UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

Técnicas de Programación Avanzada

Profesor:

Borja Monsalve Piqueras

Alumnos:

Paula Sáenz de Santa María Diez

Andrés Ramos García

Contenido

Código

Función gradoArbol

Esta función tiene como parámetro un árbol cualquiera (no tiene ninguna condición previa), y primero verificamos que el árbol no esté vacío, y recursivamente cogemos el número de hijos de cada nodo (esto es el grado), y lo guarda en una variable, hacemos un bucle recorriendo todos los hijos de éste (su LinkedList) y recogemos en otra variable el grado de cada hijo en cada iteración.
Con estos dos datos, se comparan y se queda con el mayor de los dos, haciendo que al final del recorrido del árbol devuelva el mayor grado de este.
Nodo con el mayor número de hijos = Grado del árbol.

```
public static <T extends Number> int gradoArbol(Arbol_N<T> arbol) {
    if (!arbol.esVacio()) {
        int grado = arbol.getNumHijos();
        for (int i = 1; i <= arbol.getNumHijos(); i++) {
            int gradoHijo = gradoArbol(arbol.getHijoN(i));
            if (gradoHijo > grado) {
                grado = gradoHijo;
            }
        }
        return grado;
    } else {
        return 0;
    }
} // Función gradoArbol
```

Función esABB

Esta función verifica si un árbol de grado N es un árbol Binario de búsqueda (ABB). Comienza por comprobar si el árbol está vacío, en cuyo caso se considera automáticamente como un ABB. Si el árbol no está vacío, se verifica que tenga como máximo dos hijos, ya que es uno de los requisitos de un árbol binario (AB).
Luego, se recorren recursivamente las partes izquierda y derecha del árbol. Finalmente, se combina el resultado de estas verificaciones (que se llevan a cabo en una función auxiliar), asegurando que tanto la parte izquierda como la derecha sean ABB para que el árbol completo sea considerado como tal. De lo contrario, devuelve false.

```
public static boolean esABB(Arbol_N<Integer> arbol) {
    if (arbol.esVacio()) {
        // Árbol vacío, considerado ABB
        return true;
    } else{
        //si el arbol tiene mas de dos hijos, no es ABB
        if (arbol.getNumHijos()>2){
            return false;
        } else{
            // creamos las variables de comprobacion
            boolean esABBParteIzquierda = true;
            boolean esABBParteDerecha = true;

            // Recorre la parte izquierda del árbol
            if (arbol.getNumHijos() > 0 && !arbol.getHijoN(1).esVacio()) {

                esABBParteIzquierda =
                    esABB_aux(arbol.getHijoN(1), arbol.getRaiz(), true)
                    &&  esABB(arbol.getHijoN(1));
            }
            // Recorre la parte derecha del árbol
            if (arbol.getNumHijos() > 1 && !arbol.getHijoN(2).esVacio()) {

                esABBParteDerecha =
                    esABB_aux(arbol.getHijoN(2), arbol.getRaiz(), false)
                    && esABB(arbol.getHijoN(2));
            }
            /*
            El árbol es un ABB si tanto la parte
            izquierda como la derecha son ABB
             */
            return esABBParteIzquierda && esABBParteDerecha;
        }
    }
} //Función esABB
```

Función esABB_aux

esABB_aux es una función auxiliar de la principal esABB, ya que necesitábamos su uso para poder llevar a cabo las comparaciones necesarias guardando un dato a largo plazo (mientras se recorre todo el subárbol izquierdo o derecho).
La principal función es comparar todos los datos del subárbol pasado por parámetro con el dato también pasado por parámetro.
Además, para llevar una cuenta de en qué subárbol estamos, metimos una verificación booleana para ello (true significa que estamos en el subárbol izquierdo, mientras que false significa que estamos en el subárbol derecho).
El bucle de la función recorre todos los hijos (siendo éste izquierda o derecha), mientras que la condición se mete en una nueva recursividad de comprobación sobre todos los subárboles.

```
private static boolean esABB_aux(Arbol_N<Integer> arbol, int numero, boolean subarbol) {
        if (subarbol && arbol.getRaiz() >= numero) {
            return false;
        } else if (!subarbol && arbol.getRaiz() <= numero) {
            return false;
        } else {
            boolean entrar = true;
            int i=1;
            while(i<= arbol.getNumHijos() && entrar){
                if (!esABB_aux(arbol.getHijoN(i), numero, subarbol)) {
                    // Si algún hijo no cumple la condición
                    entrar = false;
                }
                i++;
            }
            return entrar;
        }
    } //Función esABB_aux
```

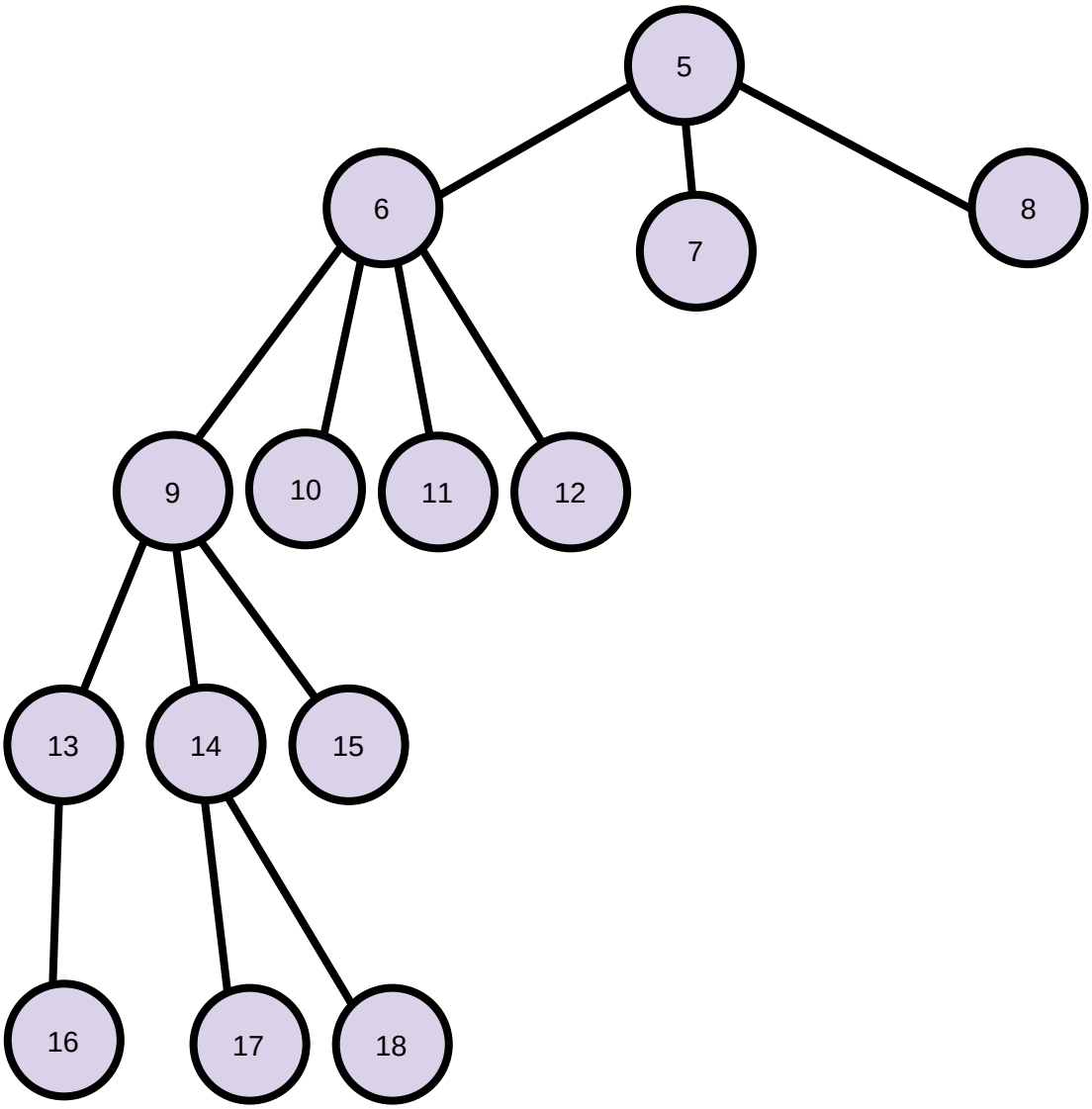
Función sonIguales

Esta función comprueba con un bucle while si son iguales (decidimos usar un bucle while ya que en el momento en el que encuentre una diferencia, ya va a ser distinto, por lo que no hace falta recorrerlo en su totalidad, significando esto que el peor caso es que sean iguales).
Nuestra primera verificación es que ambos árboles no estén vacíos, y más adelante se llevan a cabo dos comparaciones claves, la primera es que ambos tengan el mismo número de hijos, y seguidamente que ambas raíces que puede ver sean iguales.
Si entra en este condicional recorre todos los hijos (o hasta que encuentre una discrepancia entre ellos) y llama recursivamente a la función para poder llevar a cabo la misma dinámica con cada raíz y cada lista de hijos.

```
public static <T extends Number> boolean sonIguales(Arbol_N<T> arbol, Arbol_N<T> arbol1) {

    if (!arbol.esVacio() && !arbol1.esVacio()) {
        int cont = 1;
        boolean ig = true;

        if (arbol.getNumHijos() == arbol1.getNumHijos()
            && arbol.getRaiz() == arbol1.getRaiz()) {

            while (cont <= arbol.getNumHijos() && ig == true) {
                ig =  sonIguales(arbol.getHijoN(cont),
                                 arbol1.getHijoN(cont));
                cont++;
            }
            return ig;

        }else{
            return false;
        }

    } else if (!arbol.esVacio() || !arbol1.esVacio()) {
        return false;
    } else {
        return true;
    }
} //Función sonIguales
```

Dibujos con código

Función FormarArbol()

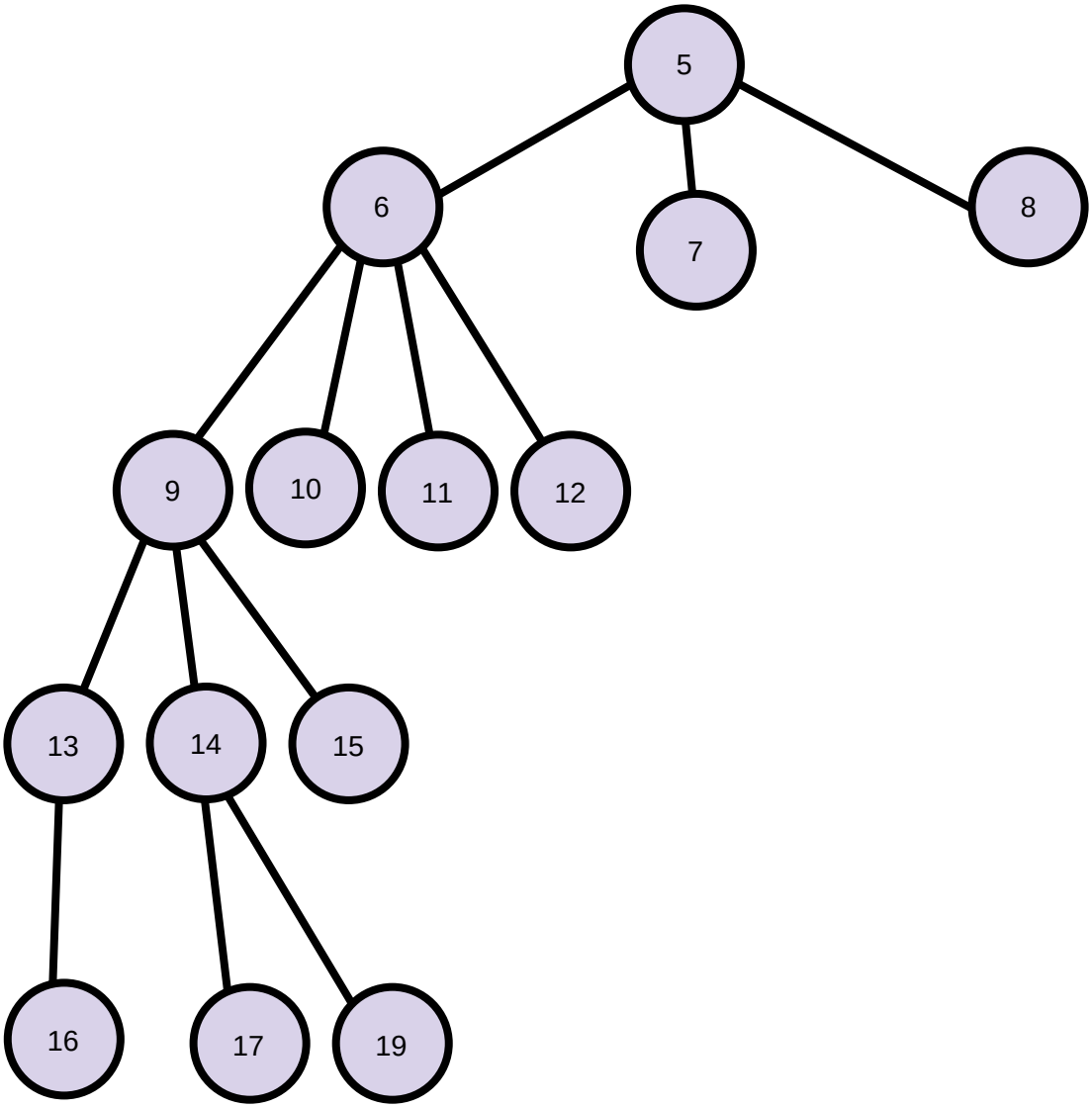



```
/* EJEMPLO:
 *              -------- 5 -------
 *             |         |        |
 *             6         7        8
 *           / | | \
 *         9 10 11 12
 *       / | \
 *      13 14 15
 *     /  / |
 *   16  17 18
 */
```

```
private static Arbol_N<Integer> formarArbol() {
        Arbol_N<Integer> n = new Arbol_N(18, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> m = new Arbol_N(17, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> l = new Arbol_N(16, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> k = new Arbol_N(15, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> j = new Arbol_N(14, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> ii = new Arbol_N(13, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> h = new Arbol_N(12, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> g = new Arbol_N(11, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> f = new Arbol_N(10, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> e = new Arbol_N(9, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> d = new Arbol_N(8, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> c = new Arbol_N(7, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> b = new Arbol_N(6, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> a = new Arbol_N(5, new LinkedList<Arbol_N.Nodo>());

        a.getHijos().add(b);
        a.getHijos().add(c);
        a.getHijos().add(d);

        b.getHijos().add(e);
        b.getHijos().add(f);
        b.getHijos().add(g);
        b.getHijos().add(h);

        e.getHijos().add(ii);
        e.getHijos().add(j);
        e.getHijos().add(k);

        ii.getHijos().add(l);

        j.getHijos().add(m);
        j.getHijos().add(n);


        return a;
    } //Funcion arbol
```

Función FormarArbol1()

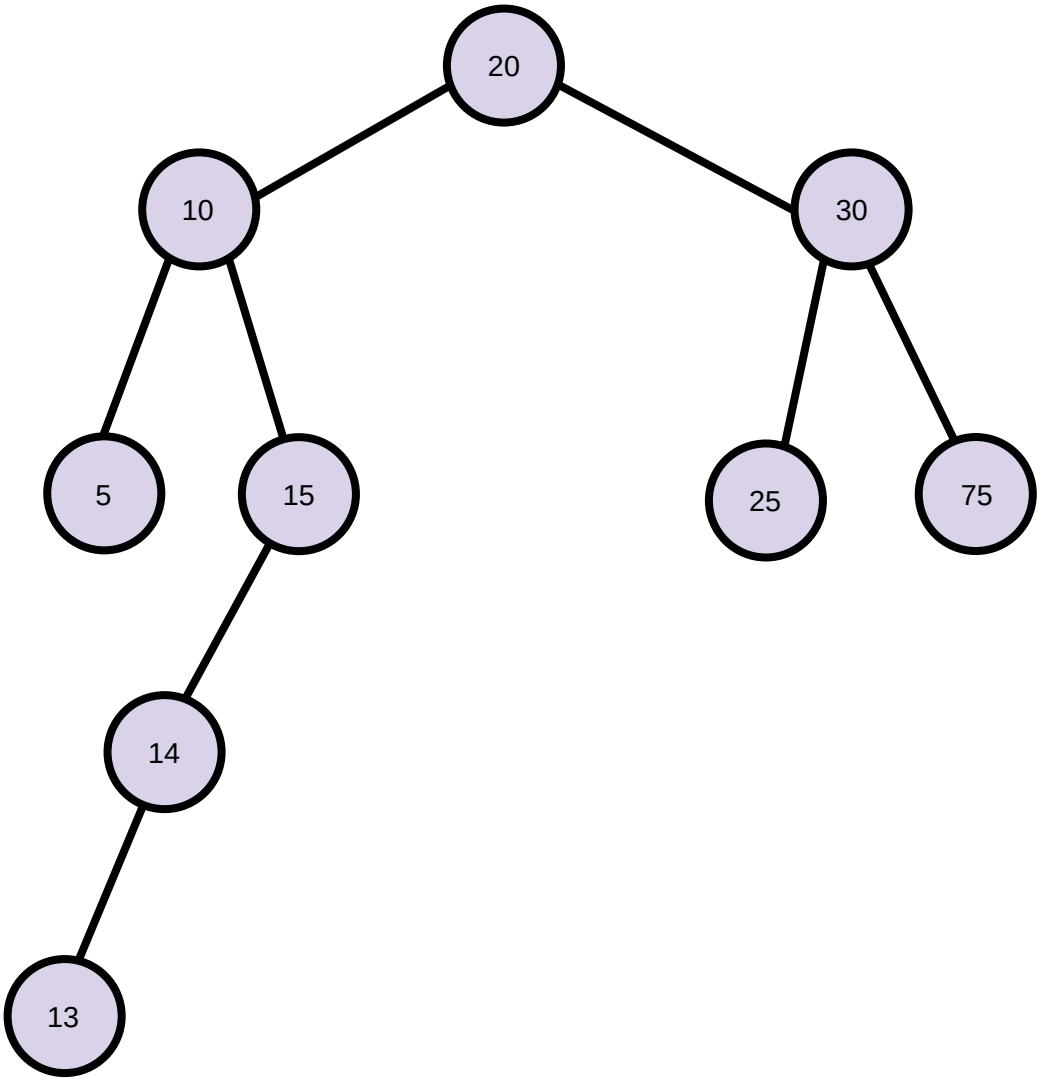
Su único cambio con el árbol anterior es que, en el último nodo, en vez de un 18, ponemos un 19 para la verificación de la función de sonIguales.



```java
    /* EJEMPLO:
     *             -------- 5 -------
     *            |         |        |
     *            6         7        8
     *          / | | \
     *         9 10 11 12
     *       / | \
     *      13 14 15
     *     /  / |
     *   16  17 19

private static Arbol_N<Integer> formarArbol1() {
    Arbol_N<Integer> n = new Arbol_N(19, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> m = new Arbol_N(17, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> l = new Arbol_N(16, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> k = new Arbol_N(15, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> j = new Arbol_N(14, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> ii = new Arbol_N(13, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> h = new Arbol_N(12, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> g = new Arbol_N(11, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> f = new Arbol_N(10, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> e = new Arbol_N(9, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> d = new Arbol_N(8, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> c = new Arbol_N(7, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> b = new Arbol_N(6, new LinkedList<Arbol_N.Nodo>());
    Arbol_N<Integer> a = new Arbol_N(5, new LinkedList<Arbol_N.Nodo>());

    a.getHijos().add(b);
    a.getHijos().add(c);
    a.getHijos().add(d);

    b.getHijos().add(e);
    b.getHijos().add(f);
    b.getHijos().add(g);
    b.getHijos().add(h);

    e.getHijos().add(ii);
    e.getHijos().add(j);
    e.getHijos().add(k);

    ii.getHijos().add(l);

    j.getHijos().add(m);
    j.getHijos().add(n);

    return a;
  } //Funcion arbol 1
```

Función FormarArbolABB()

Esta función la llevamos a cabo para verificar la función de esABB (Debería dar true)

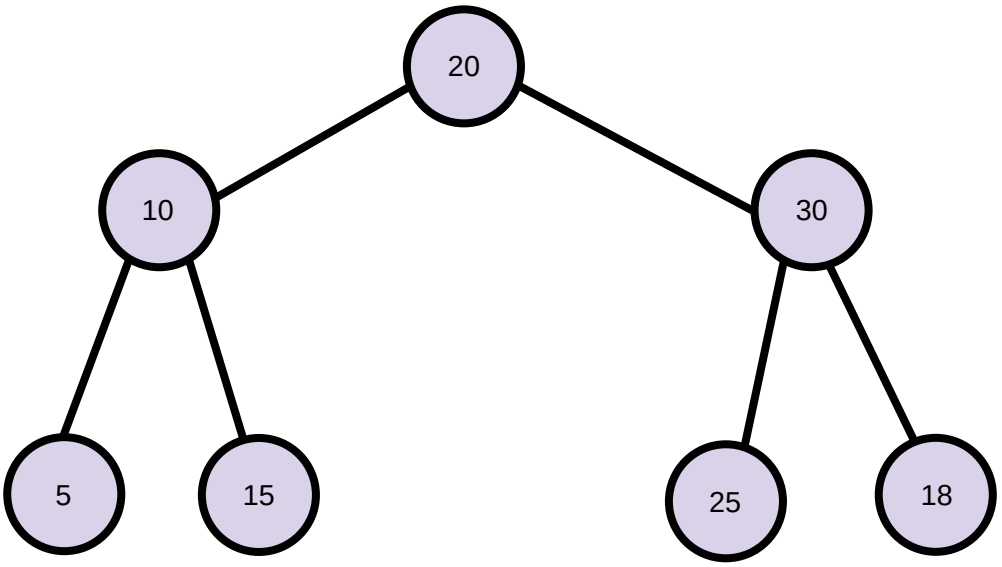



```
/* EJEMPLO: ahora no es ABB
 *              -------- a:20 -------
 *                    /      \
 *                  b: 10     c: 30
 *                  /  \      /  \
 *               e:5  d:15 g:25  f:75
 *                      /
 *                    h:16
 *                    /
 *                  j:13
 * */
```

```java
private static Arbol_N<Integer> formarArbolABB() {
        Arbol_N<Integer> j = new Arbol_N(13, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> h = new Arbol_N(16, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> g = new Arbol_N(25, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> f = new Arbol_N(75, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> e = new Arbol_N(5, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> d = new Arbol_N(15, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> c = new Arbol_N(30, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> b = new Arbol_N(10, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> a = new Arbol_N(20, new LinkedList<Arbol_N.Nodo>());

        a.getHijos().add(b);
        a.getHijos().add(c);

        b.getHijos().add(e);
        b.getHijos().add(d);


        c.getHijos().add(g);
        c.getHijos().add(f);



        d.getHijos().add(h);


        h.getHijos().add(j);

        return a;

    } //Funcion arbol ABB
```

Función FormarArbolABB1()

Este árbol se crea para poner a prueba de nuevo el




```
    /* EJEMPLO:
     *            -------- 20 -------
     *                 /     \
     *                10       30
     *               / \     / \
     *              5   15   25   18
     *
     */



private static Arbol_N<Integer> formarArbolABB1() {
        Arbol_N<Integer> g = new Arbol_N(18, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> f = new Arbol_N(25, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> e = new Arbol_N(5, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> d = new Arbol_N(15, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> c = new Arbol_N(30, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> b = new Arbol_N(10, new LinkedList<Arbol_N.Nodo>());
        Arbol_N<Integer> a = new Arbol_N(20, new LinkedList<Arbol_N.Nodo>());

        a.getHijos().add(b);
        a.getHijos().add(c);
```

```
        b.getHijos().add(e);
        b.getHijos().add(d);

        c.getHijos().add(f);
        c.getHijos().add(g);

        return a;

    } //Funcion arbol ABB1
```