



Árboles

- Presentar la estructura no lineal más importante en computación
- Mostrar la especificación e implementación de varios tipos de árboles
- Algoritmos de manipulación de árboles

Contenidos

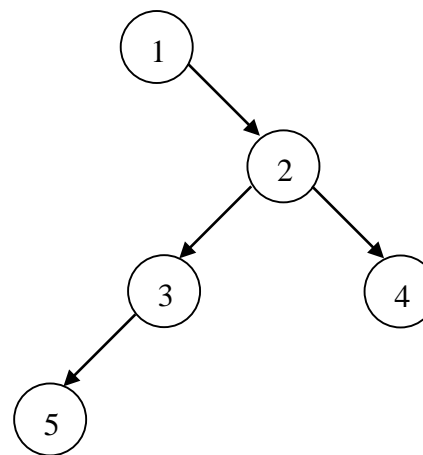
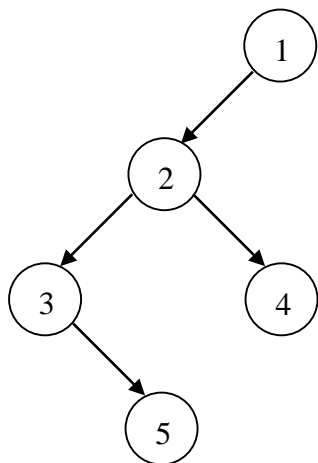
- 1. Terminología fundamental**
 - 1.1. Recorridos de un árbol
- 2. Árboles binarios**
 - 2.1. Definición
 - 2.2. Especificación
 - 2.3. Implementación
- 3. Árboles binarios de búsqueda**
 - 3.1. Definición
 - 3.2. Especificación
- 4. Árboles binarios equilibrados**
 - 4.1. Árboles AVL
- 5. Árboles generales**
 - 5.1. Especificación
 - 5.2. Implementación

Árbol Binario

- **Árbol binario:** un árbol donde cada nodo tiene como máximo **grado 2**, es decir, un árbol binario es:
 - Un árbol vacío, o
 - Un árbol en que sus nodos tienen un hijo izquierdo y un hijo derecho. Cada uno de estos hijos es a su vez un árbol binario
- Se distinguen los hijos de un nodo como **izquierdo y derecho**

Árbol Binario

■ Ej:



- **Preorden:** raíz, preorden(A_{izq}), preorden(A_{der})
 - Ej: 1, 2, 3, 5, 4 (coincide en los dos árboles)
- **Postorden:** postorden(A_{izq}), postorden(A_{der}), raíz
 - Ej: 5, 3, 4, 2, 1 (coincide en los dos árboles)
- **Inorden:** inorden(A_{izq}), raíz, inorden(A_{der})
 - Ejs: 3, 5, 2, 4, 1
1, 5, 3, 2, 4
- **Anchura:** recorrido por niveles
 - Ejs: 1, 2, 3, 4, 5 (coincide en los dos árboles)

TAD Árbol Binario

Especificación

```
public class ArbolBinario<E> {  
    // Declaración de tipos: ArbolBinario  
    // Características: Un árbol binario es un árbol vacío o un nodo con dos hijos (izquierdo y derecho)  
    // que a su vez son árboles binarios. Los objetos son modificables  
    public ArbolBinario();  
        // Produce: Un árbol vacío  
    public ArbolBinario(E elemRaiz, ArbolBinario<E> hi, ArbolBinario<E> hd) throws NullPointerException;  
        // Produce: Si hi o hd son null, lanza la excepción NullPointerException. En caso contrario,  
        // construye un árbol de raíz elemRaiz, hijo izquierdo hi e hijo derecho hd  
  
    public boolean esVacio();  
        // Produce: Cierto si this está vacío. Falso en caso contrario.  
    public E raiz() throws ArbolVacioExcepcion;  
        // Produce: Si this está vacío lanza la excepción ArbolVacioExcepcion,  
        // sino devuelve el objeto almacenado en la raíz  
    public ArbolBinario<E> hijoIzq() throws ArbolVacioExcepcion;  
        // Produce: Si this está vacío lanza la excepción ArbolVacioExcepcion,  
        // sino devuelve el subárbol izquierdo  
    public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion;  
        // Produce: Si this está vacío lanza la excepción ArbolVacioExcepcion,  
        // sino devuelve el subárbol derecho
```

TAD Árbol Binario

```
public boolean esta (E elemento);
    // Produce:  Cierto si elemento está en this, falso, en caso contrario
public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;
    // Modifica:  this
    // Produce:   Si this está vacío lanza la excepción ArbolVacioExcepcion,
    //           sino asigna el objeto elemRaiz a la raíz del árbol this
public void setHijoIzq(ArbolBinario<E> hi) throws ArbolVacioExcepcion, NullPointerException;
    // Modifica:  this
    // Produce:   Si hi es null, lanza la excepción NullPointerException.
    //           En caso contrario, si this está vacío lanza la excepción ArbolVacioExcepcion,
    //           sino asigna el árbol hi como subárbol izquierdo de this
public void setHijoDer(ArbolBinario<E> hd) throws ArbolVacioExcepcion, NullPointerException;
    // Modifica:  this
    // Produce:   Si hd es null, lanza la excepción NullPointerException.
    //           En caso contrario, si this está vacío lanza la excepción ArbolVacioExcepcion,
    //           sino asigna el árbol hd como subárbol derecho de this
public void suprimir ();
    // Modifica:  this
    // Produce:   El árbol binario vacío
```

TAD Árbol Binario

Implementación

■ Paso 1: Definición interfaz

```
public interface ArbolBinario<E>      {  
    public boolean esVacio();  
    public E raiz() throws ArbolVacioExcepcion;  
    public ArbolBinario<E> hijolzq() throws ArbolVacioExcepcion ;  
    public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion;  
    public boolean esta (E elemento);  
    public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;  
    public void setHijolzq(ArbolBinario<E> hi)  
        throws ArbolVacioExcepcion, NullPointerException;  
    public void setHijoDer(ArbolBinario<E> hd)  
        throws ArbolVacioExcepcion, NullPointerException;  
    public void suprimir();  
}
```

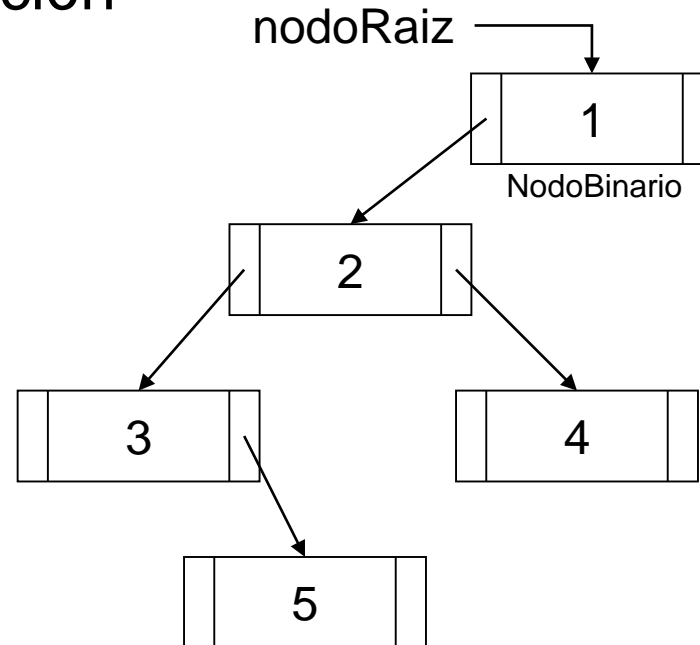
■ Paso 2: Clase implemente la interfaz

□ Mediante estructuras enlazadas genéricas

```
■ public class EnlazadoArbolBinario<E> implements ArbolBinario<E>
```

TAD Árbol Binario

■ Representación



```
public class EnlazadoArbolBinario<E> implements ArbolBinario<E>{  
    private NodoBinario<E> nodoRaiz;
```


TAD Árbol Binario

```
public class NodoBinario<E>{
    private E elemento;           // referencia al elemento del nodo
    private NodoBinario<E> izq;   // referencia al nodo izquierdo
    private NodoBinario<E> der;   // referencia al nodo derecho

    public NodoBinario(E e, NodoBinario<E> hi, NodoBinario<E> hd){
        elemento = e;
        izq = hi;
        der = hd;
    }
    public E getElemento() {
        return elemento;
    }
    public NodoBinario<E> getIzq() {
        return izq;
    }
    public NodoBinario<E> getDer() {
        return der;
    }
    public void setElemento(E e) {
        elemento = e;
    }
    public void setIzq(NodoBinario<E> hi) {
        izq = hi;
    }
    public void setDer(NodoBinario<E> hd) {
        der = hd;
    }
}
```

Uso TAD Árbol Binario

Recorridos en profundidad:

```
public static <E> void preorden(ArbolBinario<E> a){
    if (!a.esVacio()) {
        System.out.print(a.raiz() + " ");
        preorden(a.hijolzq());
        preorden(a.hijoDer());
    }
}

public static <E> void inorden(ArbolBinario<E> a){
    if (!a.esVacio()) {
        inorden(a.hijolzq());
        System.out.print(a.raiz() + " ");
        inorden(a.hijoDer());
    }
}

public static <E> void postorden(ArbolBinario<E> a){
    if (!a.esVacio()) {
        postorden(a.hijolzq());
        postorden(a.hijoDer());
        System.out.print(a.raiz() + " ");
    }
}
```

Uso TAD Árbol Binario

Recorrido en anchura:

```
public static <E> void anchura(ArbolBinario<E> a){  
    Cola<ArbolBinario<E>> c = new EnlazadaCola<ArbolBinario<E>>();  
    c.insertar(a);  
    do {  
        a = c.suprimir();  
        if (!a.esVacio()){  
            System.out.print(a.raiz() + " ");  
            c.insertar(a.hijoIzq());  
            c.insertar(a.hijoDer());  
        }  
    } while (!c.esVacio());  
}
```

Uso TAD Árbol Binario

- Escribe un método que cuente el **número de nodos** de un árbol binario.

```
public static <E> int numNodos(ArbolBinario<E> a){  
    if (a.esVacio())  
        return 0;  
    return 1 + numNodos (a.hijolzq()) + numNodos(a.hijoDer());  
}
```

- Escribe un método que devuelva el **número de hojas** de un árbol binario.

```
public static <E> int numHojas (ArbolBinario<E> a){  
    if (a.esVacio())  
        return 0;  
    if (a.hijolzq().esVacio() && a.hijoDer().esVacio())  
        return 1;  
    return numHojas(a.hijolzq()) + numHojas(a.hijoDer());  
}
```

Uso TAD Árbol Binario

- Un árbol **degenerado** es un árbol en el que cada nodo tiene solamente un subárbol. Escribe un método que dado un árbol binario indique si es degenerado o no.

```
public static <E> boolean degenerado (ArbolBinario<E> a){  
    if (a.esVacio())  
        return true;  
    else if (a.hijolzq().esVacio() && a.hijoDer().esVacio())  
        return true;  
    else if (!a.hijolzq().esVacio() && !a.hijoDer().esVacio())  
        return false;  
    else return degenerado(a.hijolzq()) && degenerado(a.hijoDer());  
}
```

Uso TAD Árbol Binario

- Escribe un método que dado un árbol binario, realice una **copia** del mismo.

```
public static <E> ArbolBinario<E> copia(ArbolBinario<E> a){  
    if (a.esVacio())  
        return new EnlazadoArbolBinario<>();  
    else  
        return new EnlazadoArbolBinario<>(a.raiz(),  
                                             copia(a.hijoIzq()), copia(a.hijoDer()));  
}
```

Uso TAD Árbol Binario

- Más ejercicios en la actividad 2 (trabajo colaborativo)