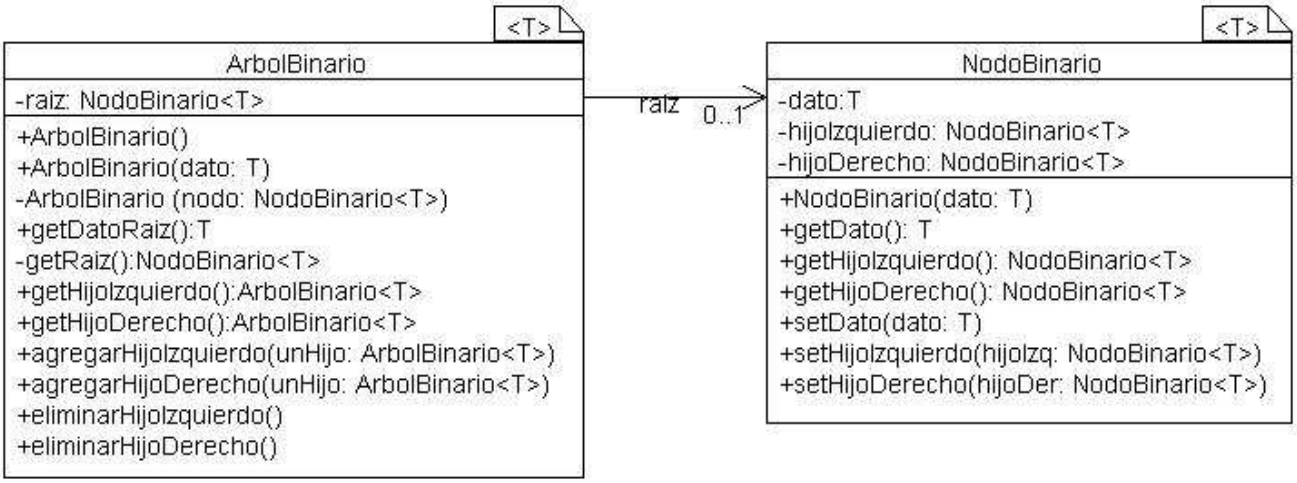
UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2013

Trabajo Práctico 4
Árboles Binarios

Objetivos
- Representar árboles binarios e implementar las operaciones de la abstracción
- Realizar distintos tipos de recorridos sobre árboles binarios
- Describir soluciones utilizando árboles binarios

Ejercicio 1

Considere la siguiente especificación de la clase **ArbolBinario** (con la representación hijo izquierdo e hijo derecho).



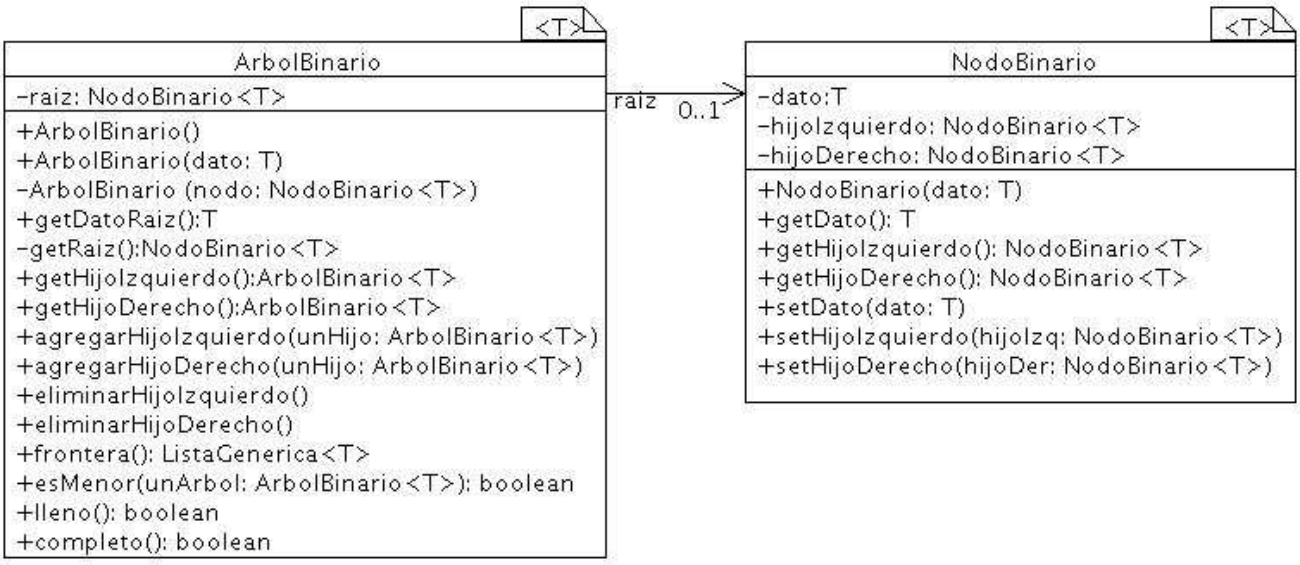
El constructor **ArbolBinario()** inicializa un árbol binario vacío, es decir, la raíz en null.
El constructor **ArbolBinario(T dato)** inicializa un árbol que tiene como raíz un nodo binario. Este nodo tiene el dato pasado como parámetro y ambos hijos nulos.
El constructor **ArbolBinario (NodoBinario<T> nodo)** inicializa un árbol donde el nodo pasado como parámetro es la raíz. (Notar que **NO** es un método público).
El método **getRaiz():NodoBinario<T>**, retorna el nodo ubicado en la raíz del árbol. (Notar que **NO** es un método público).
El método **getDatoRaiz():T**, retorna el dato almacenado en el NodoBinario raíz del árbol.
Los métodos **getHijoIzquierdo():ArbolBinario<T>** y **getHijoDerecho():ArbolBinario<T>**, retornan los hijos izquierdo y derecho respectivamente de la raíz del árbol. Tenga en cuenta que los hijos izquierdo y derecho del NodoBinario raíz del árbol son NodosBinarios y usted debe devolver ArbolesBinarios, por lo tanto debe usar el constructor privado **ArbolBinario (NodoBinario<T> nodo)** para obtener el árbol binario correspondiente.
El método **agregarHijoIzquierdo(ArbolBinario<T> unHijo)** y **agregarHijoDerecho(ArbolBinario<T> unHijo)** agrega un hijo como hijo izquierdo o derecho del árbol. Tenga presente que unHijo es un ArbolBinario y usted debe enganchar un NodoBinario como hijo. Para ello utilice el método privado getRaiz.
El método **eliminarHijoIzquierdo()** y **eliminarHijoDerecho()**, eliminan el hijo correspondiente NodoBinario raíz del árbol receptor.

a) Analice la implementación en JAVA de las clases **ArbolBinario** y **NodoBinario** brindadas por la cátedra.

UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2013

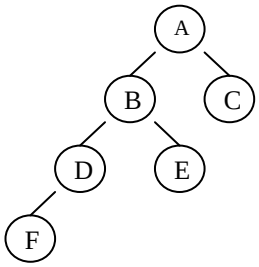
Ejercicio 2

Agregue a la clase **ArbolBinario** los siguientes métodos y constructores:



Con el fin de que gradualmente vaya realizando implementaciones cada vez mas complejas, para cada uno de los incisos solicitados se indica la complejidad en la siguiente escala: inicial y medio.

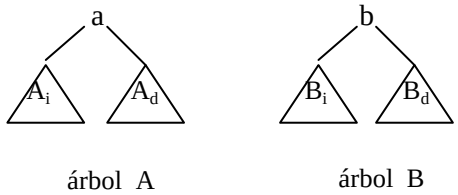
a) **frontera(): ListaGenerica<T>**. Se define **frontera** de un árbol binario, a las hojas de un árbol binario recorridos de izquierda a derecha. **Pista:** Recorrer el árbol en preorden, si el nodo es una hoja se agrega a la lista, sino se sigue recorriendo y buscando hojas. Complejidad: inicial.



La lista deberá devolver:
F,E,C

b) **esMenor(ArbolBinario<T> unArbol): boolean.** Se define una relación de orden entre árboles binarios de enteros no nulos de la siguiente forma:

$$A < B \begin{cases} a < b; \\ a = b \text{ y } Ai < Bi \\ a = b \text{ y } Ai = Bi \quad \text{y } Ad < Bd \end{cases}$$



donde a y b son los datos almacenados en los nodos raíces y, Ai, Ad, Bi y Bd son los subárboles izquierdos y derechos. **Pista:** reimplemente el mensaje equal(ArbolBinario unArbolBinario): boolean. Este método debería verificar que el árbol es estructuralmente igual y que tiene los mismos valores. Complejidad: media.

c) **lleno(): boolean.** Devuelve true si el árbol es lleno. Un árbol binario es lleno si tiene todas las hojas en el mismo nivel y además tiene todas las hojas posibles (es decir todos los nodos intermedios tienen dos hijos). **Pista:** recorra el árbol por niveles. Si encuentra una hoja, todos los demás nodos del mismo nivel deben ser hojas. Si no es así, o si encuentra un nodo que tenga un sólo hijo, el árbol no es lleno. Complejidad: media

d) **completo(): boolean.** Devuelve true si el árbol es completo. Un árbol binario de altura h es completo si es lleno hasta el nivel (h-1) y el nivel h se completa de izquierda a derecha. **Pista:** Calcule el nivel de la hoja de más a la izquierda (h). Verifique que el árbol es lleno hasta el nivel h-1 con el algoritmo del inciso e. Cuando recorre el nivel h-1, controle que si algún nodo deja de tener hijos, no exista ningún hijo para ningún nodo de más a la derecha. Complejidad: media

Ejercicio 3

Modelizar e implementar en Java la siguiente situación. Considere un árbol binario no vacío con dos tipos de nodos: nodos **MIN** y nodos **MAX**. Cada nodo tiene un valor entero asociado. Se puede definir el valor de un árbol de estas características de la siguiente manera. Si la raíz es un nodo **MIN**, entonces el valor del árbol es igual al mínimo valor entre: (i) El entero almacenado en la raíz. (ii) El valor correspondiente al subárbol izquierdo, si el mismo no es vacío. (iii) El valor correspondiente al subárbol derecho, si el mismo no es vacío. Si la raíz es un nodo **MAX**, entonces el valor del árbol es igual al máximo valor entre los valores nombrados anteriormente.
Pista: El dato que se guardará en el nodo debe ser un objeto que contenga un valor entero y un valor booleano. El valor false refiere a un nodo min y el valor true a un nodo max. Defina un método **evaluar(): int** en árbol binario. El método debe evaluar el subárbol izquierdo y el derecho, y debe tomar el valor del nodo. Luego, si es un nodo min, debe retornar el menor de los valores, caso contrario debe retornar el mayor.

Ejercicio 4

Implemente la clase **ArbolDeExpresion** como subclase de **ArbolBinario.** Incorpore un método de clase llamado **convertirPostfija(String exp): ArbolDeExpresion<T>**, que convierta una expresión aritmética en formato postfijo, en un árbol de expresión. El método recibirá la expresión postfija a convertir (como una cadena de caracteres sin blancos) y devolverá el árbol de expresión correspondiente. Esta expresión es sintácticamente válida, y está compuesta por números de un solo dígito o variables de una sola letra y los operadores binarios +, -, * y /.

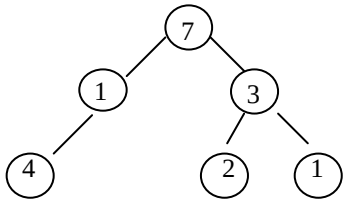
Defina una clase **TestConversion** con su método **main(String[] args)** el cual recibirá un String representando la expresión postfija. Luego utilice el método de clase **convertirPostfija(args) de la clase ArbolDeExpresión,**

para obtener el árbol de expresión correspondiente. Finalmente, realice un recorrido postorden imprimiendo cada elemento del árbol (al solo efecto de verificar si se obtuvo el String postfijo original).

Ejercicio 5

Se define el valor de trayectoria pesada de una hoja de un árbol binario como la suma del contenido de todos los nodos desde la raíz hasta la hoja multiplicado por el nivel en el que se encuentra. Implemente un método que, dado un árbol binario, devuelva el valor de la trayectoria pesada de **cada una de sus hojas**. Considere que el nivel de la raíz es 1.

Para el ejemplo siguiente: trayectoria pesada de la hoja 4 es: (4*3) + (1*2) + (7*1) = 21



Ejercicio 6 - El sistema numérico de Stern-Brocot

La **ACM** International Collegiate Programming Contest es una competencia internacional, en donde alumnos universitarios de carreras informáticas participan en equipos. Los problemas que deben resolver son muy variados, y en muchos de ellos, son necesarios los conceptos vistos en la materia. Este es un ejercicio del estilo de los que se deben resolver en las competencias. El enunciado no sufrió ningún cambio, fue extraído y traducido del libro *Programming Challenges, The Programming Contest Training Manual*, Skiena S., Revilla M., Springer, 2002.

El *árbol de Stern-Brocot* supone un bello método para construir el conjunto de todas las fracciones no negativas $\frac{m}{n}$, donde m y n son números primos entre sí. La idea es comenzar con dos fracciones $\left(\frac{0}{1}, \frac{1}{0}\right)$ y, a continuación, repetir la siguiente operación tantas veces como se desee:

- Insertar $\frac{m+m'}{n+n'}$ entre dos fracciones adyacentes $\frac{m}{n}$ y $\frac{m'}{n'}$

Por ejemplo, el primer paso da como resultado una nueva entrada entre $\frac{0}{1}$ y $\frac{1}{0}$.

$$\frac{0}{1}, \frac{1}{1}, \frac{1}{0}$$

y el siguiente da dos más:

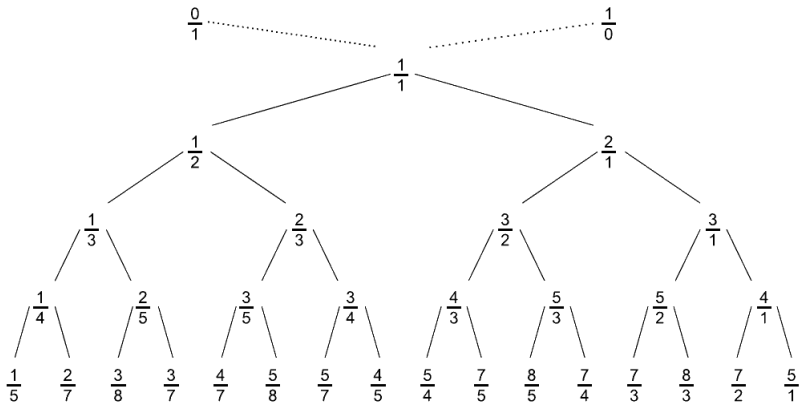
$$\frac{0}{1}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{1}{0}$$

La siguiente entrada cuatro más:

$$\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}, \frac{3}{2}, \frac{2}{1}, \frac{3}{1}, \frac{1}{0}$$

UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2013

La matriz completa es una estructura de árbol binario infinito, cuyos niveles superiores presentan este aspecto:



Esta construcción conserva el orden, por lo que no es posible que la misma fracción aparezca en dos lugares diferentes.

De hecho, podemos denominar al *árbol de Stern-Brocot* como un *sistema numérico* para la representación de números racionales, ya que cada fracción positiva reducida aparece una sola vez. Utilicemos las letras "L" y "R" para determinar el descenso por el árbol, hacia la izquierda o hacia la derecha, desde su punto más alto hasta una fracción en concreto; de esta forma, una cadena de L's y R's identifica de forma única cualquier lugar del mismo.

Por ejemplo, LRRL significa que descendemos hacia la izquierda desde $\frac{1}{1}$ hasta $\frac{1}{2}$, después hacia la derecha hasta $\frac{2}{3}$, después hacia la derecha hacia $\frac{3}{4}$, y finalmente, hacia la izquierda hasta $\frac{5}{7}$. Podemos considerar que LRRL es una representación de $\frac{5}{7}$.

Cualquier fracción positiva se puede representar de esta manera con una cadena única de L's y R's.

En realidad no todas las fracciones se pueden representar así. La fracción $\frac{1}{1}$ corresponde a una cadena vacía. La denominaremos I, ya que es una letra que se parece al número 1 y puede significar "identidad".

El objetivo del problema es representar en el *sistema numérico de Stern-Brocot* una fracción racional positiva.

Entrada

La entrada puede contener varios casos de prueba. Cada caso constará de una línea que contiene dos enteros positivos, m y n, donde m y n son números primos entre sí. La entrada terminará con un caso de prueba, en el que el valor tanto para m como para n sea 1, caso que no debe ser procesado.

Salida

Por cada caso de prueba de la entrada, mostrar una línea que contenga la representación de la fracción proporcionada en el *sistema numérico de Stern-Brocot*.

Ejemplo de entrada	**Ejemplo de salida**
5 7	LRRL
878 323	RRLRRLRLLLLRLRRR

UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2013

1 1

Anexo Ejercicios Parciales

Ejercicio 1

Una red binaria completa es una red que posee una topología de árbol binario completo (vea la Fig. 1 como ejemplo).

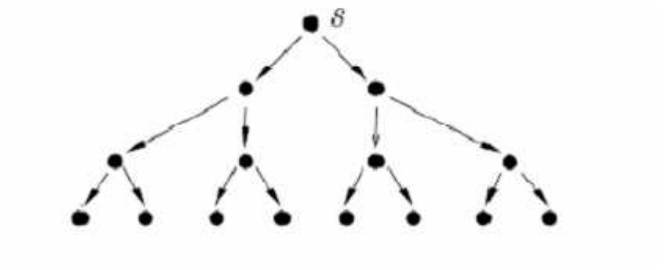


Figura 1. Ejemplo de una red binaria completa.

Los nodos que conforman una red binaria completa tienen la particularidad de que todos ellos conocen cual es su retardo de reenvío. El retardo de reenvío se define como el periodo comprendido entre que un nodo recibe un mensaje y lo reenvía a sus dos hijos.

Implemente un algoritmo que calcule el mayor retardo posible en el camino que realiza un mensaje desde la raíz hasta llegar a las hojas en una red binaria completa.

Ejercicio 2

Implemente el método **sumaElementosProfundidad (int p)** en la clase ArbolBinario que devuelva la suma de todos los nodos del arbol que se encuentren a la profundidad pasada como argumento.

Ejercicio 3

Dado un árbol binario "a" compuesto por personajes. Donde ningún personaje es dragón y princesa a la vez, y a su vez un personaje puede no ser ninguna de las dos cosas.

Se denominan *nodos accesibles* a aquellos nodos tales que a lo largo del camino del nodo raíz del árbol hasta el nodo (ambos inclusive) no se encuentra ningún dragón.

Implementar un método iterativo de costo lineal que encuentre una princesa accesible lo más cerca posible de la raíz del árbol dado "a", suponiendo que algún nodo accesible de "a" contiene una princesa.

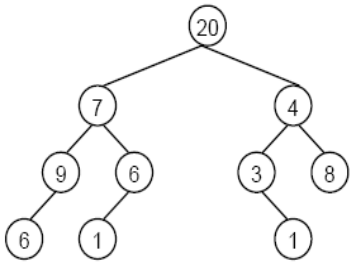
Juego
- ArbolBinario <Personaje> personajes
+princesaMasCercana (): boolean

Personaje
- String nombre;
+esDragon (): boolean +esPrincesa (): boolean

UNLP. Facultad de Informática.
Algoritmos y Estructuras de Datos
Cursada 2013

Ejercicio 4

Implemente un método que realice un recorrido que llamaremos **recorrido_guiado**, que permita recorrer un árbol de forma que, en cada iteración, se selecciona el nodo mas pequeño de entre todos los disponibles en ese momento, independientemente de en que rama se encuentre. Se entiende por nodo disponible aquel nodo cuyo padre ya ha sido procesado (excluyendo el nodo raíz).
Ejemplo de este recorrido:



Para este árbol, el método **recorrido_guiado** debe imprimir en consola: 20, 4, 3, 1, 7, 6, 1, 8, 9, 6

Ejercicio 5

Implemente la operación **trayectoriaPesada(ab: Arbol Binario) : Lista** // Retorna el valor de la trayectoria pesada de cada una de las hojas del árbol binario ab

Se define el valor de la trayectoria pesada de una hoja de un árbol binario como la suma del contenido de todos los nodos desde la raíz a la hoja multiplicada por el nivel en el que se encuentra.

Ejemplo:

Trayectoria Pesada hoja 4 es 4*2 + 1*1 + 7*0 = 9

Trayectoria Pesada hoja 2 es 2*2 + 3*1 + 7*0 = 7

Trayectoria Pesada hoja 1 es 1*2 + 3*1 + 7*0 = 5