Árbol binario y árbol binario de búsqueda

José Luis Quiroz Fabián

1 Árbol binario

Un árbol es una estructura dinámica que consta de un conjunto de finito de elementos, denominados **nodos**, y un conjunto de líneas dirigidas llamadas **ramas** (ver Figura 1). Al nodo superior del árbol se le conoce como la **raíz** y a los nodos más inferiores se le conocen como las **hojas** . También a un nodo que tiene ramas se le conoce como **nodo padre** y a los nodos conectados a las ramas como **nodos hijos**.

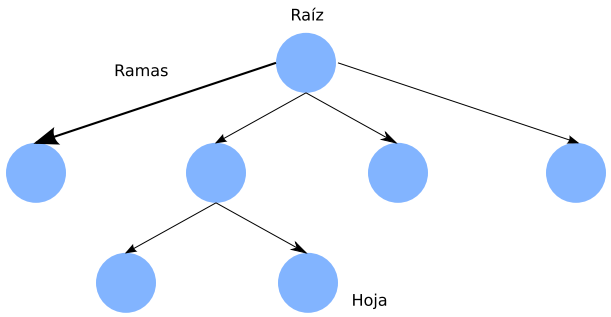

Figure 1: Ejemplo de un árbol

Un árbol binario es aquel árbol que únicamente puede tener como máximo dos subárboles (sub-árbol izquierdo y sub-árbol derecho). Una propuesta de implementación se muestra en el Código 1.

```
1  public class ArbolBinario<E> {
2
3    private  ArbolBinario<E> izq=null;
4
5    private ArbolBinario<E> der=null;
6
7    private E info=null;
8
9
10   public ArbolBinario(E info) {
11     super();
12     this.info = info;
13   }
14
```

```
15
16  public ArbolBinario(ArbolBinario<E> izq, ArbolBinario<E> der, E
      info) {
17    super();
18    this.izq = izq;
19    this.der = der;
20    this.info = info;
21  }
22
23
24  public ArbolBinario<E> getIzq() {
25    return izq;
26  }
27
28
29  public void setIzq(ArbolBinario<E> izq) {
30    this.izq = izq;
31  }
32
33
34  public ArbolBinario<E> getDer() {
35    return der;
36  }
37
38
39  public void setDer(ArbolBinario<E> der) {
40    this.der = der;
41  }
42
43
44  public E getInfo() {
45    return info;
46  }
47
48
49  public void setInfo(E info) {
50    this.info = info;
51  }
52
53  public void mostrarArbol(ArbolBinario<E> nodo) {
54    mostrarArbol(nodo, 0);
55  }
56
57  private void mostrarArbol(ArbolBinario<E> nodo, int nivel) {
58    if (nodo == null) {
59      return;
60    }
61
62    mostrarArbol(nodo.getDer(), nivel + 1);
63    System.out.println("  ".repeat(nivel * 2) + nodo.getInfo());
64    mostrarArbol(nodo.getIzq(), nivel + 1);
65  }
66
67  @Override
68  public String toString() {
69    return "" + info;
70  }
```

```
71 }
```

Código 1: Propuesta de implementación de un árbol binario

2 Recorridos

En general, hay cuatro recorridos de uso común para visitar todos los nodos de un árbol. La diferencia entre estos recorridos es el orden en que es visitado cada nodo. Los cuatro recorridos que vamos a ver se llaman:

- Recorrido en preorden
- Recorrido en inorden
- Recorrido en postorden
- Recorrido en amplitud

2.1 Recorrido en preorden

Para este recorrido:

1. Se examina/visita la raíz
2. Se recorre el subárbol izquierdo en preorden
3. Se recorre el subárbol derecho en preorden

2.2 Recorrido en inorden

Para este recorrido:

1. Se recorre el subárbol izquierdo en inorden
2. Se examina la raíz
3. Se recorre el subárbol derecho en inorden

2.3 Recorrido en postnorden

Para este recorrido:

1. Se recorre el subárbol izquierdo en postorden
2. Se recorre el subárbol derecho en postorden
3. Se examina la raíz

2.4 Recorrido en amplitud

Para este recorrido se considera una cola que previamente tiene almacena la raíz del árbol

1. Se extrae y examina de la cola el nodo
2. Se guarda en la cola el el subárbol izquierdo del nodo
3. Se guarda en la cola el el subárbol derecho del nodo
4. Se repite el proceso

2.5 Ejemplo de recorridos

Por ejemplo, considere el árbol de la Figura 2.

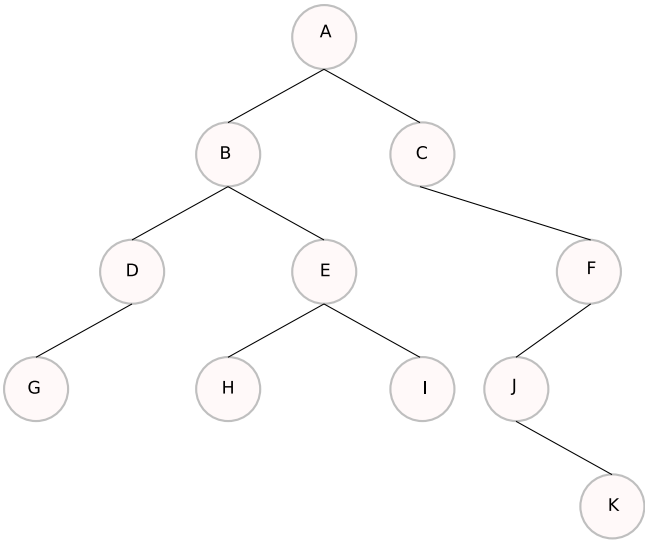

Figure 2: Recorrido en preorden

- El recorrido en preorden permite visitar a los nodos de la siguiente forma: ABDGEHICFJK
- El recorrido en inorden permite visitar a los nodos de la siguiente forma: GDBHEIACJKF
- El recorrido en postorden permite visitar a los nodos de la siguiente forma: GDHIEBKJFCA
- El recorrido en amplitud permite visitar a los nodos de la siguiente forma: ABCDEFGHIJK

El Código 2 muestra una propuesta de implementación de los recorridos en un árbol binario.

```java
import java.util.LinkedList;
import java.util.Queue;

public class Recorrido{


  public static<E> void preorden(ArbolBinario<E> arbol){

    if(arbol!=null){
      System.out.print(" "+arbol);
      preorden(arbol.getIzq());
      preorden(arbol.getDer());
    }
  }
  public static<E> void inorden(ArbolBinario<E> arbol){

    if(arbol!=null){
      inorden(arbol.getIzq());
      System.out.print(" "+arbol);
      inorden(arbol.getDer());

    }
  }

  public static<E> void postorden(ArbolBinario<E> arbol){

    if(arbol!=null){
      postorden(arbol.getIzq());
      postorden(arbol.getDer());
      System.out.print(" "+arbol);

    }
  }

  public static<E> void amplitud(ArbolBinario<E> arbol) {
    ArbolBinario<E> izq,der;
    if (arbol == null) {
      return;
    }

    Queue<ArbolBinario<E>> cola = new LinkedList<ArbolBinario<E>>()
    ;
    cola.add(arbol);

    while (!cola.isEmpty()) {
      ArbolBinario<E> nodoActual = cola.poll();
      System.out.print (" "+nodoActual);
      izq=nodoActual.getIzq();
      der=nodoActual.getDer();
      if ( izq!= null) {
        cola.add(izq);
      }
      if (der != null) {
        cola.add(der);
      }
```

```
55      }
56    }
57
58
59  }
```

Código 2: Recorridos en un árbol binario.

3 Prueba del Árbol Binario

El Código 3 muestra una propuesta de clase **App** para verificar la implementación del árbol binario y sus recorridos.

```
1
2  import java.util.Stack;
3
4  public class App {
5
6
7    public static void main(String[] args){
8
9      Stack<ArbolBinario<Integer>> pila = new Stack<ArbolBinario<
       Integer>>();
10
11
12     ArbolBinario<Integer> izq= new ArbolBinario<Integer>(2);
13     ArbolBinario<Integer> der= new ArbolBinario<Integer>(4);
14     ArbolBinario<Integer> raiz= new ArbolBinario<Integer>(izq,der
       ,3);
15
16     pila.push(raiz);
17     izq= new ArbolBinario<Integer>(6);
18     der= new ArbolBinario<Integer>(9);
19     raiz= new ArbolBinario<Integer>(izq,der,9);
20     pila.push(raiz);
21     izq= new ArbolBinario<Integer>(3);
22     der= new ArbolBinario<Integer>(2);
23     raiz= new ArbolBinario<Integer>(izq,der,8);
24     pila.push(raiz);
25     raiz= new ArbolBinario<Integer>(5);
26     raiz.setIzq(pila.pop());
27     raiz.setDer(pila.pop());
28
29     raiz.mostrarArbol(raiz);
30     System.out.println("Preorden:");
31     ecorrido.preorden(raiz);
32     System.out.println("\nInorden:");
33     Recorrido.inorden(raiz);
34     System.out.println("\nPostorden:");
35     Recorrido.postorden(raiz);
36     System.out.println("\nAmplitud:");
37     Recorrido.amplitud(raiz);
38     System.out.println();
39
40   }
```

```
41 }
```

Código 3: Recorridos en un árbol binario.

4 Árbol binario de búsqueda

Un Árbol Binario de Búsqueda (ABB) es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x.

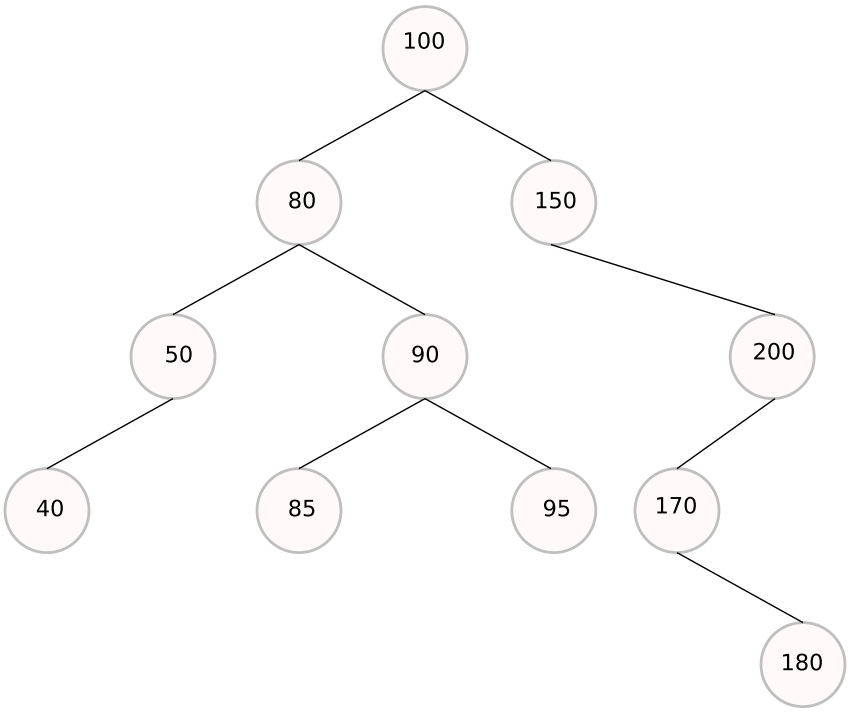

Figure 3: Ejemplo de árbol binario de búsqueda

Una forma simple de implementar un árbol binario de búsqueda se muestra en la Figura 4.

```
1
2 public class ArbolBinarioDeBusqueda<E extends Comparable<E>>
    extends ArbolBinario<E>{
3
4
5   public ArbolBinarioDeBusqueda(E info) {
6     super(info);
```

```
7   }
8
9
10  public ArbolBinarioDeBusqueda(ArbolBinarioDeBusqueda<E> izq,
      ArbolBinarioDeBusqueda<E> der, E info) {
11    super(izq,der,info);
12
13  }
14
15  public void insertar(E item){
16
17    if (item.compareTo(getInfo()) < 0) {
18      if (getIzq() == null) {
19        setIzq( new ArbolBinarioDeBusqueda<>(item));
20      } else {
21        ((ArbolBinarioDeBusqueda<E>) getIzq()).insertar(item);
22      }
23    } else if (item.compareTo(getInfo()) > 0) {
24      if (getDer()  == null) {
25        setDer(new ArbolBinarioDeBusqueda<>(item));
26      } else {
27        ((ArbolBinarioDeBusqueda<E>) getDer()).insertar(item);
28      }
29    }
30
31
32}
33  private E buscar(ArbolBinarioDeBusqueda<E> arbol, E buscado){
34
35
36    if(arbol!=null) {
37      if(buscado.compareTo(arbol.getInfo())==0)
38        return arbol.getInfo();
39
40      else{
41        if(buscado.compareTo(arbol.getInfo())<0){
42          return buscar((ArbolBinarioDeBusqueda<E>) arbol.getIzq(),
    buscado);
43
44        }else{
45
46          return buscar((ArbolBinarioDeBusqueda<E>) arbol.getDer(),
    buscado);
47        }
48
49      }
50    }
51    return null;
52  }
53
54  public E buscar(E buscado){
55
56    return buscar(this,buscado);
57
58  }
59
```

```
}
```

Código 4: Árbol binario de búsqueda.

El Código 5 muestra una propuesta de clase **App** para verificar la implementación del árbol binario de búsqueda.

```
import java.util.Stack;

public class App {


  public static void main(String[] args){

    Stack<ArbolBinario<Integer>> pila = new Stack<ArbolBinario<
    Integer>>();

    ArbolBinario<Integer> izq= new ArbolBinario<Integer>(2);
    ArbolBinario<Integer> der= new ArbolBinario<Integer>(4);
    ArbolBinario<Integer> raiz= new ArbolBinario<Integer>(izq,der
    ,3);

    pila.push(raiz);


    izq= new ArbolBinario<Integer>(6);
    der= new ArbolBinario<Integer>(9);
    raiz= new ArbolBinario<Integer>(izq,der,9);
    pila.push(raiz);
    izq= new ArbolBinario<Integer>(3);
    der= new ArbolBinario<Integer>(2);
    raiz= new ArbolBinario<Integer>(izq,der,8);
    pila.push(raiz);
    raiz= new ArbolBinario<Integer>(5);
    raiz.setIzq(pila.pop());
    raiz.setDer(pila.pop());

    raiz.mostrarArbol(raiz);
    System.out.println();

    ArbolBinarioDeBusqueda<Integer> abb= new ArbolBinarioDeBusqueda
    <Integer>(6);


    abb.insertar(9);
    abb.insertar(6);
    abb.insertar(3);
    abb.insertar(2);
    abb.insertar(8);
    abb.insertar(5);

    abb.mostrarArbol(abb);

    System.out.println(abb.buscar(5));
    System.out.println(abb.buscar(20));
  }

```

```
48 }
```

Código 5: Prueba del árbol binario de búsqueda.

5 Ejercicios

- Crear un proyecto en **Visual Studio Code** y llamarlo **Arboles**. Crear los paquetes **edatos.nolineales.arboles.binario** y **edatos.nolineales.arboles.binario.abb**. En el primer paquete almacenar las clases **ArbolBinario**, **Recorrido** y la primer **App**. En el segundo paquete **ArbolBinarioDeBusqueda** y la segunda clase **App**. Prueba su implementación.

- Eliminar un nodo en un árbol binario de búsqueda es un poco más complejo que insertar o buscar, porque se debe mantener la propiedad de orden del árbol después de la eliminación. Investigue e implemente la eliminación (**public void eliminar(E e)**) en nuestra propuesta de árbol binario de búsqueda.

- Desarrollar una aplicación móvil donde los usuarios puedan interactuar con el árbol para agregar nuevos nodos, eliminar nodos, buscar nodos y mostrar el árbol.