

# Examen E.D 2023 Árbol

ArbolBinario {

- NodoArbol raiz
- ArbolBinario
- ArbolBinario (int dato)

NodoArbol {

- NodoArbol izq, der
- NodoArbol (int dato, NodoArbol izq, NodoArbol der)
- NodoArbol getIzq()
- NodoArbol getDer()
- int getPato()

```
public boolean sonHijosMenores() {
    boolean res = true;
    if (raiz != null) {
        res = sonHijosMenoresRec(raiz, raiz.getPato());
    }
    return res;
}
```

```
private boolean sonHijosMenoresRec(NodoArbol nodo, int valor) {
    boolean res = true;
    if (nodo != null) {
        if (valor < nodo.getPato()) {
            res = false;
        } else {
            res = sonHijosMenoresRec(nodo.getIzquierdo(), nodo.getPato()) &&
                  sonHijosMenoresRec(nodo.getDerecho(), nodo.getPato());
        }
    }
    return res;
}
```

At least a root father

# Examen 2024 E.D. Árbol

Árbol {

- NodoArbol raiz
- Árbol()
- Árbol(int dato)

} NodoArbol (int dato, NodoArbol izq,  
NodoArbol der)

int getData()

NodoArbol getIzq()

NodoArbol getDer()

setIzq(NodoArbol izquierdo)

setDer(NodoArbol derecho)

public int sumarNodosNivelesPares()

int res = 0;

if (raiz != null) {

res = sumarNodosNivelesParesRec(raiz, 1);

}

return res;

private int sumarNodosNivelesParesRec(NodoArbol nodo, int nivel)

int res = 0;

if (nodo != null) {

if (nivel % 2 == 0) {

res = res + nodo.getData();

}

res = res + sumarNodosNivelesParesRec(nodo.getIzq(), nivel + 1) +  
sumarNodosNivelesParesRec(nodo.getDer(), nivel + 1);

}

return res;

}

# Examen Extraordinario 2023 Árboles

Árbol Binario Busqueda

$\left\{ \begin{array}{l} \text{NodoArbol raiz} \\ \text{int getClave()} \\ \text{NodoArbol getDerecho()} \\ \text{NodoArbol getIzquierdo()} \end{array} \right.$

```
public int getNivel(int clave) {
    int res = 0;
    if (raiz != null) {
        res = getNivelRec(raiz, clave, 1)
    }
    return res;
}
```

```
private int getNivelRec(NodoArbol nodo, int clave, int nivel) {
    if (nodo != null) {
        if (nodo.getData() == clave) {
            return 1;
        } else if (nodo.getData() > clave) {
            return getNivelRec(nodo.getIzquierdo(), clave, nivel + 1);
        } else {
            return getNivelRec(nodo.getDerecho(), clave, nivel + 1);
        }
    }
}
```

# Examen Extraordinario 2024 Árboles

Arbol {  
 } NodoArbol raiz  
 Arbol()  
 Arbol(int dato)

NodoArbol { int dato, NodoArbol izq,  
 NodoArbol der)  
 int getDato();  
 NodoArbol getIzquierdo();  
 NodoArbol getDerecho();  
 setDato(int dato);  
 setIzquierdo(NodoArbol izquierdo);  
 setDerecho(NodoArbol derecho);

public boolean esIsomorfia(Arbol otroArbol) {

boolean res = true;

if ((this.raiz != null && otroArbol.raiz == null) || (this.raiz == null && otroArbol.raiz != null)) {

res = false;

} else {

res = esIsomorfiaRec(this.raiz, otroArbol.raiz);

return res;

private boolean esIsomorfiaRec(NodoArbol nodo, NodoArbol otroNodo) {

boolean res = false;

if (otroNodo == null && nodo == null) {

res = true;

} else if (otroNodo != null && nodo != null) {

boolean izq = esIsomorfiaRec(nodo.getIzq(), otroNodo.getIzq());

boolean der = esIsomorfiaRec(nodo.getDerecho(), otroNodo.getDerecho());

res = izq && der;

}

return res;

# Examen Extraordinario 2024 Grafos

GrafoMA

matrizAdy  
int getNumVertices()  
boolean existeArista(int u, int v)  
boolean[] profundidadDesdeVertice(int v)

```
public boolean existeCaminoTresVertices(int vOrigen, int vDest, int vIter) {
    boolean res = false;
    boolean[] visitados = profundidadDesdeVertice(vOrigen);
    if (visitados[vIter]) {
        visitados = profundidadDesdeVertice(vIter);
        if (visitados[vDest]) {
            res = true;
        }
    }
    return res;
}
```

Origen → Iter  
↓  
Dest

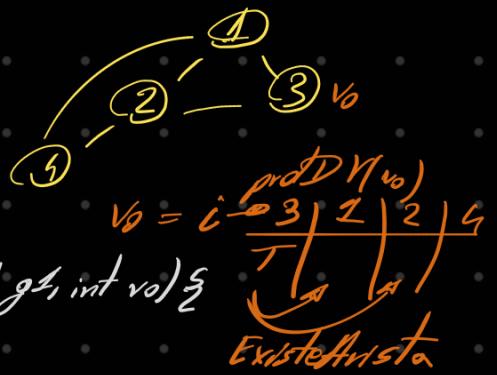
# Examen 2023 Grafos

## GrafoMA

```
boolean verticeEnRango(int v)  
int getNumVertices()  
boolean existeArista(int u, int v)  
boolean[] profundidadDesdeVertice(int v)
```

```
public void verticesComponenteConexaNoAdy(GrafoMA g1, int vo) {  
    if (!g1.verticeEnRango(vo)) {
```

```
        boolean[] vistos = g1.profundidadDesdeVertice(vo);  
        for (int i = 0; i < g1.getNumVertices(); i++) {  
            if (!vistos[i] && i != vo) {  
                if (!g1.existeArista(vo, i)) {  
                    System.out.println(i + " ");  
                }  
            }  
        }  
    }
```



ExistenteArista

# Examen 2024 Grafos

## Grafo Dirigido

### Grafo MA

```
boolean adj[][] matrizAdy  
int getNumVertices()  
boolean existeArista(int u, int v)  
boolean verticeEnRango(int u)
```

```
public Lista verticesAdyacentesA(int v){  
    if(!verticeEnRango(v)) return null;  
    Lista lista = new Lista();  
    for(int j = 0; j < getNumVertices(); j++){  
        if(existeArista(j, v) && !existeArista(v, j)){  
            lista.insertar(j);  
        }  
    }  
    return lista;  
}
```

### Lista

```
boolean insertar(int dato)  
boolean contiene(int dato)  
boolean borrar(int dato)  
Iterador getIterador()
```



## Ejercicio Grafo Deepseek

GrafoEnPuntos

agregarVertices(int v)  
agregarArista(int p, int v)  
int getNumVertices()  
boolean existeArista(int p, int v)  
boolean verticeEnGrafo(int v)

public TreeSet<Integer> verticeConMayorEntradaQueSalida()