al igual las operaciones básicas sobre árboles binarios que sin duda son herramientas que sirven y servirán en nuestra vida escolar.

COMPETENCIA(S) A DESARROLLAR

Comprende y aplica estructuras no lineales para la solución de problemas.

MARCO TEORICO

DEFINICION DE ÁRBOL

Un árbol es un conjunto de n registros ($n > 0$, árbol vacío no está definido), tales que, hay un registro especial llamado RAÍZ y los demás registros están particionados en conjuntos disjuntos, cada uno de los cuales es un árbol.

Un árbol es una estructura recursiva por definición, además cada registro que tenga hijos se podrá considerar como la raíz de un sub-árbol. Las ramificaciones de cada nodo o registro suelen llamarse hijos y los nodos de los cuales salen las ramificaciones se llaman padres. Los registros que tienen un mismo padre se denominan hermanos.

En ciencias de la informática, un árbol es una estructura de datos ampliamente usada que imita la forma de un árbol (un conjunto de nodos conectados). Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él. Se dice que un nodo a es padre de un nodo b si existe un enlace desde a hasta b (en ese caso, también decimos que b es hijo de a). Solo que puede haber un único nodo sin padres, que llamaremos raíz. Un nodo que no tiene hijos se conoce como hoja. Los demás nodos (tienen padre y uno o varios hijos) se les conoce como rama.

4.1.2 CLASIFICACIÓN DE ÁRBOLES

Árboles Binarios

Ejemplo de árbol (binario) 1

Un árbol binario es una estructura de datos en la cual cada nodo siempre tiene un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno. Usos comunes de los árboles binarios son los árboles binarios de búsqueda, los montículos binarios y Codificación de Huffman.

Tipos de árboles binarios

Un árbol binario es un árbol con raíz en el que cada nodo tiene como máximo dos hijos.

Un árbol binario lleno es un árbol en el que cada nodo tiene cero o dos hijos.

Un árbol binario perfecto es un árbol binario lleno en el que todas las hojas (vértices con cero hijos) están a la misma profundidad (distancia desde la raíz, también llamada altura).

A veces un árbol binario perfecto es denominado árbol binario completo. Otros definen un árbol binario completo como un árbol binario lleno en el que todas las hojas están a profundidad n o $n-1$, para alguna n .

Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario cada nodo puede tener cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.

Árbol de búsqueda binario auto-balanceable

En ciencias de la computación, un árbol binario de búsqueda auto-balanceable o equilibrado es un árbol binario de búsqueda que intenta mantener su altura, o el número de niveles de nodos bajo la raíz, tan pequeños como sea posible en todo momento, automáticamente. Esto es importante, ya que muchas operaciones en un árbol de búsqueda binaria tardan un tiempo proporcional a la altura del árbol, y los árboles binarios de búsqueda ordinarios pueden tomar alturas muy grandes en situaciones normales, como cuando las claves son insertadas en orden. Mantener baja la altura se consigue habitualmente realizando transformaciones en el árbol, como la rotación de árboles, en momentos clave.

Para algunas implementaciones estos tiempos son el peor caso, mientras que para otras están amortizados.

Estructuras de datos populares que implementan este tipo de árbol:

Árbol AVL

Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

- es AVL
- es AVL

Árboles Rojo-Negro

Un árbol rojo-negro es un tipo abstracto de datos. Concretamente, es un árbol binario de búsqueda equilibrado, una estructura de datos utilizada en informática y ciencias de la computación. Es complejo, pero tiene un buen peor caso de tiempo de ejecución para sus operaciones y es eficiente en la práctica. Puede buscar, insertar y borrar en un tiempo $O(\log n)$, donde n es el número de elementos del árbol.

Un árbol rojo-negro es un árbol binario de búsqueda en el que cada nodo tiene un atributo de color cuyo valor es rojo o negro. En adelante, se dice que un nodo es rojo o negro haciendo referencia a dicho atributo.

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

- Todo nodo es o bien rojo o negro.
- La raíz es negra.
- Todas las hojas (NIL) son negras.
- Todo nodo rojo debe tener dos nodos hijos negros.
- Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

Árbol AA

Un árbol AA es un tipo de árbol binario de búsqueda auto-balanceable utilizado para almacenar y recuperar información ordenada de manera eficiente.

Los árboles AA son una variación del árbol rojo-negro, que a su vez es una mejora del árbol binario de búsqueda. A diferencia de los árboles rojo-negro, los nodos rojos en un árbol AA sólo pueden añadirse como un hijo derecho. En otras palabras, ningún nodo rojo puede ser un hijo izquierdo. De esta manera se simula un árbol 2-3 en lugar de un árbol 2-3-4, lo que simplifica las operaciones de mantenimiento. Los algoritmos de mantenimiento para un árbol rojo-negro necesitan considerar siete diferentes formas para balancear adecuadamente el árbol.

Árboles Multicamino

Los árboles multicamino o árboles multirrama son estructuras de datos de tipo árbol usadas en computación.

Un árbol multicamino posee un grado g mayor a dos, donde cada nodo de información del árbol tiene un máximo de g hijos.

Sea un árbol de m -caminos A , es un árbol m -caminos si y solo si:

- A está vacío

- Cada nodo de A muestra la siguiente estructura:
[nClaves,Enlace0,Clave1,...,ClavenClaves,EnlacenClaves]

nClaves es el número de valores de clave de un nodo, pudiendo ser: $0 \leq nClaves \leq g-1$

Enlacei, son los enlaces a los subárboles de A, pudiendo ser: $0 \leq i \leq nClaves$

Clavei, son los valores de clave, pudiendo ser: $1 \leq i \leq nClaves$

- $Clavei < Clavei+1$

- Cada valor de clave en el subárbol Enlacei es menor que el valor de Clavei+1

- Los subárboles Enlacei, donde $0 \leq i \leq nClaves$, son también árboles m-caminos.

La principal ventaja de este tipo de árboles consiste en que existen más nodos en un mismo nivel que en los árboles binarios con lo que se consigue que, si el árbol es de búsqueda, los accesos a los nodos sean más rápidos.

Árboles B (Árboles de búsqueda multicamino autobalanceados)

los árboles-B o B-árboles son estructuras de datos de árbol que se encuentran comúnmente en las implementaciones de bases de datos y sistemas de archivos. Son árboles balanceados de búsqueda en los cuales cada nodo puede poseer más de dos hijos. Los árboles B mantienen los datos ordenados y las inserciones y eliminaciones se realizan en tiempo logarítmico amortizado.

La idea tras los árboles-B es que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten.

La idea tras los árboles-B es que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten.

Árbol-B+

Un árbol B+ es un tipo de estructura de datos de árbol, representa una colección de datos ordenados de manera que se permite una inserción y borrado eficientes de elementos. Es un índice, multinivel, dinámico, con un límite máximo y mínimo en el número de claves por nodo. Un árbol B+ es una variación de un árbol B.

En un árbol B+, toda la información se guarda en las hojas. Los nodos internos sólo contienen claves y punteros. Todas las hojas se encuentran en el mismo nivel, que corresponde al más bajo. Los nodos hoja se encuentran unidos entre sí como una lista enlazada para permitir búsqueda secuencial.

Las estructuras de árbol B+ reúnen las siguientes características:

- El número máximo de claves en un registro es llamado el orden del árbol B+.
- El mínimo número de claves por registro es la mitad del máximo número de claves. Por ejemplo, si el orden de un árbol B+ es n , cada nodo (exceptuando la raíz) debe tener entre $n/2$ y n claves.
- El número de claves que pueden ser indexadas usando un árbol B+ está en función del orden del árbol y su altura.

Árbol-B*

Un árbol-B* es una estructura de datos de árbol, una variante de Árbol-B utilizado en los sistemas de ficheros HFS y Reiser4, que requiere que los nodos no raíz estén por lo menos a $2/3$ de ocupación en lugar de $1/2$. Para mantener esto los nodos, en lugar de generar inmediatamente un nodo cuando se llenan, comparten sus claves con el nodo adyacente. Cuando ambos están llenos, entonces los dos nodos se transforman en tres. También requiere que la clave más a la izquierda no sea usada nunca.

No se debe confundir un árbol-B* con un árbol-B+, en el que los nodos hoja del árbol están conectados entre sí a través de una lista enlazada, aumentando el coste de inserción para mejorar la eficiencia en la búsqueda.

4.1.3 OPERACIONES BASICAS CON ARBOLES BINARIOS

Las rotaciones en árboles binarios son operaciones internas comunes utilizadas para mantener el balance perfecto (o casi perfecto) del árbol binario. Un árbol balanceado permite operaciones en tiempo logarítmico.

Las operaciones comunes en árboles son:

- Enumerar todos los elementos.
- Buscar un elemento.
- Dado un nodo, listar los hijos (si los hay).
- Borrar un elemento.
- Eliminar un subárbol (algunas veces llamada podar).
- Añadir un subárbol (algunas veces llamada injertar).
- Encontrar la raíz de cualquier nodo.

Por su parte, la representación puede realizarse de diferentes formas. Las más utilizadas son:

- Representar cada nodo como una variable en el heap, con punteros a sus hijos y a su padre.
- Representar el árbol con un array donde cada elemento es un nodo y las relaciones padre-hijo vienen dadas por la posición del nodo en el array.

4.1.4 APLICACIONES

Los árboles representan las estructuras no lineales y dinámicas de datos más importantes en Computación. Dinámicas porque las estructuras de árbol pueden cambiar durante la ejecución de un programa. No lineales, puesto que a cada elemento del árbol pueden seguirle varios elementos.

La definición de árbol es la siguiente: es una estructura jerárquica aplicada sobre una colección de Elementos u objetos llamados nodos; uno de los cuales es conocido como raíz. Además se crea una relación o parentesco entre los nodos dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etc. Los árboles tienen una gran variedad de aplicaciones. Por ejemplo, se pueden utilizar para representar fórmulas matemáticas, para organizar adecuadamente la información, para construir un árbol genealógico, en la toma de decisiones, para el análisis de circuitos eléctricos y para numerar los capítulos y secciones de un libro.

A los árboles ordenados de grado dos se les conocen como árboles binarios ya que cada nodo del árbol no tendrá más de dos descendientes directos. Las aplicaciones de los árboles binarios son muy variadas ya que se les puede utilizar para representar una estructura en la cual es posible tomar decisiones con dos opciones en distintos puntos.

4.1.5 ÁRBOLES BALANCEADO (AVL)

Definición. Un árbol AVL es un árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1.

La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis).

Recordamos que un árbol binario de búsqueda es un árbol binario en el cual cada nodo cumple con que todos los nodos de su subárbol izquierdo son menores que la raíz y todos los nodos del subárbol derecho son mayores que la raíz.

Recordamos también que el tiempo de las operaciones sobre un árbol binario de búsqueda son $O(\log n)$ promedio, pero el peor caso es $O(n)$, donde n es el número de elementos.

La propiedad de equilibrio que debe cumplir un árbol para ser AVL asegura que la profundidad del árbol sea $O(\log(n))$, por lo que las operaciones sobre estas estructuras no deberán recorrer mucho para hallar el elemento deseado. Como se verá, el tiempo de ejecución de las operaciones sobre estos árboles es, a lo sumo $O(\log(n))$ en el peor caso, donde n es la cantidad de elementos del árbol.

Sin embargo, y como era de esperarse, esta misma propiedad de equilibrio de los árboles AVL implica una dificultad a la hora de insertar o eliminar elementos: estas operaciones pueden no conservar dicha propiedad.

4.2 GRAFOS

Un grafo es una estructura de datos no lineal con la siguiente característica: un nodo puede apuntar a varios y a su vez ser apuntado por otro.

Consiste en un conjunto de nodos o vértices y un conjunto de ejes o aristas. Usualmente se lo denota $G = (V, E)$. Las aristas son pares de vértices $(u, v) \in V \times V$ con $u \neq v$. Si $(u, v) \in E$, se dice que u y v están conectados o son adyacentes. Un trazado o dibujo de un grafo $G = (V, E)$ usando segmentos de 1 recta para las aristas es una asignación de una posición en el plano a cada vértice en V . En las versiones más simples, los vértices son dibujados como puntos y las aristas como segmentos entre vértices. De aquí en más supondremos que el trazado usa segmentos para las aristas, excepto cuando se aclare lo contrario. También se supondrá que el grafo es conexo, ya que si el grafo a trazar no lo es, se puede trazar cada componente conexa por separado.

Los grafos sólo contienen información relacional entre los vértices, así que en principio no tienen ninguna forma "natural" de ser dibujados. Sin embargo, de las infinitas formas de trazar un grafo en el plano, algunas son claramente peores que otras, siempre teniendo en cuenta que el objetivo es visualizar el grafo.

4.2.1 TERMINOLOGÍA DE GRAFOS

Un grafo se compone por un conjunto de V vértices y un conjunto de A aristas. Cada arista se identifica con el par de vértices que une. Los vértices de una arista son entre sí nodos adyacentes.

- Grado de un nodo:
Número de aristas que contiene ese nodo. Si el grado de un nodo es 0, se dice que es un nodo aislado.
- Grado de un grafo:
Número de vértices de ese grafo.
- Camino:
Un camino C de longitud N de un nodo V_1 a un nodo V_2 , se define como la secuencia de nodos por los que hay que pasar para llegar del nodo V_1 a V_2 . La longitud es el número de aristas que comprende el camino. El

camino es cerrado si empieza y termina en el mismo nodo. El camino es simple si todos los nodos de dicho camino son distintos a excepción de los de los extremos que pueden ser iguales.

- Bucles:
Aristas cuyos extremos son idénticos.
- Aristas múltiples:
Dos o más aristas que conectan los mismos nodos

Tipos de grafos:

- Grafo conectado o conexo:
Existe un camino simple entre dos cualquiera de sus nodos.
- Grafo desconectado:
Aquel en que existen nodos que no están unidos por ningún camino
- Grafo dirigido:
Cada arista tiene asignada una dirección (identificada por un par ordenado).
- Grafo no dirigido:
La arista está definida por un par no ordenado.
- Grafo sencillo:
Aquel que no tiene ni bucles ni aristas múltiples
- Grafo múltiple o multígrafo:
Permite la existencia de aristas múltiples o bucles.
- Grafo completo:
Cada nodo del grafo es adyacente a todos los demás.
- Grafo etiquetado con peso ponderado:
Cada arista tiene asociado un valor denominado peso. Se usa para indicar algún criterio de evaluación como la longitud o la importancia de la arista respecto a un parámetro.
- Peso de un camino:
La suma de los pesos de las aristas del camino.

Representación de los grafos:

Existen dos formas de representar un grafo:

Con memoria estática (Matriz adyacente): Matriz de $N \times N$ elementos donde N es el número de nodos del grafo. Cada posición $M(i, j)$ indica si hay una conexión o no entre los nodos que están asociados a la posición i y j de la matriz. Con memoria dinámica. Se utilizan 2 listas enlazadas:

- *Lista de nodos: Formada por todos los vértices.
- *Lista de adyacentes: Contiene las aristas del grafo.

4.2.2 OPERACIONES BASICAS CON GRAFOS

En los grafos, como en todas las estructuras de datos, las dos operaciones básicas son insertar y borrar. En este caso, cada una de ellas se desdobla en dos, para insertar/eliminar vértices e insertar/eliminar aristas.

Insertar vértice

La operación de inserción de un nuevo vértice es una operación muy sencilla, únicamente consiste en añadir una nueva entrada en la tabla de vértices (estructura de datos que almacena los vértices) para el nuevo nodo. A partir de ese momento el grafo tendrá un vértice más, inicialmente aislado, ya que ninguna arista llegará a él.

Insertar arista

Esta operación es también muy sencilla. Cuando se inserte una nueva arista en el grafo, habrá que añadir un nuevo nodo a la lista de adyacencia (lista que almacena los nodos a los que un vértice puede acceder mediante una arista) del nodo origen, así si se añade la arista (A,C), se deberá incluir en la lista de adyacencia de A el vértice C como nuevo destino.

Eliminar vértice

Esta operación es inversa a la inserción de vértice. En este caso el procedimiento a realizar es la eliminación de la tabla de vértices del vértice en sí. A continuación habrá que eliminar las aristas que tuviesen al vértice borrado como origen o destino.

Eliminar arista

Mediante esta operación se borra un arco del grafo. Para llevar a cabo esta acción es necesario eliminar de la lista de adyacencia del nodo origen el nodo correspondiente al nodo destino.

Otras operaciones

Las operaciones adicionales que puede incluir un grafo son muy variadas. Además de las clásicas de búsqueda de un elemento o recorrido del grafo, también podemos encontrarnos con ejecución de algoritmos que busquen caminos más cortos entre dos vértices, o recorridos del grafo que ejecuten alguna operación sobre todos los vértices visitados, por citar algunas operaciones de las más usuales.

RECURSOS, MATERIALES Y EQUIPO

- Computadora
- Java
- Lectura de los materiales de apoyo del tema 4 (AULA PADLET)
- Notas de clase (problemas resueltos en clase y materiales de trabajo de la profesora)

CÓDIGOS EXPLICADOS EN JAVA

```
6
7 public class ArbolBin<T> {
8     private Nodito raiz;
9     public ArbolBin() {
10         raiz=null;
11     }
12     public Nodito getRaiz() {
13         return raiz;
14     }
15     public void setRaiz(Nodito raiz) {
16         this.raiz = raiz;
17     }
18     public boolean arbolVacio() {
19         return raiz==null;
20     }
21     public void VaciarArbol() {
22         raiz=null;
23     }
```

Creamos una clase llamada arbolBin dentro del paquete de árbol binario, de tipo genérico <T> invocando a Nodito al crear la variable raíz para luego crear el método árbol binario inicializando la variable raíz con nullo, poniendo los gets y sets correspondientes, agregamos el método arbolVacio, que nos ayuda a indicar que el árbol se encuentra vacío y vaciarArbol que como indica su nombre nos ayuda a vaciar el árbol.

```

public void InsertarArbol(T info) {
    Nodito p= new Nodito(info); //construye el nodo y almacena lo que tiene info
    if(arbolVacio()) {
        raiz=p;
    }else {
        Nodito padre=buscarPadre(raiz ,p);
        if(p.hashCode()>=padre.hashCode())
            padre.der=p;
        else
            padre.izq=p;
    }
}

```

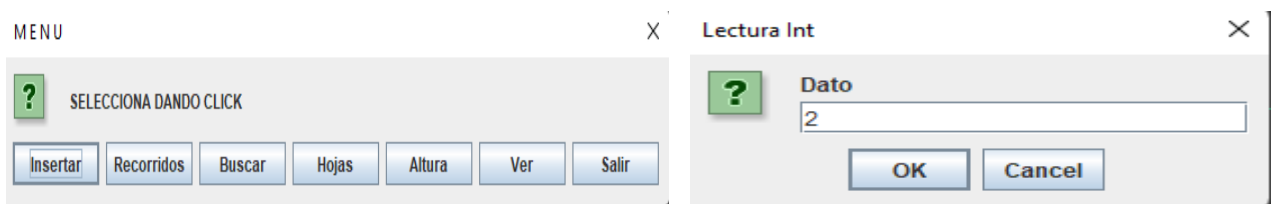
Este método es uno de los más importantes debido que este crea el nodo y lo almacena en una variable de tipo genérico llamada info.

```

public Nodito buscarPadre(Nodito actual,Nodito p) {
    Nodito padre=null;
    while(actual!=null) {
        padre=actual;
        if((int )p.info>=(int)padre.info)
            actual=padre.der;
        else
            actual=padre.izq;
    }
    return padre;
}

```

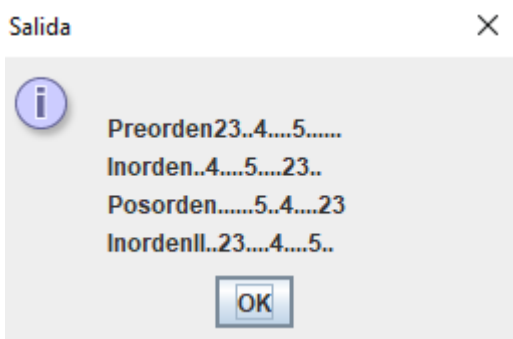
El siguiente método también es fundamental al momento de agregar datos ya que sin él no podríamos insertar los datos.



Los siguientes métodos son los recorridos que podemos tener dentro de los árboles empezando por preorden creando una variable de tipo nodito r para cada uno de los recorridos ,en el recorrido preorden(RID) consiste en empezar por la raíz,luego ir a la izquierda y luego a la derecha hasta recorrer todo el árbol donde raíz significa imprimir el número que se encuentra ahí ,el recorrido inorden(IRD) empieza por la izquierda ,raíz(imprime) y derecha ,el recorrido preorden(IDR) empieza por la izquierda ,derecha y la raíz(imprimir), y por último inorden II (DRI),empezando por la derecha,raíz(imprime),izquierda

```
public String preorden(Nodito r) { // re es la raíz
    if(r!=null) {
        return r.info+".." +preorden(r.izq)+".." +preorden(r.der);
    }
    else return "";
}
public String inorden(Nodito r) {
    if(r!=null) {
        return inorden(r.izq)+".." +r.info+".." +inorden(r.der);
    }
    else return "";
}
public String posorden(Nodito r) {
    if(r!=null) {
        return posorden(r.izq)+".." +posorden(r.der)+".." +r.info;
    }
    else return "";
}
public String inordenII(Nodito r) {
    if(r!=null) {
        return inorden(r.der)+".." +r.info+".." +inorden(r.izq);
    }
    else return "";
}
}
```

Dándonos como resultado en pantalla ,los recorridos según el orden en que se fue recorriendo el árbol.



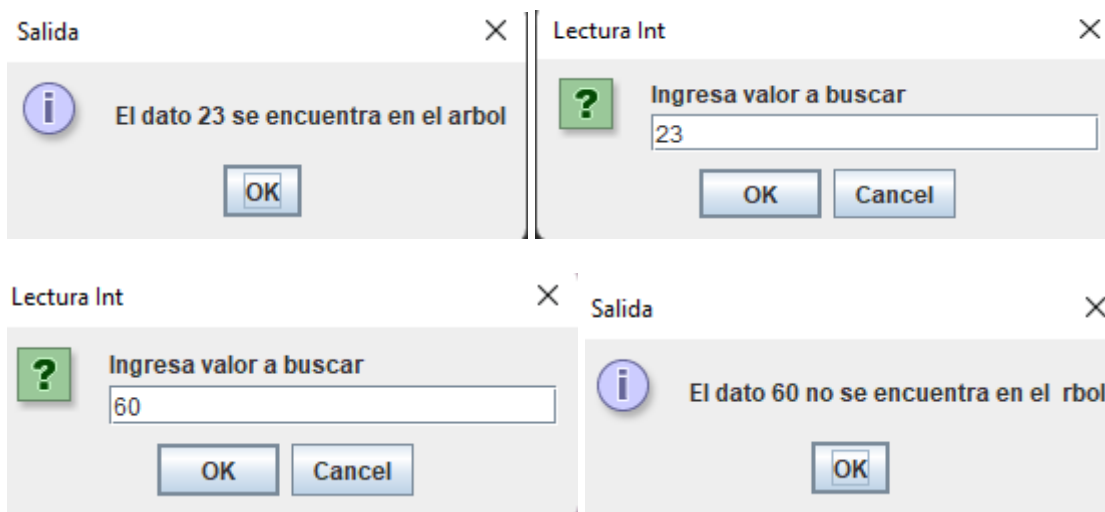
```

public Nodito Buscar(Nodito r, int dato) {
    while (r != null) {
        if (r.getInfo().equals(dato)) {
            return r;
        } else {
            int y = (int) r.info;

            if (dato > y) {
                r = r.getIzq();
            } else {
                r = r.getDer();
            }
        }
    }
    return r;
}

```

El siguiente método buscar nos ayudará a encontrar un dato dentro del árbol recorriendo el árbol de izquierda a derecha hasta encontrar el número que buscamos y si no está simplemente nos dirá que el dato no se encuentra dentro del árbol.

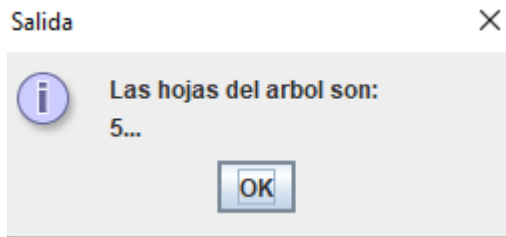


El siguiente método nos imprimirá las hojas que tiene el árbol las cuales son las que no tienen nodos hijos.

```

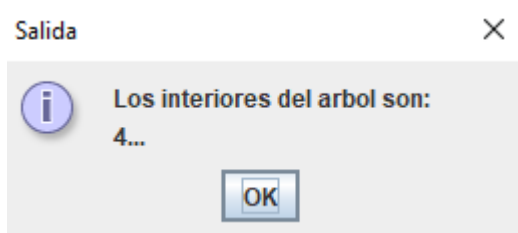
public String imprimirhojas(Nodito r) {
    String cad = "";
    if (r != null) {
        if (r.izq == null && r.der == null) {
            cad = r.info + "...";
        }
        return cad + imprimirhojas(r.izq) + imprimirhojas(r.der);
    } else {
        return "";
    }
}

```



El siguiente método nos imprimirá los nodos interiores los que se encuentran dentro de un árbol y si tiene nodos hijos.

```
public String interiores(Nodito n) {  
    String cad = "";  
    if (n != null) {  
        if (n.izq != null || n.der != null) {  
            if(n.info!=raiz.info)  
                cad = n.info + "...";  
        }  
        return cad +interiores(n.izq) + interiores(n.der);  
    } else {  
        return "";  
    }  
}
```



El método de altura nos ayuda a determinar la altura del árbol haciendo uso de la función math para calcular la altura.

```
public int Altura(Nodito n) {  
    return (n != null) ? Math.max(Altura(n.izq), Altura(n.der)) + 1  
        : 0;  
}
```



Y por último tenemos dos métodos que nos ayudará ve en consola como se ve el árbol mostrando la raíz ,ramas y hojas esto también es posible ocupando JOption para para que fuera más práctico lo hemos dejado con system.out.

```
public static void mostrar(Nodito n, int l) {
    if (n == null) return;

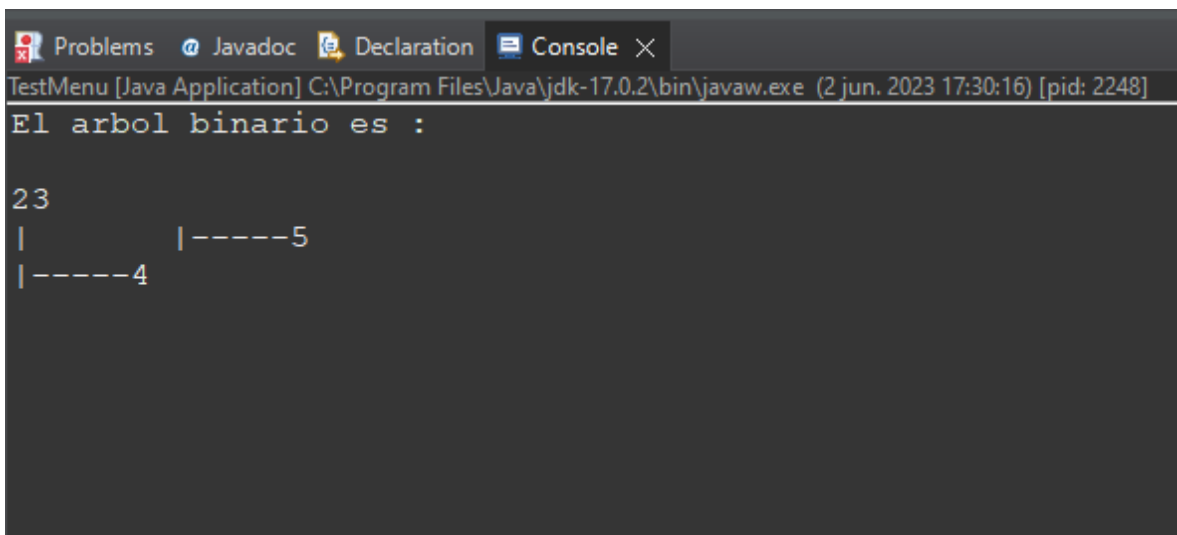
    mostrar(n.der, l + 1);

    if (l != 0) {
        for (int i = 0; i < l - 1; i++) {
            System.out.print("\t");
        }
        System.out.println("|-----" + n.info);
    } else {
        System.out.println(n.info);
    }
    mostrar(n.izq, l + 1);
}

public void imprimirArbol(Nodito nd, String p, boolean ultimo) {
    if (nd == null) {
        return;
    }
    imprimirArbol(nd.der, p + (ultimo ? "    " : "|    "), false);
    System.out.print(p);
    System.out.print(ultimo ? "|--" : "|--");
    System.out.println(nd.info);

    imprimirArbol(nd.izq, p + (ultimo ? "|    " : "    "), true);
}
```

Mostrándonos el árbol de esta manera:



```
Problems  Javadoc  Declaration  Console X
TestMenu [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe (2 jun. 2023 17:30:16) [pid: 2248]
El arbol binario es :

23
|          |-----5
|-----4
```

CODIGOS CON EL MENU

```
public class ArbolBin<T> {  
    private Nodito raiz;  
    public ArbolBin() {  
        raiz=null;  
    }  
    public Nodito getRaiz() {  
        return raiz;  
    }  
    public void setRaiz(Nodito raiz) {  
        this.raiz = raiz;  
    }  
    public boolean arbolVacio() {  
        return raiz==null;  
    }  
    public void VaciarArbol() {  
        raiz=null;  
    }  
    public void InsertarArbol(T info) {  
        Nodito p= new Nodito(info); //construye el nodo y almacena lo que  
tiene info  
        if(arbolVacio()) {  
            raiz=p;  
        }else {  
            Nodito padre=buscarPadre(raiz ,p);  
            if(p.hashCode()>=padre.hashCode())  
                padre.der=p;  
        }  
    }  
}
```

```

        else
            padre.izq=p;
    }
}

public Nodito buscarPadre(Nodito actual,Nodito p) {
    Nodito padre=null;
    while(actual!=null) {
        padre=actual;
        if((int )p.info>=(int)padre.info)
            actual=padre.der;
        else
            actual=padre.izq;
    }
    return padre;
}

public String preorden(Nodito r) {// re es la raiz
    if(r!=null) {
        return r.info+".." +preorden(r.izq)+".." +preorden(r.der);
    }
    else return"";
}

public String inorden(Nodito r) {
    if(r!=null) {
        return inorden(r.izq)+".." +r.info+".." +inorden(r.der);

    }
    else return "";
}

```

```

public String posorden(Nodito r) {
    if(r!=null) {
        return posorden(r.izq)+".."+posorden(r.der)+".."+r.info;
    }
    else return "";
}

public String inordenII(Nodito r) {
    if(r!=null) {
        return inorden(r.der)+".."+r.info+".."+inorden(r.izq);

    }
    else return "";
}

public Nodito Buscar(Nodito r, int dato) {
    while (r != null) {
        if (r.getInfo().equals(dato)) {
            return r;
        } else {
            int y = (int) r.info;

            if (dato > y) {
                r = r.getIzq();
            } else {
                r = r.getDer();
            }
        }
    }
    return r;
}

```

```
}
```

```
public String imprimirhojas(Nodito r) {  
    String cad = "";  
    if (r != null) {  
        if (r.izq == null && r.der == null) {  
            cad = r.info + "...";  
        }  
        return cad + imprimirhojas(r.izq) + imprimirhojas(r.der);  
    } else {  
        return "";  
    }  
}
```

```
public String interiores(Nodito n) {  
    String cad = "";  
    if (n != null) {  
        if (n.izq != null || n.der != null) {  
            if(n.info!=raiz.info)  
                cad = n.info + "...";  
        }  
        return cad +interiores(n.izq) + interiores(n.der);  
    } else {  
        return "";  
    }  
}
```

```

public int Altura(Nodito n) {
    return (n != null) ? Math.max(Altura(n.izq), Altura(n.der)) + 1
        : 0;
}

public static void mostrar(Nodito n, int l) {
    if (n == null) return;

    mostrar(n.der, l + 1);

    if (l != 0) {
        for (int i = 0; i < l - 1; i++) {
            System.out.print("\t");
        }
        System.out.println("|-----" + n.info);
    } else {
        System.out.println(n.info);
    }
    mostrar(n.izq, l + 1);
}

public void imprimirArbol(Nodito nd, String p, boolean ultimo) {
    if (nd == null) {
        return;
    }
    imprimirArbol(nd.der, p + (ultimo ? "  " : "|  "), false);
    System.out.print(p);
    System.out.print(ultimo ? "|--" : "|--");
    System.out.println(nd.info);
}

```

```

        imprimirArbol(nd.izq, p + (ultimo ? "| " : " "), true);
    }

}

```

Menú

```

public class TestMenu {
    public static void menu3(String menu)
    {
        ArbolBin <Integer> arbol = new ArbolBin();
        String sel="";

        do {
            sel=boton(menu);
            switch(sel){
                case "Insertar":
                    arbol.InsertarArbol(Tools.leerInt("Dato"));
                    break;
                case "Recorridos":
                    Tools.imprime( "\nPreorden"+arbol.preorden(
arbol.getRaiz())+"\nInorden"+arbol.inorden(arbol.getRaiz())+"\nPosorden"+arbol.po
sorden(arbol.getRaiz())+"\nInordenII"+arbol.inordenII(arbol.getRaiz()));
                    break;
                case "Buscar":
                    if (arbol.arbolVacio()) {
                        Tools.imprimeErrorMsje("Arbol vacio ");
                    } else {
                        int datoBuscar = Tools.leerInt("Ingresa
valor a buscar");

```

```

                                Nodito resultado =
arbol.Buscar(arbol.getRaiz(), datoBuscar);

                                if (resultado != null) {
                                    Tools.imprime("El dato " +
datoBuscar + " se encuentra en el arbol");
                                } else {
                                    Tools.imprime("El dato " +
datoBuscar + " no se encuentra en el rbol");
                                }
                            }
                        }
                    break;

                case "Hojas":
                    if (arbol.arbolVacio()) {
                        Tools.imprimeErrorMsje("Arbol vacio");
                    } else {
                        Tools.imprime("Las hojas del arbol son:
\n" + arbol.imprimirhojas(arbol.getRaiz()));
                        Tools.imprime("Los interiores del arbol
son: \n" + arbol.interiores(arbol.getRaiz()));
                    }

                break;

                case "Altura":
                    if (arbol.arbolVacio()) {
                        Tools.imprimeErrorMsje("Arbol vacio");
                    } else {
                        Tools.imprime("altura : \n" +
arbol.Altura(arbol.getRaiz()));
                    }

```



```

        break;
    case "Ver":
        if (arbol.arbolVacio()) {
            Tools.imprimeErrorMsje("arbol vacio");
        } else {
            System.out.println("El arbol binario es :
\n");
            arbol.mostrar(arbol.getRaiz(), 0);
        }

        break;
    case "Salir":
        Tools.imprime("Fin del programa");
        break;
    }
}while(!sel.equalsIgnoreCase("Salir"));
}

public static void main(String[] args) {
    String menu = "Insertar,Recorridos,Buscar,Hojas,Altura,Ver,Salir";
    menu3(menu);
}

public static String boton(String menu) {
    String valores[]=menu.split(",");
    int n;
    n = JOptionPane.showOptionDialog(null," SELECCIONA DANDO CLICK ", "
M E N U",
        JOptionPane.NO_OPTION,
        JOptionPane.QUESTION_MESSAGE,null,
        valores,valores[0]);
    return ( valores[n]);
}

```

}

}

CONCLUSIÓN

De este trabajo se podría decir que un árbol binario se define como un conjunto finito de elementos llamados nodos. En estos casos se puede usar terminología de relaciones familiares para descubrir las relaciones entre los nodos de un árbol; y que un árbol puede ser implementado fácilmente en una computadora. Es bueno hacer énfasis en esto ya que se puede saber mucho sobre lo que tiene que ver con los árboles; entre las cosas que podemos mencionar se encuentra la raíz, los nodos de un árbol y la diferencia entre nodos sucesores y nodos terminales, como se muestran en el contenido del trabajo.

Por otro lado, conocimos, identificamos y aplicamos las estructuras no lineales en la solución de problemas del mundo real. Al igual conocimos un poco acerca de las teorías de esta unidad así como el aprendizaje utilizando un lenguaje de programación podemos implementar las operaciones básicas (insertar, altura, hojas ver, buscar) en un árbol binario de búsqueda, así como los recorridos en PreOrden, InOrden y PostOrden.

BIBLIOGRAFÍA

- De Árbol, C. (s/f). Unidad IV: Estructuras no lineales 4.1 Árboles. Itpn.mx. Recuperado el 2 de junio de 2023, de <http://itpn.mx/recursosisc/3semestre/estructuradedatos/Unidad%20IV.pdf>
- De forma jerárquica, P. V. Á. C. U. F. de O. I., De vista formal, C. un Ú. P. de E. y. U. S. de C. Q. van A. en C. P. H. S. S. D. un P., Conjunto, C. P. un, Nodos, C. E. se, De orden parcial transitiva, y. U. R., Definida, S. N., la existencia de-Un elemento minimo único, y. C. P., raíz., L., De la raíz, N.-U. P. Ú. P. C. N. p. D., Decir, E., Nodo, U., & q y p y para cualquier nodo q' con las mismas características se cumple q' y q., T. Q. (s/f). 4.1 ARBOLES Y SUS PROPIEDADES. Uniovi.es. Recuperado el 2 de junio de 2023, de https://www6.uniovi.es/usr/cesar/Uned/EDA/Apuntes/TAD_apUM_04.pdf
- Hernandez, E. M., & Perfil, V. T. mi. (s/f). 4.1 ARBOLES - 4.1.1. CONCEPTO DE ARBOL. Blogspot.com. Recuperado el 2 de junio de 2023, de <http://estructuranolineales.blogspot.com/2013/10/41-arboles-411-concepto-de-arbol.html>
- Perfil, V. T. mi. (s/f-a). ESTRUCTURA DE DATOS. Blogspot.com. Recuperado el 2 de junio de 2023, de <http://estructurasnolineales.blogspot.com/2013/11/arboles.html>
- Perfil, V. T. mi. (s/f-b). ESTRUCTURA DE DATOS Unidad 4: ESTRUCTURAS NO LINEALES. Blogspot.com. Recuperado el 2 de junio de 2023, de <http://cuatro-estructuras-no-lineales.blogspot.com/2013/11/>
- Salguero, R. L. (2019). ESTRUCTURAS DE DATOS ÁRBOLES 143 TEMA 4. ÁRBOLES 4.1. CONCEPTOS GENERALES. https://www.academia.edu/40053087/ESTRUCTURAS_DE_DATOS_%C3%81RBOLES_143_TEMA_4_%C3%81RBOLES_4_1_CONCEPTOS_GENERALES