2024-I

UNAM

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

MATERIA

ESTRUCTURA DE DATOS

TEMA

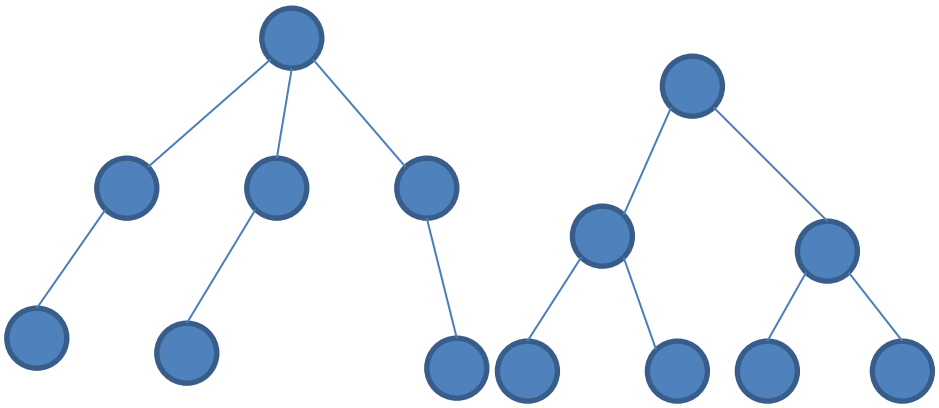
Arboles Binarios

PROFESOR

Mtro. ISC. MIGUEL ANGEL SÁNCHEZ HERNÁNDEZ

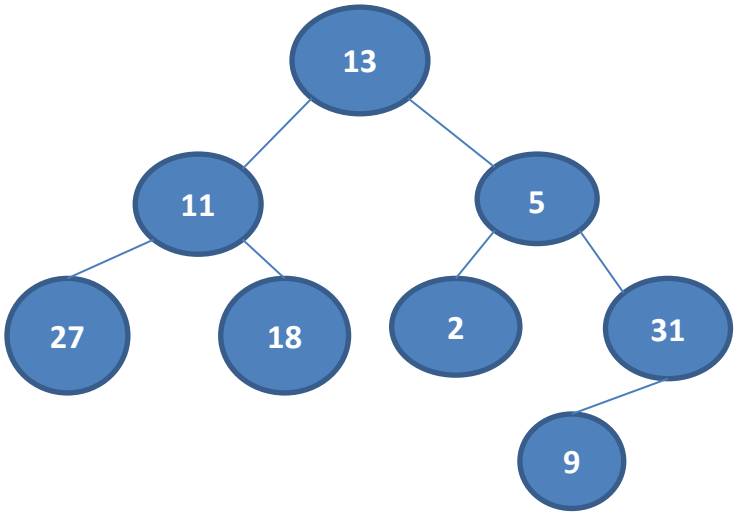
Introducción

Las listas ligadas por lo general proporcionan mayor flexibilidad que los arreglos, pero son estructuras lineales y es difícil utilizarlas para organizar una representación jerárquica de los objetos. Aun cuando las pilas y colas reflejan cierta jerarquía, están limitadas a una sola dimensión. Para superar estas limitaciones, se crean un tipo de datos nuevo llamado árbol que se componen de nodos y aristas.



Un árbol binario es aquel cuyos nodos tienen dos hijos (posiblemente vacios), y cada hijo se designa ya sea como un hijo izquierdo o como un hijo derecho, por ejemplo el árbol de la parte derecha de la figura de arriba, representa un árbol binario.

Si no nosotros queremos implementar una estructura de datos que sea un árbol binario, podemos ocupar otra estructura de datos que sería la lista ligada.



Ejercicio: Implementar el árbol binario que se muestra en la figura, ocupando la estructura de datos de una lista ligada.

Otra manera de implementar un árbol es mediante un arreglo, esto se logra declarando un arreglo que contiene un objeto que representa un nodo, el cual tiene un campo para la información y dos campos más, que nos indican la referencia de los nodos hijo, que no sería otra cosa que el índice en donde se encuentran dichos nodos en el arreglo, si tomamos en cuenta el árbol binario de arriba el arreglo quedaría de la siguiente manera.

Índice	Información	izquierdo	derecho
0	13	1	2
1	11	3	4
2	5	5	6
3	27	null	null
4	18	null	null
5	2	null	null
6	31	7	null
7	9	null	null

Ejercicio: De acuerdo con lo explicado de cómo implementar un árbol binario con un arreglo y tomando en cuenta tabla de arriba construye el árbol binario.

Vamos a implementar los arboles binarios creando una clase que represente los nodos, los campos a tomar son:

Dato: almacena la información del nodo.
Izquierdo: almacena la referencia del nodo izquierdo.
Derecho: almacena la referencia del nodo derecho.

```
public class Nodo<E> {
	protected E dato;
	protected Nodo<E> izquierdo,derecho;
	protected String etiqueta;
	public Nodo() {
		izquierdo=derecho=null;
	}
	public Nodo(E dato,String etiqueta) {
		this(dato,null,null,etiqueta);
	}
	public Nodo(E dato,Nodo<E> izquierdo,Nodo<E> derecho,String etiqueta) {
		this.dato=dato;
		this.izquierdo=izquierdo;
		this.derecho=derecho;
		this.etiqueta=etiqueta;
	}
	@Override
	public int hashCode() {
		return Objects.hash(dato);
	}
	public boolean mayor(Object obj) {
		boolean resultado=false;
		if(obj instanceof Integer && this.dato instanceof Integer) {
			Integer dato1=(Integer)this.dato;
			Integer dato2=(Integer)obj;
			if(dato1<dato2) {
				return true;
			}
		}
		return resultado;
	}
```

```
	@Override
	public boolean equals(Object obj) {
		if(obj instanceof Integer && this.dato instanceof Integer) {
			Integer dato1=(Integer)this.dato;
			Integer dato2=(Integer)obj;
			if(dato1.equals(dato2)) {
				return true;
			}else {
				return false;
			}
		}
		Nodo other = (Nodo) obj;
		return Objects.equals(dato, other.dato);
	}
}
```

Inserción

Para insertar un nodo nuevo, primero se debe buscar un nodo de árbol con un punto muerto y el nodo nuevo tiene que adjuntarse al mismo.

```
public class ArbolBinarioOrden<E> {
	protected Nodo<E> raiz;

	public ArbolBinarioOrden() {
		raiz = null;
	}

	public Nodo<E> getRaiz() {
		return raiz;
	}

	public void insertar(E dato) {
		Nodo<E> n = raiz;
		Nodo<E> previo = null;
		while (n != null) {
			previo = n;
			if (n.mayor(dato)) {
				n = n.derecho;
			} else {
				n = n.izquierdo;
			}
		}
		if (raiz == null) {
			raiz = new Nodo<E>(dato, "raiz");
		} else if (previo.mayor(dato)) {
			previo.derecho = new Nodo<E>(dato, "derecho,padre=" + previo.dato);
		} else {
			previo.izquierdo = new Nodo<E>(dato, "izquierdo,padre=" + previo.dato);
		}
	}

	public void imprimir(Nodo<E> n) {
		System.out.println(n.dato + " " + n.etiqueta);
	}
}
```

Recorrido de un árbol.

Existen dos métodos importantes de recorrido, los cuales son el recorrido de amplitud y el de profundidad, el primero que tocaremos es el recorrido de amplitud.

Recorrido de amplitud.

Este consiste en visitar cada nodo empezando por el nivel inferior (o superior) y moverse hacia abajo (o hacia arriba) nivel por nivel, visitando cada nodo de cada nivel de izquierda a derecha (o de derecha a izquierda).

```
public void recorridoAmplitud() throws Exception {
        Nodo<E> n = raiz;
        Cola<Nodo<E>> cola = new Cola<>();
        if (n != null) {
                cola.insertar(n);
                while (!cola.estaVacia()) {
                        n = cola.extraer();
                        imprimir(n);
                        if (n.izquierdo != null) {
                                cola.insertar(n.izquierdo);
                        }
                        if (n.derecho != null) {
                                cola.insertar(n.derecho);
                        }
                }
        }
}
```

Recorrido de Profundidad

Este nos indica que primero tiene que ir lo más lejos posible a la izquierda (o a la derecha), luego regresa hacia arriba hasta el primer cruce, avanza una paso a la derecha (o a la izquierda), y de nuevo lo más lejos posible a la izquierda (o la derecha). Se repite este proceso hasta que todos lo nodos se visiten.

Tomando esto en cuenta tenemos lo siguiente:

V: visitar un nodo.
L: recorrer el subárbol izquierdo.
R: recorrer el subárbol derecho.

Si nosotros ordenamos estas posibilidades tenemos 3!=6, lo cual queda como sigue:
VLR VRL
LVR LRV
RVL RLV

Nosotros podemos tomar lo siguiente:

VLR: recorrido del árbol en preorden.
LVR: recorrido del árbol en orden.
LRV: recorrido del árbol en postorden.

La gran ventaja de ocupar estos métodos es que son muy sencillos, y se emplean en forma recursiva, este trabajo lo realiza el sistema de pila en tiempo de ejecución.

```
public void preorden(Nodo<E> n) {
        if (n != null) {
                imprimir(n);
                preorden(n.izquierdo);
                preorden(n.derecho);
        }
}

public void orden(Nodo<E> n) {
        if (n != null) {
                orden(n.izquierdo);
                imprimir(n);
                orden(n.derecho);
        }
}

public void postorden(Nodo<E> n) {
        if (n != null) {
                postorden(n.izquierdo);
                postorden(n.derecho);
                imprimir(n);
        }
}
```

Implementación No Recursiva

La implementación no recursiva del recorrido de un árbol es otra opción que podemos implementar para esto ocuparemos una pila, la cual almacenara los nodos izquierdos y derechos de cada nodo padre.

```
public void noRecursivoOrden() throws Exception {
		Nodo<E> n = raiz;
		Pila<Nodo<E>> pila = new Pila();
		while (n != null) {
			while (n != null) {
				if (n.derecho != null) {
					pila.insertar(n.derecho);
				}
				pila.insertar(n);
				n = n.izquierdo;
			}
			n = pila.extraer();
			while (!pila.estaVacia() && n.derecho == null) {
				imprimir(n);
				n = pila.extraer();
			}
			imprimir(n);
			if (!pila.estaVacia()) {
				n = pila.extraer();
			} else {
				n = null;
			}
		}
	}
```

Tomando el ejemplo de arriba cambia el siguiente código:

```
public void noRecursivoPreorden(){
    Nodo n=raiz;
    Pila pila=new Pila();
    if(n!=null){
      pila.insertar(n);
      while(!pila.estaVacia()){
        n=(Nodo)pila.extraer();
        imprimir(n);
        if(n.derecho!=null){
          pila.insertar(n.derecho);
        }
        if(n.izquierdo!=null){
          pila.insertar(n.izquierdo);
        }
      }
    }
  }
```

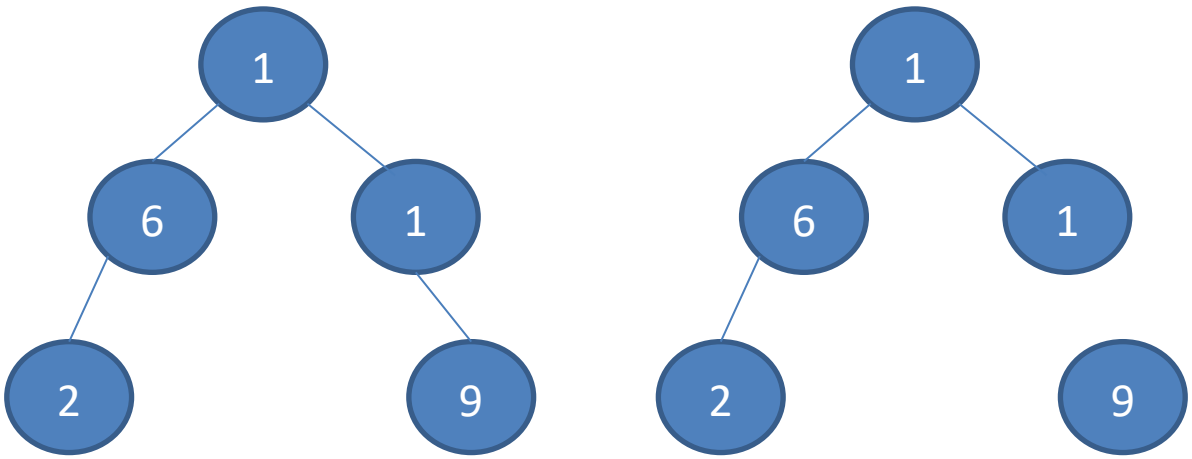
```
public void noRecursivoPostOrden(){
      Nodo n=raiz,p=raiz;
      Pila pila=new Pila();
      while(n!=null){
         for(;n.izquierdo!=null;n=n.izquierdo){
            pila.insertar(n);
         }
         while(n!=null && (n.derecho==null || n.derecho==p)){
            imprimir(n);
            p=n;
            if(pila.estaVacia()){
               return;
            }
            n=(Nodo)pila.extraer();
         }
         pila.insertar(n);
         n=n.derecho;
      }
   }
```

Eliminación de un Nodo del Árbol Binario

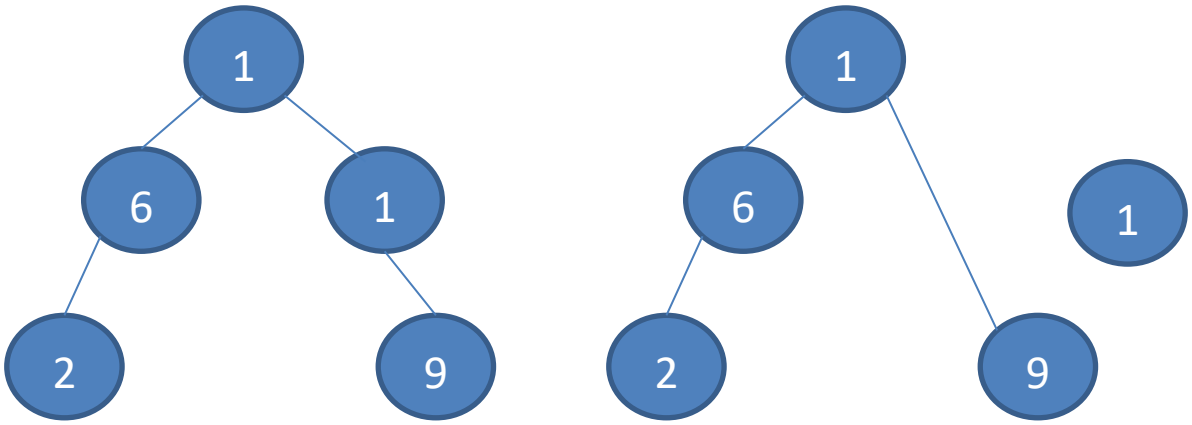
Existen tres casos que se presentan cuando se quiere eliminar un nodo los cuales son:

1.- El nodo es una hoja.
2.- El nodo tiene un hijo.
3.- El nodo tiene dos hijos.

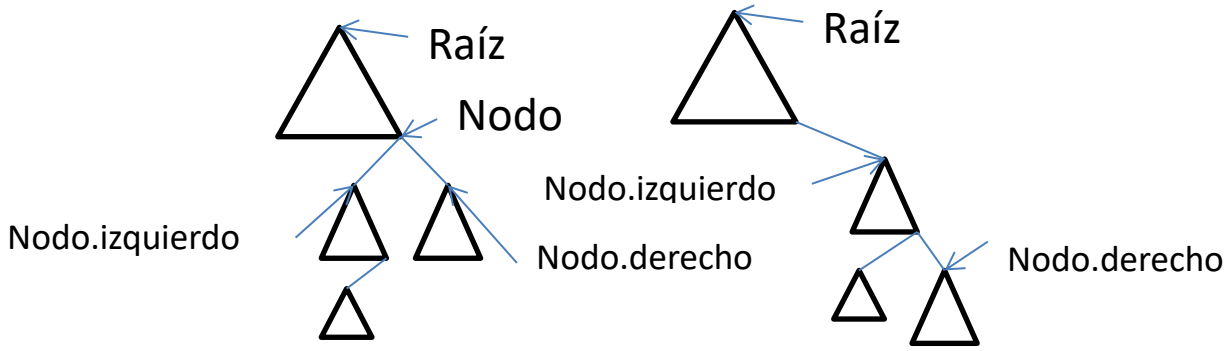
Caso 1



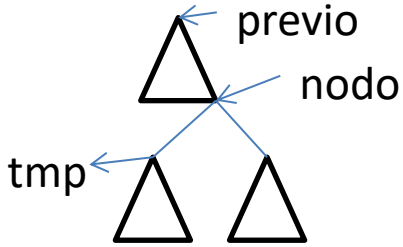
Caso 2

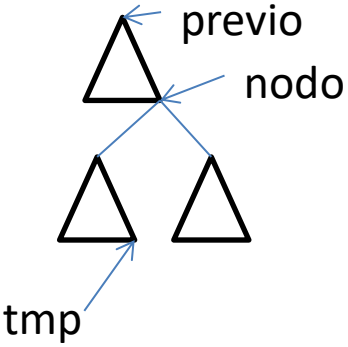

1
6
1
2
9
1
6
1
2
9

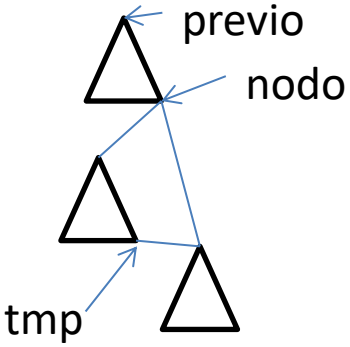
Caso 3

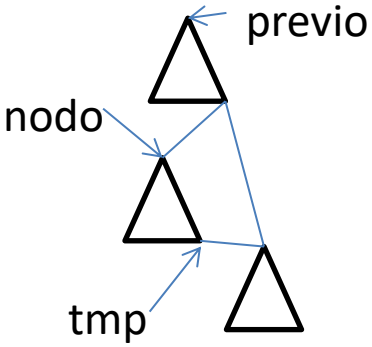

Raíz
Nodo
Nodo.izquierdo
Nodo.derecho
Raíz
Nodo.izquierdo
Nodo.derecho

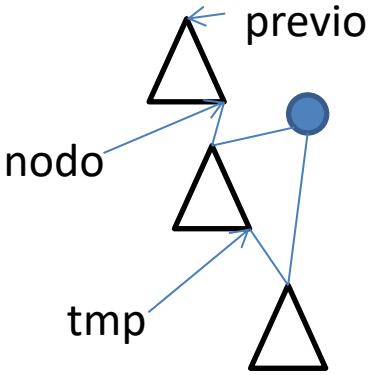
Más casos Especiales

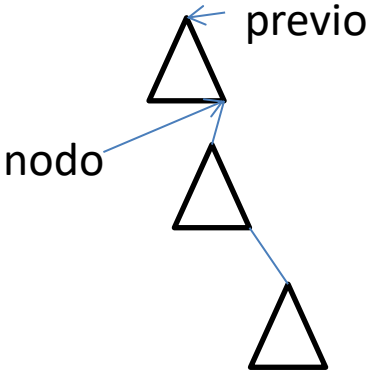












```
@SuppressWarnings("unused")
	public void eliminar(E dato) {
		Nodo<E> tmp;
		Nodo<E> nodo;
		Nodo<E> n = raiz;
		Nodo<E> previo = null;
		while (n != null && !n.equals(dato)) {
			previo = n;
			if (n.mayor(dato)) {
				n = n.derecho;
			} else {
				n = n.izquierdo;
			}
		}
		nodo = n;
		if (n != null && n.equals(dato)) {
			if (nodo.derecho == null) {
				nodo = nodo.izquierdo;
			} else if (nodo.izquierdo == null) {
				nodo = nodo.derecho;
			} else {
				tmp = nodo.izquierdo; // 1
				while (tmp.derecho != null) { // 2
					tmp = tmp.derecho;
				}
				tmp.derecho = nodo.derecho; // 3
				nodo = nodo.izquierdo; // 4
			}
			if (n == raiz) {
				raiz = nodo;
			} else if (previo.izquierdo == n) {
				previo.izquierdo = nodo;
			} else {
				previo.derecho = nodo; // 5
			}
		} else if (raiz != null) {
			System.out.println("No se encuentra el dato " + dato);
		} else {
			System.out.println("Arbol vacio");
		}
	}
```

Programa de prueba

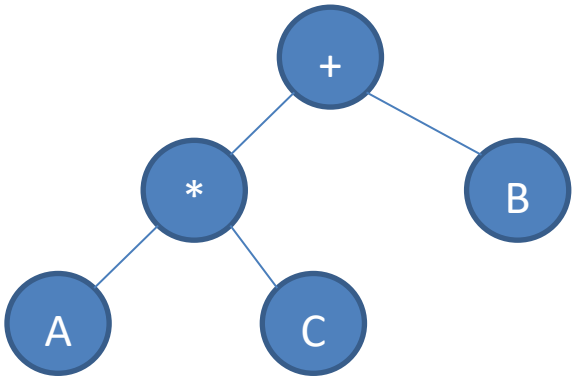
```
public class PruebaArbolBinario {
	public static void main(String[] args) {
		ArbolBinarioOrden<Integer> arbol=new ArbolBinarioOrden<>();
		arbol.insertar(45);
		arbol.insertar(3);
		arbol.insertar(200);
		arbol.insertar(20);
		arbol.insertar(100);
		arbol.insertar(50);
		arbol.insertar(15);
		arbol.insertar(30);
		try {

			//arbol.orden(arbol.getRaiz());
			arbol.noRecursivoOrden();
			arbol.eliminar(45);
			System.out.println("----");
			arbol.preorden(arbol.getRaiz());
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

}
```

Construcción de un árbol binario de una expresión

Sabemos que una expresión prefija sencilla como +AB, el operador, +, ira en el nodo raíz, y los operandos A y B quedan en el nodo izquierdo y el nodo derecho. Pero si tenemos la siguiente expresión, +*A C B, el árbol queda de la siguiente manera.



El siguiente algoritmo nos proporciona una manera como construir el árbol binario de una expresión.

```
Obtener símbolo
Crear Nodo
Poner símbolo en nodo nuevo
sigMovimiento=izquierdo
Limpiar pila
Obtener símbolo
While no sea fin de la cadena do
        ultimoNodo←nuevoNodo
        Crear Nodo
        Poner símbolo en nodo nuevo
        If(sigMovimiento es “izquierdo”)
                Añadir nuevoNodo a la izquierda del ultimoNodo
                Meter ultimoNodo en la pila
        Else
                Sacar de la pila nodo para ultimoNodo
                Añadir el nodo de la pila a la derecha ultimoNodo
        If(símbolo es un operador)
                sigMovimiento=izquierda
        else
                sigMovimiento=derecha
        Obtener símbolo
End while
```

Ejercicio 1: Implementar con el algoritmo anterior y construye otro algoritmo para evaluar la expresión.