Universidad Autónoma de Zacatecas

Unidad Académica de Ingeniería Eléctrica

Programa Académico de Ingeniería de Software.

Nombre de la Práctica Árboles.

Numero de Práctica 24

Nombre de la carrera Ingeniería de Software

Nombre de la materia Lab. Estructuras de Datos

Nombre del alumno Jesús Manuel Juárez Pasillas

Nombre del docente Aldonso Becerra Sánchez

Fecha: 18/10/2021

Práctica 24: Árboles.

Introducción:

Los arboles binarios nos permiten almacenar nodos los cuales tendrán cero, uno o máximo dos nodos hijos los cuales se comportan igual, siempre partiendo de un nodo raíz que es el padre de todos los nodos. En los arboles binarios es muy fácil hacer métodos recursivos por su estructura, aunque se tienen que hacer dos llamadas recursivas ya que es una para el hijo izquierdo y otro para el hijo derecho del nodo.

Desarrollo:

En la primera actividad de esta practica se pide realizar varios métodos con los cuales identificar los operadores y operandos de una expresión aritmética almacenada en un árbol binario donde cada uno de sus nodos era un elemento de la expresión.

Primero se pide introducir la expresión aritmética en el árbol binario, esto con el fin de tener separados todos sus elementos, luego de eso se pidió sacar en una lista encadenada los operadores, los cuales se van a identificar ya que son las raíces de todos los subárboles, además también se pido agregar todos los operandos en una lista encadenada hash la cual tendrá de clave el valor de la nodo y de valor el valor que se le quiera dar a la variable (si es variable, si no se pone el valor que tiene), estos nodos se identifican por ser las hojas del árbol.

Para realizar esto se hicieron dos métodos los cuales en si son solo uno, ya que uno es el método publico el cual llama al método privado el cual es recursivo.

- Crea una lista encadenada con una lista encadenada y una lista hash y llama al método sacarOperadoresYOperandos.
  `public ListaEncadenada sacarOperadoresYOperandos().`

- Saca los operadores y operandos, en una lista encadenada y una lista hash respectivamente.
  `private ListaEncadenada sacarOperadoresYOperandos(NodoDoble tmp,ListaEncadenada lista).`

Después de esto se pido que, a partir del primer árbol creado, se creara otro automáticamente (el usuario no lo crea), pero suplantando las variables por un valor dado por el usuario, luego de esto imprimir los arboles en las tres formas vistas en clase.

Para esto también se hicieron dos métodos con los cuales uno es recursivo y el otro lo llama a el para regresar un valor.

- Crea un árbol binario y llama al método suplantarVariables
  `public ArbolBinario suplantarVariables().`

- Rellena el árbol creado con el actual, suplantando variables.
  `private void suplantarVariables(NodoDoble tmp, NodoDoble tmp2).`

En la segunda actividad se pidió agregar varios métodos de utilidad en la clase ArbolBinario los cuales tienen que ser recursivos.

Estos métodos son:

- Invoca el método altura pasándole el parámetro raíz.
  `public int altura().`

  Saca la altura que tiene el árbol.
  `private int altura(NodoDoble subArbol, int altura).`

- Manda llamar al método alturaNodo con los parámetros elemento, raíz y 1.
  `public int alturaNodo(Object elemento).`

  Busca la altura del nodo que contenga a elemento.
  `private int alturaNodo(Object elemento, NodoDoble subArbol, int altura).`

- Crea una lista hash a la cual le agrega un elemento por cada nivel del árbol, con la clave "Altura x" y el valor 0.
  `public ListaEncadenadaHash numElementosXNivel().`

  Le va sumando uno a la altura en la lista hash en la posición correspondiente.
  `private ListaEncadenadaHash numElementosXNivel(NodoDoble tmp, int altura,ListaEncadenadaHash lista).`

- Evalúa si la raíz es igual al elemento y dice que es el nodo raíz y no tiene padre si no son iguales llama al método raizIntermedioHoja.
  `public String raizIntermedioHoja(Object elemento).`

  Busca el nodo con el elemento dado y regresa una cadena diciendo si es nodo intermedio o hoja y cuál es su padre.
  `private String raizIntermedioHoja(Object elemento,NodoDoble tmp).`

Todos estos métodos son muy útiles para saber información del árbol, como el de su altura, saber a que altura esta un nodo en particular, saber el numero de elementos por altura, y saber el tipo de nodo que es, toda esta información es muy útil a la hora de querer utilizar un árbol binario.

Como todos los métodos tienen que ser recursivos y estos tenían que recorrer el árbol se tuvieron que hacer dos métodos para solo una acción ya que el método recursivo necesita saber el nodo en el que esta posicionado para saber cualquier cosa

Nota: Toda la documentación del proyecto esta agregada en la carpeta “doc” dentro de la carpeta del proyecto (“edylab_2021_24/doc”).

Capturas del programa funcionando:

```
Introduce la raiz del árbol: +
¿El nodo + tiene hijo izquierdo? [S/N] s
Introduce el dato del hijo izquierdo de + : x
¿El nodo x tiene hijo izquierdo? [S/N] n
¿El nodo x tiene hijo derecho? [S/N] n
¿El nodo + tiene hijo derecho? [S/N] s
Introduce el dato del hijo derecho de +: *
¿El nodo * tiene hijo izquierdo? [S/N] s
Introduce el dato del hijo izquierdo de * : /
¿El nodo / tiene hijo izquierdo? [S/N] s
Introduce el dato del hijo izquierdo de / : resta
¿El nodo resta tiene hijo izquierdo? [S/N] n
¿El nodo resta tiene hijo derecho? [S/N] n
¿El nodo / tiene hijo derecho? [S/N] s
Introduce el dato del hijo derecho de /: 4
¿El nodo 4 tiene hijo izquierdo? [S/N] n
¿El nodo 4 tiene hijo derecho? [S/N] n
¿El nodo * tiene hijo derecho? [S/N] s
Introduce el dato del hijo derecho de *: ^
¿El nodo ^ tiene hijo izquierdo? [S/N] s
Introduce el dato del hijo izquierdo de ^ : y
¿El nodo y tiene hijo izquierdo? [S/N] n
¿El nodo y tiene hijo derecho? [S/N] n
¿El nodo ^ tiene hijo derecho? [S/N] s
Introduce el dato del hijo derecho de ^: 2
¿El nodo 2 tiene hijo izquierdo? [S/N] n
¿El nodo 2 tiene hijo derecho? [S/N] n
```

```
Ingresa el valor para x: 5
Ingresa el valor para resta: 7
Ingresa el valor para y: 7
Operadores:
+ -> / -> * -> ^ -> null
Operandos:
x-5.0 -> resta-7.0 -> 4-4.0 -> y-7.0 -> 2-2.0 -> null
Nuevo arbol:
Introduce el valor de x: 5
Introduce el valor de resta: 7
Introduce el valor de y: 7
Arbol original:
PreOrden:
+ x * / resta 4 ^ y 2
PostOrden:
x resta 4 / y 2 ^ * +
InOrden:
x + resta / 4 * y ^ 2

Arbol con variables suplantadas:
PreOrden:
+ 5.0 * / 7.0 4.0 ^ 7.0 2.0
PostOrden:
5.0 7.0 4.0 / 7.0 2.0 ^ * +
InOrden:
5.0 + 7.0 / 4.0 * 7.0 ^ 2.0
```

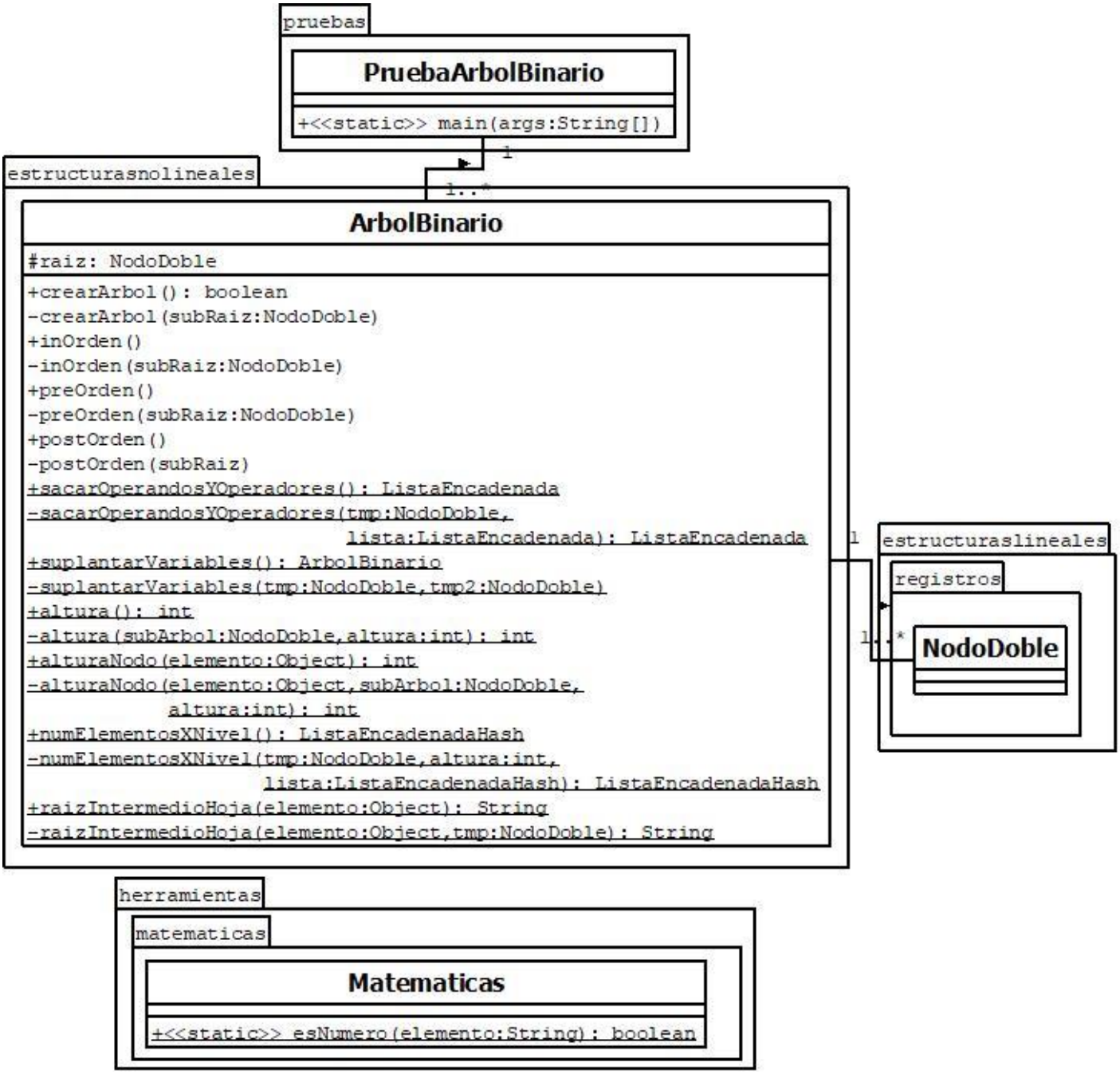
```
Metodos recursivos en arbol binario:
Altura del arbol: 4
Ingresa el nodo a buscar (arbol original): ^

Altura del nodo(^): 3
Numero de elementos por nivel:
Altura 1-1 -> Altura 2-2 -> Altura 3-2 -> Altura 4-4 -> null
Ingresa el nodo a evaluar: resta

Tipo nodo(resta): Nodo hoja, y su padre es: /
```

Código agregado:

Solo los métodos subrayados son nuevos.



Pre-evaluación:

Pre-Evaluación para prácticas de Laboratorio de Estructuras de Datos	PRE-EVALUACIÓN DEL ALUMNO
CUMPLE CON LA FUNCIONALIDAD SOLICITADA.	Sí
DISPONE DE CÓDIGO AUTO-DOCUMENTADO.	Sí
DISPONE DE CÓDIGO DOCUMENTADO A NIVEL DE CLASE Y MÉTODO.	Sí
DISPONE DE INDENTACIÓN CORRECTA.	Sí
CUMPLE LA POO.	Sí
DISPONE DE UNA FORMA FÁCIL DE UTILIZAR EL PROGRAMA PARA EL USUARIO.	Sí
DISPONE DE UN REPORTE CON FORMATO IDC.	Sí
LA INFORMACIÓN DEL REPORTE ESTÁ LIBRE DE ERRORES DE ORTOGRAFÍA.	Sí
SE ENTREGÓ EN TIEMPO Y FORMA LA PRÁCTICA.	Sí
INCLUYE LA DOCUMENTACIÓN GENERADA CON JAVADOC.	Sí
INCLUYE EL CÓDIGO AGREGADO EN FORMATO UML.	Sí
INCLUYE LAS CAPTURAS DE PANTALLA DEL PROGRAMA FUNCIONANDO.	Sí
LA PRÁCTICA ESTÁ TOTALMENTE REALIZADA (ESPECIFIQUE EL PORCENTAJE COMPLETADO).	100%
Observaciones:	

Conclusión:

Los arboles binarios son de gran utilidad a la hora de separar expresiones aritméticas, ya que en cada nodo queda un elemento de esta. Y con esto es muy fácil identificar operandos y operadores, además de las variables y poder sustituir su valor.

En cuanto a los métodos que se requieren tener en un árbol binario son sencillos ya que estos son fáciles de sacar, además de que proporcionan información de valor en su salida. Es por esto que es necesario tenerlos. Los métodos de los arboles suelen ser naturalmente recursivos por lo que la recursión es el método principal por el cual resolver una problemática.