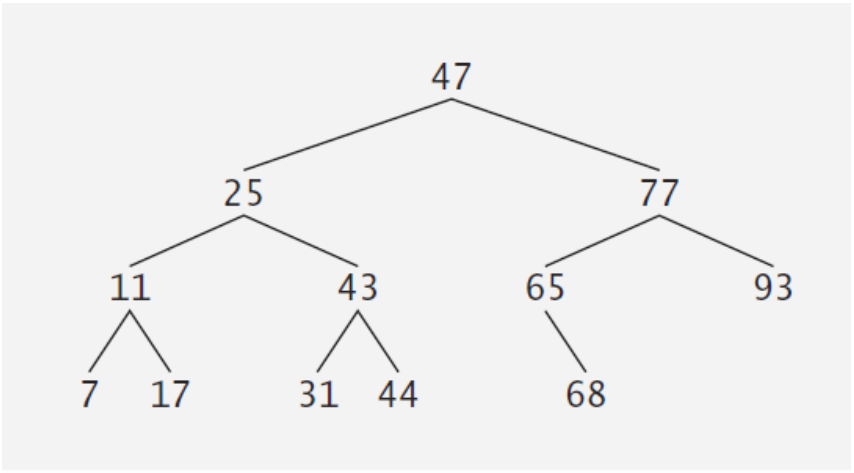
ÁRBOLES

Un árbol también es una estructura de datos multidimensional ya que los nodos de un árbol contienen dos o más enlaces. El nodo raíz es el primer nodo en un árbol. Los hijos de un nodo se llaman hermanos.

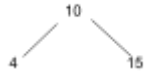
ÁRBOLES BINARIOS DE BÚSQUEDA.

En general un nodo de un árbol puede tener un número de hijos variable n. Un tipo de árbol concreto muy usado es el árbol binario donde cada nodo puede tener 0,1 o 2 hijos como mucho. Y un tipo de árbol binario concreto es el árbol binario de búsqueda (ABB). Los ABB son muy importantes y además son los más fáciles de manejar y será con los que más nos paremos en este boletín. Los ABB son árboles binarios sin valores de nodo duplicados y además los valores en cualquier subárbol izquierdo son menores que el valor del nodo padre de ese subárbol, y los valores en cualquier subárbol derecho son mayores que el valor del nodo padre de ese subárbol. Es decir "están ordenados".

EJEMPLO DE ÁRBOL BINARIO DE BÚSQUEDA:

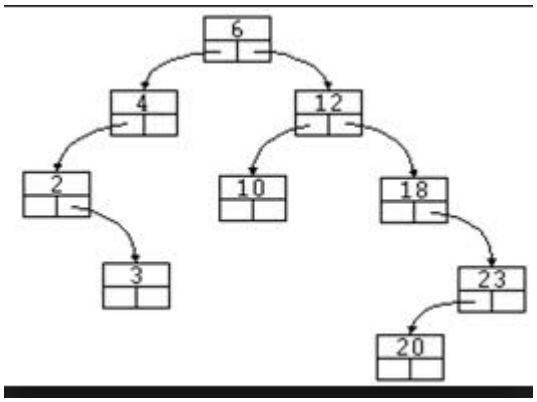


Se puede obtener el árbol anterior, con varios órdenes de inserción, por ejemplo con este:
47,25,77,11,43,65,93,7,17,31,44,68
pero observa que simplemente si intercambio el orden de los dos primeros
25,47,,77,11,43,65,93,7,17,31,44,68
el ABB es diferente (para empezar cambia la raíz)
Por lo tanto, la forma de un ABB está condicionada por el orden de inserción de los elementos, pero al mismo tiempo, puede ocurrir que órdenes de inserción diferentes generan el mismo ABB
por ejemplo 10,4,15 genera el mismo árbol que 10,15,4



Implementación de un ABB

Un ABB se puede implementar con un array pero es más fácil con nodos enlazados. Un ejemplo gráfico de ABB con nodos enlazados y clave de búsqueda int:



Un código Java para crear un ABB con nodos enlazados

Observa que el nodo de un ABB es diferente que el de una lista ya que necesita enlazarse con dos "hijos"

Supón que el programador de App es diferente que el de la clase Arbol. También supón que la clase Arbol desea ocultar la existencia de la clase NodoArbol al programador de la clase App para simplificar su manejo.

```
import java.util.Random;

class NodoArbol {

   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }

}

class Arbol {

   private NodoArbol raiz;

   public Arbol() {
      raiz = null;
   }

   // si el valor ya existe en el arbol, no inserta nada
   public void insertar(int valorInsertar) {
      if (raiz == null) {
         raiz = new NodoArbol(valorInsertar);
      } else {
         ayudanteInsertarNodo(raiz,valorInsertar);
      }
   }
   private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
      // inserta en el subárbol izquierdo
      if (valorInsertar < a.datos) {
         // inserta nuevo NodoArbol
         if (a.nodoIzq == null) {
            a.nodoIzq = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
         }
      }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
         if (a.nodoDer == null) {
            a.nodoDer = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoDer,valorInsertar);
         }
      }
   }
}

public class App {
  //el programador de App ignora la existencia de NodoArbol
   public static void main(String args[]) {
      Arbol arbol = new Arbol();
      int valor;
      Random numeroAleatorio = new Random();

      System.out.println("Insertando los siguientes valores: ");
      // inserta 10 enteros aleatorios de 0 a 99 en arbol
     //puede ser menos de 10 si se generan duplicados
      for (int i = 1; i <= 10; i++) {
         valor = numeroAleatorio.nextInt(100);
         System.out.print(valor + " ");
         arbol.insertar(valor);
```

```
        }
    }
}
```

Ejercicio: siguiendo el código anterior dibujar en papel paso a paso árbol resultante de las inserciones de 6,12,4,2,3,10,...

Recorridos básicos de un ABB: preorden, inorden y postorden

El recorrido de un árbol es un tema muy extenso:
https://es.wikipedia.org/wiki/Recorrido_de_%C3%A1rboles
En el ejemplo que sigue utilizamos preorden, inorden y postorden. Son recorridos que se aplican a cualquier tipo de árbol, no sólo a un ABB. En el caso de árboles no binarios en lugar de dos hijos tendremos una lista de n hijos con lo que en lugar de hablar del hijo izquierdo y el derecho hablamos del hijo 0, hijo 1, …, hijo n-1. Por ahora nos contentamos con recorrer un ABB.

- **Preorden**: (**raíz**, izquierdo, derecho). Para recorrer un árbol binario no vacío en preorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo de raíz:
    1. **Visite la raíz**
    2. Atraviese el sub-árbol izquierdo
    3. Atraviese el sub-árbol derecho
- **Inorden**: (izquierdo, **raíz**, derecho). Para recorrer un árbol binario no vacío en inorden (simétrico), hay que realizar las siguientes operaciones recursivamente en cada nodo:
    1. Atraviese el sub-árbol izquierdo
    2. **Visite la raíz**
    3. Atraviese el sub-árbol derecho
- **Postorden**: (izquierdo, derecho, **raíz**). Para recorrer un árbol binario no vacío en postorden, hay que realizar las siguientes operaciones recursivamente en cada nodo:
    1. Atraviese el sub-árbol izquierdo
    2. Atraviese el sub-árbol derecho
    3. **Visite la raíz**

aquí "visitar" significa hacer lo que se quiera con el dato de dicho nodo, por ejemplo, simplemente imprimirlo o leerlo para hacer algún cálculo, es decir, significa "procesar" aunque lo típico es decir "visitar"

EJEMPLO: Recorrido preorden, inorden y postorden

```
import java.util.Random;

class NodoArbol {

    NodoArbol nodoIzq;
    int datos;
    NodoArbol nodoDer;

    public NodoArbol(int datosNodo) {
        datos = datosNodo;
        nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
    }

}

class Arbol {

    private NodoArbol raiz;

    public Arbol() {
        raiz = null;
    }

    // si el valor ya existe en el arbol, no inserta nada
    public void insertar(int valorInsertar) {
```

```
        if (raiz == null) {
            raiz = new NodoArbol(valorInsertar);
        } else {
            ayudanteInsertarNodo(raiz,valorInsertar);
        }
    }
    private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
        // inserta en el subárbol izquierdo
        if (valorInsertar < a.datos) {
            // inserta nuevo NodoArbol
            if (a.nodoIzq == null) {
                a.nodoIzq = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
            }
        }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
            if (a.nodoDer == null) {
                a.nodoDer = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoDer,valorInsertar);
            }
        }
    }
    public void recorridoPreorden() {
        ayudantePreorden(raiz);
    }

     private void ayudantePreorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }

        System.out.print(nodo.datos+ " ");
        ayudantePreorden(nodo.nodoIzq);
        ayudantePreorden(nodo.nodoDer);
    }

    public void recorridoInorden() {
        ayudanteInorden(raiz);
    }

    private void ayudanteInorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }
        ayudanteInorden(nodo.nodoIzq);
        System.out.print(nodo.datos+ " ");
        ayudanteInorden(nodo.nodoDer);
    }


    public void recorridoPostorden() {
        ayudantePostorden(raiz);
    }


    private void ayudantePostorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }

        ayudantePostorden(nodo.nodoIzq);
        ayudantePostorden(nodo.nodoDer);
        System.out.print(nodo.datos+ " ");
    }

}

public class App {

    public static void main(String args[]) {
        Arbol arbol = new Arbol();
        int valor;
        Random numeroAleatorio = new Random();

        System.out.println("Insertando los siguientes valores: ");
        // inserta 10 enteros aleatorios de 0 a 99 en arbol
       //puede ser menos de 10 si se generan duplicados
        for (int i = 1; i <= 10; i++) {
            valor = numeroAleatorio.nextInt(100);
            System.out.print(valor + " ");
            arbol.insertar(valor);
        }
        System.out.println("\nRecorrido preorden .....");
        arbol.recorridoPreorden();
        System.out.println("\nRecorrido inorden .....");
        arbol.recorridoInorden();
        System.out.println("\nRecorrido postorden .....");
        arbol.recorridoPostorden();
    }
}
```

Observa la interesante propiedad de que si se listan los nodos del ABB en inorden nos da la lista de nodos ordenada. Por lo tanto, crear un ABB es una forma de ordenar una lista.

Ejercicio U6_2C_E1: Escribe un método que recorra el árbol en preorden pero que **no** use "ayudante". Partimos del siguiente código que hay que modificar.

```
class NodoArbol {
   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }
}

class Arbol {
   private NodoArbol raiz;
   public Arbol() {
      raiz = null;
   }

   // si el valor ya existe en el arbol, no inserta nada
   public void insertar(int valorInsertar) {
      if (raiz == null) {
         raiz = new NodoArbol(valorInsertar);
      } else {
         ayudanteInsertarNodo(raiz,valorInsertar);
      }
   }
   private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
      // inserta en el subárbol izquierdo
      if (valorInsertar < a.datos) {
         // inserta nuevo NodoArbol
         if (a.nodoIzq == null) {
            a.nodoIzq = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
         }
      }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
         if (a.nodoDer == null) {
            a.nodoDer = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoDer,valorInsertar);
         }
      }
   }
   public void recorridoPreorden() {
      ayudantePreorden(raiz);
   }

    private void ayudantePreorden(NodoArbol nodo) {
      if (nodo == null) {
         return;
      }

      System.out.print(nodo.datos+ " ");
      ayudantePreorden(nodo.nodoIzq);
      ayudantePreorden(nodo.nodoDer);
   }

}

public class App {
   public static void main(String args[]) {
      Arbol arbol = new Arbol();
      arbol.insertar(10);arbol.insertar(20);arbol.insertar(8);arbol.insertar(15);
      arbol.insertar(26);arbol.insertar(22);arbol.insertar(17);arbol.insertar(4);
      arbol.recorridoPreorden();

   }
}
```

Ejercicio U6_2C_E2: Escribe en la clase árbol un método existe() que toma como parámetro un entero y si el entero existe en el árbol devuelve true. Si no existe devuelve false.
Un pseudocódigo podría ser

```
si nodo==null
        false
si valorBuscado==nodo.datos
        true
sino valorBuscado<nodo.datos
        busca en subarbol izquierdo(y esto da true o false)
sino
        busca en subarbol derecho (y esto da true o false)
```

Observa que la búsqueda en un ABB es rapidísima ya que en cada decisión eliminamos la mitad de los nodos en que buscar.

Puedes usar el siguiente main para probarlo

```
public class App {
      public static void main(String args[]) {
         Arbol arbol = new Arbol();
         arbol.insertar(10);arbol.insertar(20);arbol.insertar(8);arbol.insertar(15);
         arbol.insertar(26);arbol.insertar(19);arbol.insertar(17);arbol.insertar(4);

         System.out.println("\nExiste el valor "+ 3 + "?: "+ arbol.existe(3));
         System.out.println("\nExiste el valor "+ 19 + "?: "+ arbol.existe(19));

      }
 }
```

Ejercicio U6_2C_E3: Ahora queremos también imprimir usando preorden una versión con tabulaciones y saltos de línea que ayuda a visualizar el árbol que hay en memoria que es dificil de ver al imprimirlo como una lista de nodos secuencial

```
public class App {

   public static void main(String args[]) {
      Arbol arbol = new Arbol();

      arbol.insertar(8);arbol.insertar(4);arbol.insertar(10);
      arbol.insertar(2);arbol.insertar(5);arbol.insertar(9);
      arbol.insertar(12);

      System.out.println("\nRecorrido preorden .....");
      arbol.recorridoPreorden();
      System.out.println("\nRecorrido preorden con indentaciones.....");
      arbol.recorridoPreordenConTAB();
   }
}
```

salida:

```
Recorrido preorden .....
8 4 2 5 10 9 12
Recorrido preorden con indentaciones.....
8
        4
                2
                5
        10
                9
                12
```

TRUCO: añade al método ayudanteecorridoPreordenConTAB() un parámetro String sangrado que vaya creciendo a medida que avanza la recursividad.

Ejercicio U6_2C_E4: añadir la impresión de valor NULL al ejercicio anterior

Una solución al ejercicio anterior es la siguiente.

```
 public void recorridoPreordenConTAB() {
      ayudantePreordenConTAB(raiz,"");
   }
   private void ayudantePreordenConTAB(NodoArbol nodo,String tab) {
      if (nodo == null) {
         return;
      }
```

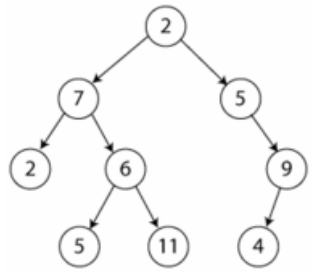
```
        System.out.println(tab+nodo.datos);
        tab=tab+"\t";
        ayudantePreordenConTAB(nodo.nodoIzq,tab);
        ayudantePreordenConTAB(nodo.nodoDer,tab);
    }
```

Ahora además queremos modificar este código de forma que también nos imprima los hijos null, y la salida sería:

```
Recorrido preorden .....
8 4 2 5 10 9 12
Recorrido preorden con indentaciones.....
8
        4
                2
                        null
                        null
                5
                        null
                        null
        10
                9
                        null
                        null
                12
                        null
                        null
```

OTROS ÁRBOLES BINARIOS

Un ABB es un tipo concreto de árbol binario, que se concreta por su forma de inserción. Veamos un ejemplo de árbol binario que no es un ABB ya que tiene un valor duplicado (el 2) y hay valores insertados sin el criterio < (el 7)



Además un árbol binario puede almacenar muchas cosas para las que no tiene sentido el concepto de ABB, por ejemplo una expresión aritmética

En muchos árboles binarios las inserciones se complican ya que el criterio de inserción no va a ser "tan cómodo y claro" como el del ABB.

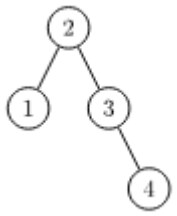
REPRESENTAR ÁRBOLES SIN NODOS ENLAZADOS.

Los nodos enlazados es con mucho la forma de representación de un árbol más usada y flexible, pero, no es la única. Piensa por ejemplo un fichero de texto XML, tiene dentro un árbol o estructura jerárquica de etiquetas. También se puede usar un array para almacenar un árbol y esto se puede hacer con diversas técnicas. Como muestra de cómo representar un árbol en un array, nosotros nos vamos a restringir a usar una forma, de las muchas que hay, y basada en un String. Ten en cuenta que un String no es más que

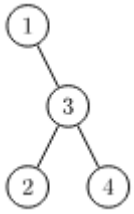
un array de caracteres. El String será una línea de texto que es la que contiene la descripción del árbol.

Crear un AB a partir de una línea de texto
El árbol se describe en una línea de texto. Se usa en "acepta el reto". Es una forma bastante retorcida de representar un árbol pero se adapta a la forma de trabajar del juez. En los enunciados específica que se describe el valor de un nodo e inmediatamente a continuación su hizo izquierdo y luego su hijo derecho, pero esto hay que interpretarlo recursivamente(lo que no se indica en los enunciados). Un árbol/subArbol vacío se indica con -1

Por ejemplo, para la línea de texto
2 1 -1 -1 3 -1 4 -1 -1
si represento cada subárbol entre paréntesis, la recursión implica usar paréntesis anidados
2 (1 (-1 -1)) (3 (-1 4 (-1 -1)))
y se corresponde con el árbol



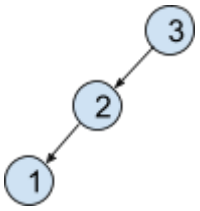
2 1 -1 -1 3 -1 4 -1 -1
otro ejemplo, la línea de texto
1 -1 3 2 -1 -1 4 -1 -1 se corresponde con



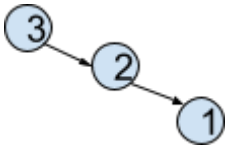
Observa que ambos son AB pero sólo el primero es ABB.

Nosotros a partir de la línea de texto podemos crear un árbol binario basado en nodos. Luego con el árbol ya en memoria los podemos procesar para hacer por ejemplo lo que indica un enunciado de *acepta el reto*. También de forma similar a lo que ocurria con arrays y matrices, es posible que podamos procesar la entrada al vuelo sin crear ningún árbol de nodos, pero igualmente hay que entender la estructura de árbol a la hora de hacer el procesamiento al vuelo.

sobre la impresión del árbol para comprobar resultados
Una vez que tengo el árbol creado en memoria a menudo me gustaría comprobar que lo hice bien, para eso recorro el árbol y lo imprimo. Recuerda que ya analizamos que dos o más órdenes de inserción pueden generar el mismo ABB, no confundas esto con lo siguiente: a diferencia de un ABB, dos o más árboles binarios diferentes pueden tener el mismo recorrido, por ejemplo el mismo preorden. El siguiente árbol es un árbol AB (y también ABB) con recorrido preorden
3 2 1



el siguiente AB diferente (y no es ABB) también tiene recorrido preorden 3 2 1.



Por esta razón, en el siguiente ejemplo usamos el recorrido preorden añadiendo tabuladores para comprobar que creamos bien el árbol.

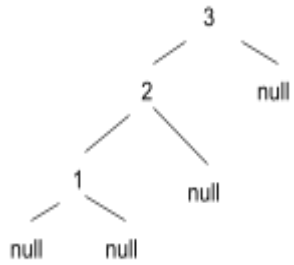
Código para la creación del árbol de nodos

A la clase *Arbol* le pasamos el texto ya troceado en nodos en un array de Strings, cada String es un nodo. Para crear el árbol de nodos a partir de un array de Strings la clase Arbol utiliza una variable *posArray* que lleva la cuenta de que elemento del array le toca incorporarse al árbol de nodos. LAS FUNCIONES RECURSIVAS DEBERÍAN TRABAJAR SÓLO CON VARIABLES LOCALES pero en este caso para hacer más legible el código permitimos el uso de esta variable global al objeto (una variable instancia). Una vez entendido podríamos fácilmente expresar la recursividad sólo con variables locales.

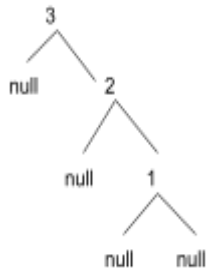
```
import java.util.Scanner;

class NodoArbol {
   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }
}

class Arbol {
   public NodoArbol raiz;
   String[] arbolString;//el arbol como una línea de
   int posArray=0;

   public Arbol(String[] arbolString) {
      raiz = null;
      this.arbolString=arbolString;
      this.crearArbol();
   }

   public void recorridoPreorden() {
      ayudantePreorden(raiz, "");
   }

   private void ayudantePreorden(NodoArbol nodo, String tab) {
      if (nodo == null) {
         System.out.println(tab + "null");
         return;
      }
```

```
        System.out.println(tab + nodo.datos);
        tab = tab + "\t";
        ayudantePreorden(nodo.nodoIzq, tab);
        ayudantePreorden(nodo.nodoDer, tab);
    }
    private  void crearArbol() {
        //la raiz está en arbolString[0]
        int dato=Integer.parseInt(arbolString[0]);

        if (dato == -1) {//árbol vacío
            raiz = null;

        } else {
            raiz = new NodoArbol(dato);
            ayudanteCrearArbol(raiz);


        }
    }

    private void ayudanteCrearArbol(NodoArbol padre) {
            //subarbol izquierdo del padre
            posArray++;
            int  dato=Integer.parseInt(arbolString[posArray]);
            if(dato!=-1){//si es -1 entonces nodoIzq queda con  null
                padre.nodoIzq=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoIzq);
            }

            //subarbol derecho del padre
            posArray++;
            dato=Integer.parseInt(arbolString[posArray]);
            if(dato!=-1){
                padre.nodoDer=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoDer);
            }

    }
}

public class App {

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        String[] arbolString = sc.nextLine().split(" ");
        Arbol arbol = new Arbol(arbolString);
        System.out.println("recorrido preorden con TABS");
        arbol.recorridoPreorden();

    }
}
```

prueba este código con
3 2 1 -1 -1 -1 -1
que se corresponde gráficamente con el árbol, y que debes comparar con la salida en preorden con tabs



y con
3 -1 2 -1 1 -1 -1



Ejercicio U6_2C_E5: Altura de un AB
La altura de un AB (mismo procedimiento si es ABB) se podría calcular con el siguiente pseudocódigo.

```
si el árbol es vacío
        altura 0
sino
        1 + máximo(alturasubarbolizquierdo,alturasubarbolderecho)
```

con el código del último ejemplo que crea un árbol dinámicamente a partir de una línea de texto, incorpora el método que calcule la altura para obtener una entrada/salida similar a la siguiente:

```
run:
4 2 3 8 -1 -1 -1 -1 -1
altura:4
```

Ejercicio U6_2C_E6: Máximo y mínimo de un AB
Dado un árbol imprimir su valor máximo y mínimo. Ejemplo:

```
run:
4 2 3 8 -1 -1 -1 -1 -1
Máximo:8
Minimo:2
```

Podrías obtener una lista de un recorrido(preorden u otro cualquiera) y buscar el máximo y mínimo en dicha lista. Pero utiliza mejor un código del tipo

```
max:
    Si nodo vacio
            return Integer.MIN_VALUE (el valor más pequeño posible)
    sino
            return elMáximoDe(nodo.datos, max(nodo.nodoIzquierdo), max(nodo.nodoDerecho))
```

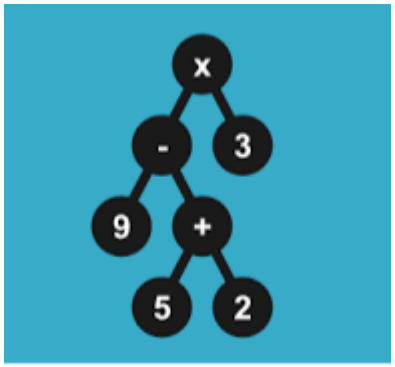
Ejercicio U6_2C_E7: Acepta el reto "Altura de un árbol binario" id 290

No es necesario crear un árbol de nodos, se puede procesar directamente la entrada. Pero no está mal crear un árbol de nodos en memoria y luego usarlo para hacer el cálculo, es quizá así incluso más didáctico. Puedes resolverlo como gustes. Lo más fácil quizá sea adaptar el código anterior, por ejemplo usando un char[] y en lugar de comparar con -1 con '.' etc.

Ejercicio U6_2C_E8:

Árbol binario que almacena expresión aritmética

Un árbol de expresión es un árbol binario en el que cada nodo no hoja se corresponde con un operador y cada nodo hoja se corresponde con un operando. Observa el siguiente el árbol de expresión



La estructura del árbol determina el orden de las operaciones, la forma del árbol produce un efecto similar a cuando escribimos paréntesis en una expresión con notación infija

Si recorres el árbol INORDEN puedes observar que almacena la siguiente expresión aritmética en notación infija.

((9 - (5 + 2))x 3)

Es una expresión totalmente parentizada, por cada operador hay siempre un paréntesis de apertura y otro de cierre. Hay paréntesis que se podrían suprimir pero para nuestro objetivo es más fácil pensar en una expresión totalmente parentizada.

El siguiente código, por el momento incompleto, obtendría del árbol anterior la expresión anterior.

```
class NodoArbol {
   NodoArbol nodoIzq;
   String datos;
   NodoArbol nodoDer;
   public NodoArbol(String datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null;
   }
}
class Arbol {
   NodoArbol raiz;
}
public class App {
    public static void main(String args[]) {
      Arbol arbol = new Arbol();
```

```
      NodoArbol n=new NodoArbol("x");
      arbol.raiz=n;
      NodoArbol n1=new NodoArbol("-");
      NodoArbol n2=new NodoArbol("3");
      n.nodoIzq=n1;
      n.nodoDer=n2;
      NodoArbol n11=new NodoArbol("9");
      NodoArbol n12=new NodoArbol("+");
      n1.nodoIzq=n11;
      n1.nodoDer=n12;
      NodoArbol n121=new NodoArbol("5");
      NodoArbol n122=new NodoArbol("2");
      n12.nodoIzq=n121;
      n12.nodoDer=n122;
      //arbol.imprimir();
   }
}
```

SE PIDE:
descomentar la última instrucción del main y escribir el código necesario en la clase Árbol para que se imprima la expresión formato infijo correctamente parentizada, es decir:
((9 - (5 + 2))x 3)

NO TODO ES RECURSIVIDAD CON ÁRBOLES.

La recursividad en los árboles suele facilitar las cosas, pero hay operaciones que se realizan mejor sin recursividad, por ejemplo si queremos recorrer un árbol en anchura.

Recorrido en anchura

La forma más fácil de recorrer un árbol en anchura es utilizando una cola auxiliar. Pronto manejaremos una cola del JDK y se simplificará el siguiente ejemplo que por el momento utiliza nuestra cola casera. La cola en lugar de almacenar *int* o *char* va a almacenar *NodoArbol*.
Usamos un fichero a parte para la cola. En el código de la cola ya visto, simplemente cambiamos int por NodoArbol, es decir, el campo dato antes era un int y ahora es tipo NodoArbol

//MiCola.java

```
class Nodo {

   private Nodo sig;
   private NodoArbol dato;



   public Nodo(NodoArbol dato, Nodo sig) {
      this.dato = dato;
      this.sig = sig;
   }

   public void setSiguiente(Nodo sig) {
      this.sig = sig;
   }

   public Nodo getSiguiente() {
      return sig;
   }

   public NodoArbol getDato() {
      return dato;
```

```
    }
}

class MiCola  {

    private Nodo primero = null;
    private Nodo ultimo = null;

    public boolean esVacia() {
        //vacia si primero==ultimo==null
        return ultimo == null;
    }

    public void encolar(NodoArbol dato) {
        if (esVacia()) {
            ultimo = new Nodo(dato,null);
            primero = ultimo;
        } else {
            Nodo temp = new Nodo(dato, ultimo);
            ultimo = temp;
        }

    }

    public NodoArbol desencolar() {
        NodoArbol dato = primero.getDato();
        //recorrer la cola para hacer el segundo el primero

        if (primero == ultimo) {
            //si sólo hay un elemento al borrar la cola queda vacía
            ultimo = null;
            primero = null;
        } else {
            Nodo temp = ultimo;
            while (temp.getSiguiente() != primero) {
                temp = temp.getSiguiente();
            }
            primero = temp;
        }

        return dato;
    }

}
```

El código del árbol, utiliza en el método recorridoAnchura() la cola anterior

```
import java.util.Random;

import java.util.Scanner;

class NodoArbol {
    NodoArbol nodoIzq;
    int datos;
    NodoArbol nodoDer;

    public NodoArbol(int datosNodo) {
        datos = datosNodo;
        nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
    }
}

class Arbol {
    public NodoArbol raiz;
    String[] arbolString;//el arbol como una línea de
    int posArray=0;

    public Arbol(String[] arbolString) {
        raiz = null;
        this.arbolString=arbolString;
        this.crearArbol();
    }

    public void recorridoPreorden() {
```

```
        ayudantePreorden(raiz, "");
    }

    private void ayudantePreorden(NodoArbol nodo, String tab) {
        if (nodo == null) {
            System.out.println(tab + "null");
            return;
        }

        System.out.println(tab + nodo.datos);
        tab = tab + "\t";
        ayudantePreorden(nodo.nodoIzq, tab);
        ayudantePreorden(nodo.nodoDer, tab);
    }
    private  void crearArbol() {
        //la raiz está en arbolString[0]
        int dato=Integer.parseInt(arbolString[0]);

        if (dato == -1) {//árbol vacío
            raiz = null;

        } else {
            raiz = new NodoArbol(dato);
            ayudanteCrearArbol(raiz);


        }
    }

    private void ayudanteCrearArbol(NodoArbol padre) {
            //subarbol izquierdo del padre
            posArray++;
            int  dato=Integer.parseInt(arbolString[posArray]);
            if(dato!=-1){//si es -1 entonces nodoIzq queda con  null
                padre.nodoIzq=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoIzq);
            }

            //subarbol derecho del padre
            posArray++;
            dato=Integer.parseInt(arbolString[posArray]);
            if(dato!=-1){
                padre.nodoDer=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoDer);
            }

    }
    void recorridoAnchura() {
        MiCola cola = new MiCola();
        if (raiz == null) {
            return;
        }
        cola.encolar(raiz);
        while (!cola.esVacia()) {
            NodoArbol n = cola.desencolar();
            System.out.print(n.datos + " ");

            if (n.nodoIzq != null) {
                cola.encolar(n.nodoIzq);
            }
            if (n.nodoDer != null) {
                cola.encolar(n.nodoDer);
            }

        }
    }

}

public class App {

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        //10 66 7 -1 -1 -1 4 9 -1 -1 5 -1 88 -1 -1
        String[] arbolString = sc.nextLine().split(" ");
        Arbol arbol = new Arbol(arbolString);
```

```
        System.out.println("recorrido preorden con TABS");
        arbol.recorridoPreorden();
        System.out.println("\nrecorrido en anchura");
        arbol.recorridoAnchura();


    }
}
```

Árbol n-ario

Cada nodo puede tener de 0 a n hijos. Como el número de hijos es variable, en lugar de usar variables usamos una lista(un array por el momento).

Los métodos recursivos en lugar invocar al hijo izquierdo y/o derecho, tendrán en su interior un bucle para recorrer el array de hijos

Ejemplo:

```
class NodoArbol {
    public String path;
    public NodoArbol[] hijos; //un array de nodos

    public NodoArbol() {//no haría falta escribirlo
        this.path = null;
        hijos = null; //recien creado un nodo, no tiene hijos
    }

}

class Arbol {

    public  NodoArbol raiz;

    public Arbol() {
        raiz = null;
    }
    void imprimirArbol(NodoArbol raiz,String tab){
        System.out.println(tab+ raiz.path);
        if(raiz.hijos!=null){
            for(NodoArbol nodo:raiz.hijos){
                imprimirArbol(nodo,tab+"\t");
            }
        }
     }

}
public class App{
    public static void main(String args[]) {
        Arbol arbol = new Arbol();
        //un nodo raiz con 3 hijos
        arbol.raiz=new NodoArbol();
        arbol.raiz.path="C:\\";
        arbol.raiz.hijos= new NodoArbol[]{new NodoArbol(),new NodoArbol(),new NodoArbol()};

        //crear hijo1
        NodoArbol nodo0= arbol.raiz.hijos[0];
        nodo0.path="mis ficheros";
        //nodo 0  tiene otros 3 hijos que ya son hojas (sin hijos)
        nodo0.hijos= new NodoArbol[]{new NodoArbol(),new NodoArbol(),new NodoArbol()};
        NodoArbol nodo00=nodo0.hijos[0];
        nodo00.path="fotos";

        NodoArbol nodo01=nodo0.hijos[1];
        nodo01.path="videos";

        NodoArbol nodo02=nodo0.hijos[2];
```

```
        nodo02.path="documentos";

        //los otros hijos de la raiz para simplificar son hojas
        NodoArbol hijo1= arbol.raiz.hijos[1];
        hijo1.path="System";

        NodoArbol hijo2= arbol.raiz.hijos[2];
        hijo2.path="Windows";

        arbol.imprimirArbol(arbol.raiz,"");

    }
}
```