Práctica 3: Árboles Binarios y Generales

Programación III - UNLP

Cursada 2013

Todos los ejercicios de la presente práctica deberán hacerse en `Java` dentro de un mismo proyecto llamado "`Prog3_2013`". Se recomienda realizar cada ejercicio en un paquete diferente, y se recomienda el uso de `Eclipse`.

Además, se recomienda realizar todas las practicas de la presente materia dentro de un mismo proyecto de `Eclipse`, a fin de poder reutilizar código escrito entre diferentes ejercicios y prácticas.

Objetivos

- Representar árboles binarios e implementar las operaciones de la abstracción.
- Realizar distintos tipos de recorridos sobre árboles binarios.
- Describir soluciones utilizando árboles binarios.
- Modelar un árbol n-ario.
- Realizar recorridos e implementar aplicaciones varias sobre el árbol.

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

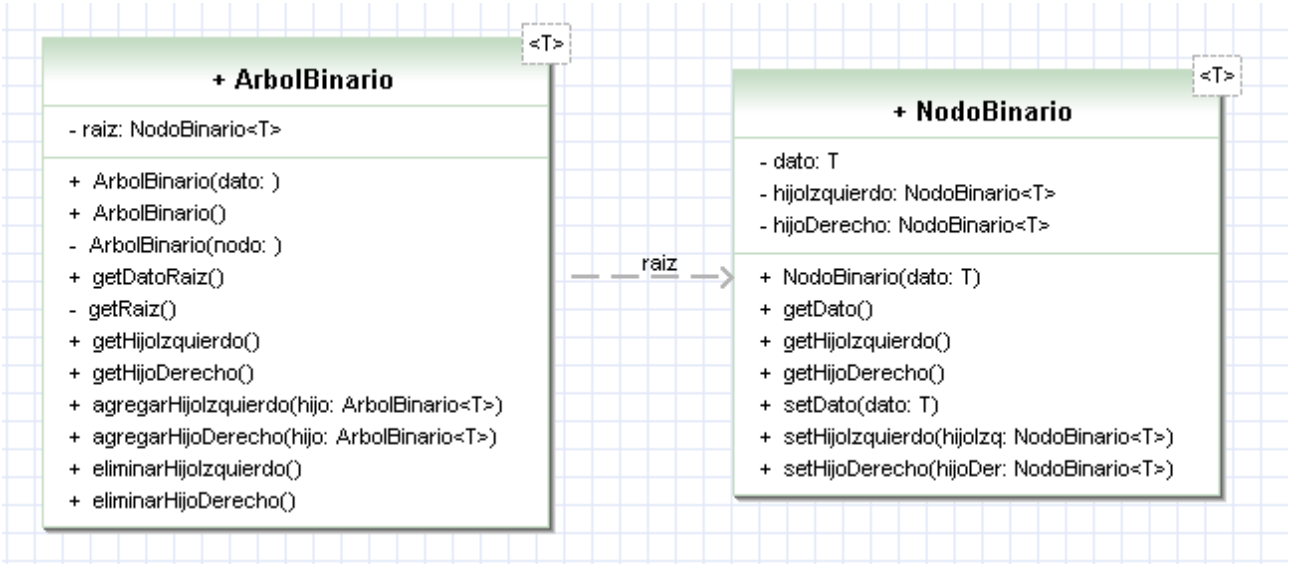

Figura 1: Ejercicio 1. Árbol Binario

a) El constructor `ArbolBinario()` inicializa un árbol binario vacío, es decir, la raíz en `null`.

b) 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.

c) 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).

d) El método `getRaiz():NodoBinario<T>`, retorna el nodo ubicado en la raíz del árbol. (Notar que **no es** un método público).

e) El método `getDatoRaiz():T`, retorna el dato almacenado en el `NodoBinario` raíz del árbol.

f) Los métodos

1) `getHijoIzquierdo():ArbolBinario<T>`
2) `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.

g) Los métodos

1) `agregarHijoIzquierdo(ArbolBinario<T>unHijo)`
2) `agregarHijoDerecho(ArbolBinario<T>unHijo)`

agregan 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()`.

h) El método `eliminarHijoIzquierdo()` y `eliminarHijoDerecho()`, eliminan el hijo correspondiente `NodoBinario` raíz del árbol receptor.

A partir de la especificación anterior, implemente en Java las clases `ArbolBinario` y `NodoBinario`.

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(): Lista<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.

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

$$A < B = \begin{cases} a_0 < b_0 \\ a_0 = b_0 \ \wedge \ A_i < B_i \\ a_0 = b_0 \ \wedge \ A_i = B_i \ \wedge \ A_d = B_d \end{cases}$$

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

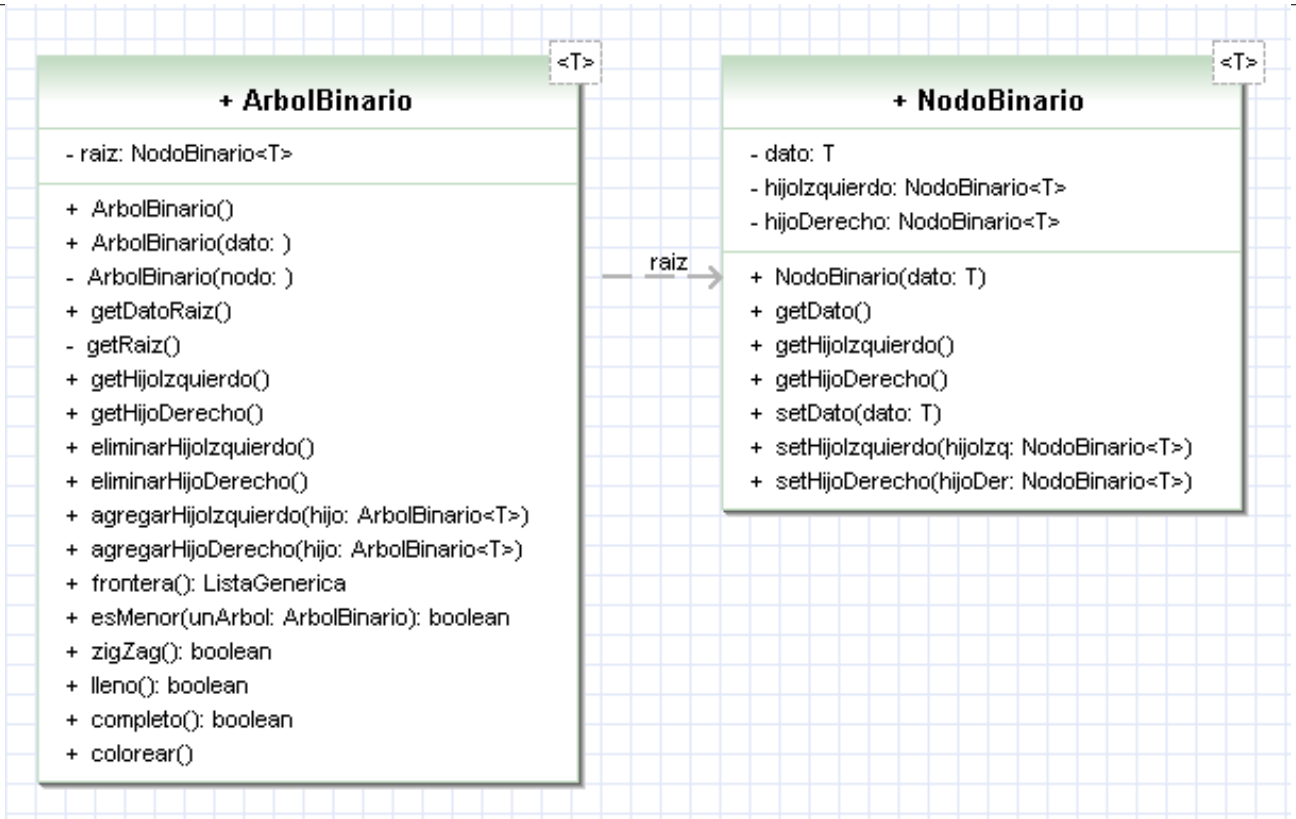

Figura 2: Ejercicio 2

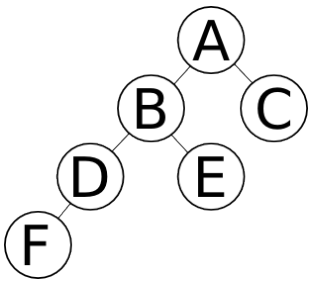

Figura 3: Ejercicio 2-A. `frontera()`, deberá devolver "`FEC`".

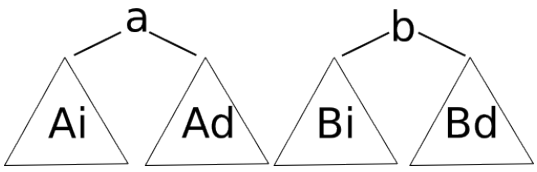

Figura 4: Ejercicio 2-B, Árboles A y B.

c) `zigZag(): boolean`. Devuelve `true` si el árbol es degenerado con direcciones alternadas, esto es, si en lugar de ser una lista donde todos los hijos están en el lado izquierdo o derecho, se van alternando. **Complejidad**: inicial.

d) `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

e) `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 anterior. 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

f) `colorear()`. Un árbol binario coloreado es un árbol binario lleno cuyos nodos tienen uno de dos colores posibles: negro o blanco. El color para la raíz del árbol es negro. Y para cada nivel los colores de los nodos se intercalan, como así también al comenzar cada nivel. Por ejemplo un árbol coloreado de altura 3 deberá ser:

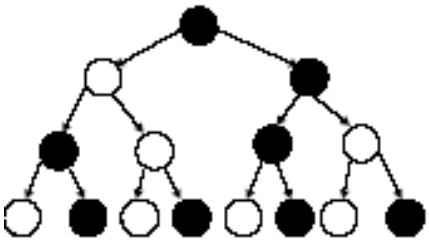

Figura 5: Ejercicio 2. `colorear()`

Pista: coloree la raíz y encole los hijos para realizar un recorrido por niveles. Encole una marca de fin de nivel. Inicialice una variable con el color a usar. Mientras la cola no se vacía itere. Vaya coloreando los nodos alternando el color y encolando los hijos. Cuando encuentra la marca de fin de nivel, no alterne el color. **Complejidad**: media

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. Ya sea si la raíz es un nodo **min** o **max**, 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.

   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.

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).

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 de la figura: trayectoria pesada de la hoja 4 es: $(4 * 3) + (1 * 2) + (7 * 1) = 21$.

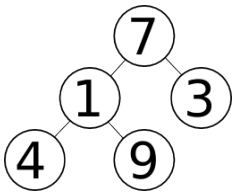

Figura 6: Ejercicio 5

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 $\left(\frac{0}{1}, \frac{1}{0}\right)$:

$$\left(\frac{0}{1}, \frac{1}{1}, \frac{1}{0}\right)$$

   y el siguiente da dos más:

$$\left(\frac{0}{1}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{1}{0}\right)$$

   La siguiente entrada cuatro más:

$$\left(\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}, \frac{3}{2}, \frac{2}{1}, \frac{3}{1}, \frac{1}{0}\right)$$

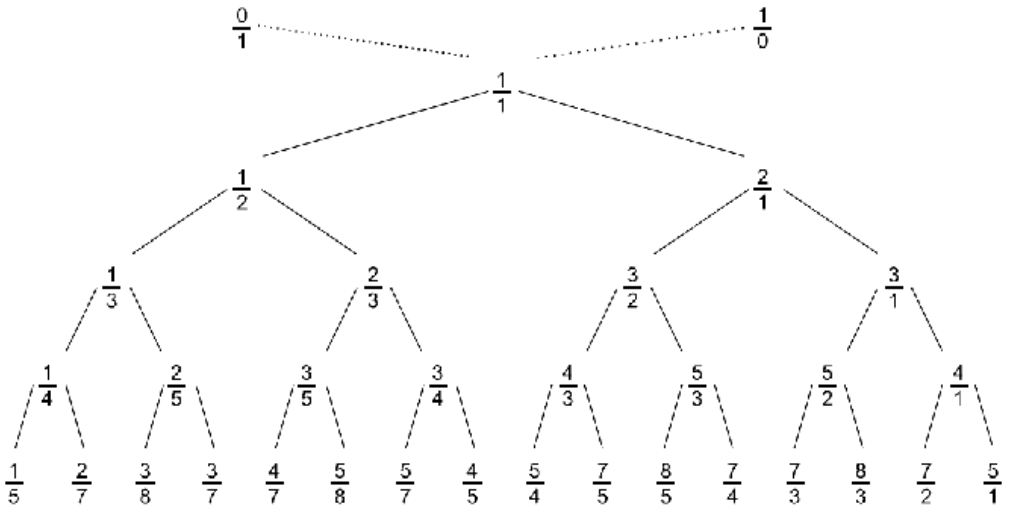

Figura 7: Ejercicio 6. Stern-Brocot

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 $1/1$ hasta $1/2$, después hacia la derecha hasta $2/3$, después hacia la derecha hacia $3/4$, y finalmente, hacia la izquierda hasta $5/7$. Podemos considerar que "`LRRL`" es una representación de $5/7$. 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 $1/1$ 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

```
5 7
```

```
878 323
1   1
```

Ejemplo de salida

```
LRRL
RRLRRLRLLLLRLRRR
```

7. Considere la siguiente especificación de la clase `ArbolGeneral` (con la representación de **Lista de Hijos**)

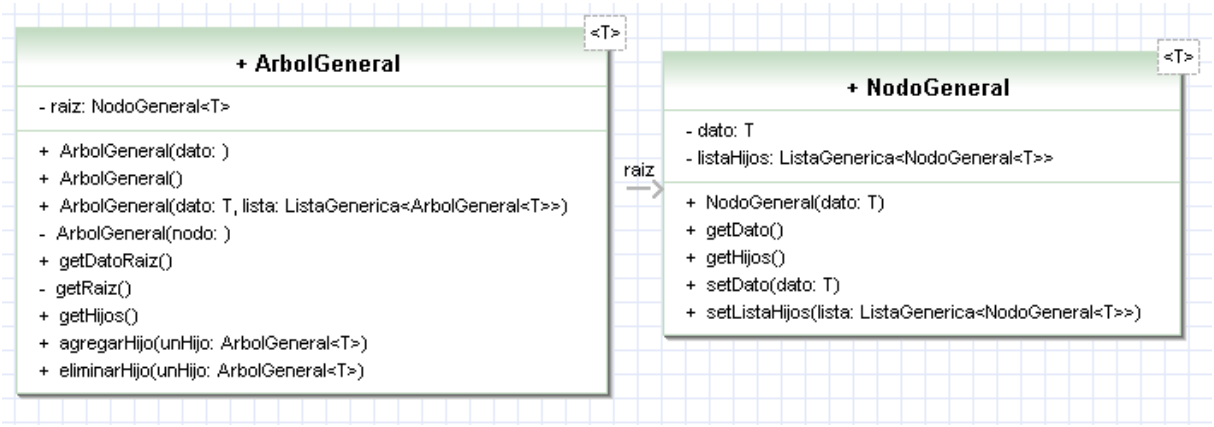

Figura 8: Ejercicio 7. Árbol General

Notas:

- Este ejercicio tiene como fin implementar las operaciones básicas utilizando la lista definida en la práctica anterior.
- La clase Lista es la utilizada en la práctica anterior. Tenga en cuenta que el árbol general también puede implementarse con la representación de Hijo más izquierdo – Hermano derecho, sin embargo en la práctica sólo vamos a utilizar la representación lista de hijos.

Implemente en Java (pase por máquina) las clases `ArbolGeneral` y `NodoGeneral` de acuerdo a la especificación siguiente:

a) El constructor `ArbolGeneral()` inicializa un árbol vacío, es decir, la raíz en `null`.

b) El constructor `ArbolGeneral(T dato)` inicializa un árbol que tiene como raíz un nodo general. Este nodo tiene el dato pasado como parámetro y una lista vacía.

c) El constructor `ArbolGeneral(T dato, Lista<ArbolGeneral <T>>hijos)` inicializa un árbol que tiene como raíz un nodo general. Este nodo tiene el dato pasado como parámetro y tiene como hijos una copia de la lista pasada como parámetro. Tenga presente que la Lista pasada como parámetro es una lista de árboles generales, mientras que la lista que se debe guardar en el nodo general es una lista de nodos generales. Por lo cuál, de la lista de árboles generales debe extraer la raíz (un `NodoGeneral`) y guardar solamente este objeto.

d) El constructor `ArbolGeneral(NodoGeneral<T>nodo)` inicializa un árbol donde el nodo pasado como parámetro es la raíz. Notar que este constructor **no es público**.

e) El método `getRaiz():NodoGeneral<T>` retorna el nodo ubicado en la raíz del árbol. Notar que este constructor **no es público**.

f) El método `getDatoRaiz():T` retorna el dato almacenado en la raíz del árbol (`NodoGeneral`).

g) El método `getHijos():Lista<ArbolGeneral<T>>`, retorna la lista de hijos de la raíz del árbol. Tenga presente que la lista almacenada en la raíz es una lista de nodos generales, mientras que debe devolver una lista de árboles generales. Para ello, por cada hijo, debe crear un árbol general que tenga como raíz el `NodoGeneral` hijo.

h) El método `agregarHijo(ArbolGeneral<T>unHijo)` agrega `unHijo` a la lista de hijos del árbol. Es decir, se le agrega a la lista de hijos de la raíz del objeto receptor del mensaje el `NodoGeneral` raíz de `unHijo`.

i) El método `eliminarHijo(ArbolGeneral<T>unHijo)` elimina `unHijo` del árbol. Con la misma salvedad indicada en el método anterior.

8. Agregue a la clase `ArbolGeneral` siguientes métodos:

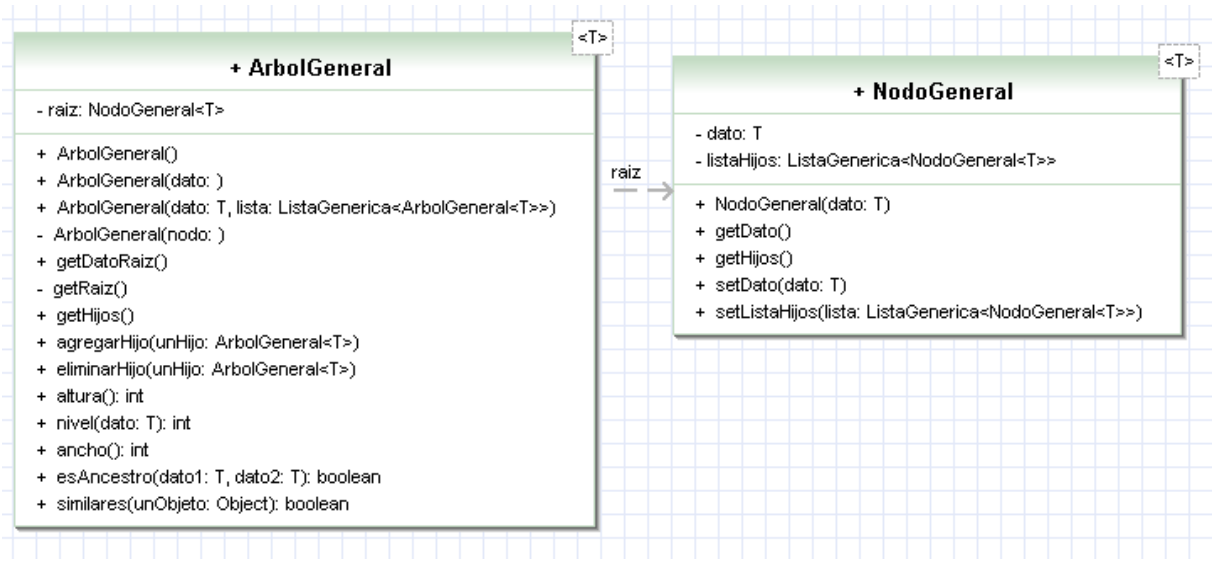

Figura 9: Ejercicio 8

a) `altura(): int` devuelve la altura del árbol, es decir, la longitud del camino más largo desde el nodo raíz hasta una hoja. **Pista**: el mensaje altura debe chequear si el árbol es una sola hoja o no. Si el árbol es una sola hoja, se devuelve 0. Si no, se utiliza el método `getHijos()` para obtener la lista de hijos (recuerde que devuelve una lista de árboles hijos). Luego, debe iterar por cada uno de los hijos, guardando la máxima altura. A este valor se le debe sumar 1 y retornarlo.

b) `nivel(T dato): int` devuelve la profundidad o nivel del dato en el árbol. El nivel de un nodo es la longitud del único camino de la raíz al nodo. **Pista**: si el nodo raíz posee el mismo dato que pasado como parámetro, se retorna 0. En caso contrario, se debe buscar en cuales de los subárboles hijos se encuentra el dato (implemente el mensaje `include (T dato)` en la clase `ArbolGeneral`) y se debe retornar 1 más el nivel que arroje enviar el mensaje `nivel()` al subárbol que incluye el dato.

c) `ancho(): int` la amplitud (ancho) de un árbol se define como la cantidad de nodos que se encuentran en el nivel que posee la mayor cantidad de nodos. **Pista**: realice un recorrido por niveles. Encole inicialmente la raíz del árbol y luego una marca `null` (o el número de

nivel) para indicar el fin de nivel. Mientras la cola no se vacía, itere. En cada iteración extraiga el tope de la cola, y con la operación `getHijos()` encole los mismos. Cuando encuentra la marca de fin de nivel cuente si los elementos del nivel es mayor a la máxima cantidad que poseía.

d) `esAncestro (T dato1, T dato2): boolean` determina si `dato1` es ancestro de `dato2`, es decir si hay un camino de `dato1` hasta `dato2`. Considere que no se repite el contenido de los nodos. **Pista**: Implemente un mensaje `subárbol (T dato)` que retorna el subárbol que tiene como raíz el nodo general que contiene el dato pasado como parámetro. Tenga presente que el método ancestro debe devolver un valor booleano, y que el mensaje subárbol que se recomienda implementar devuelve árboles.

e) `similares(ArbolGeneral<T>otroArbol): boolean` determina si dos árboles son o no similares. Dos árboles son similares si son estructuralmente iguales (aunque no coincidan los valores). **Pista**: realice un recorrido por niveles en simultáneo en el objeto receptor y en otroArbol. Si la cantidad de hijos de las raíces de los subárboles no es igual, los árboles ya son distintos. Si son iguales, compare cada para de árboles hijos.

Nota: tenga presente que no importaría qué representación interna utilizaría (lista de hijos o hijo más izquierdo – hermano derecho), estas operaciones deberían estar implementadas a partir de las operaciones especificadas en el ejercicio 1. El fin de este ejercicio es implementar operaciones utilizando las operaciones básicas definidas anteriormente.

9. Modelizar e implementar en Java la siguiente situación. En una empresa los empleados están categorizados con un número en el rango de 1 a 10, siendo el 1 el presidente, el 2 el vicepresidente, 3 los gerentes y así siguiendo. Los mismos también poseen una antigüedad. Esta información está dispuesta en un árbol general. Para este ejercicio realice métodos que al menos resuelvan los siguientes problemas:

   a) devolver la cantidad de empleados por categoría.

   b) determinar la categoría que cuenta con la mayor cantidad de empleados.

   c) determinar la cantidad total de empleados.

   d) sea la situación en donde el presidente deja su función, reemplazarlo por la persona más antigua de sus subordinados, quién a su vez es reemplazada de la misma forma.

   Pista:

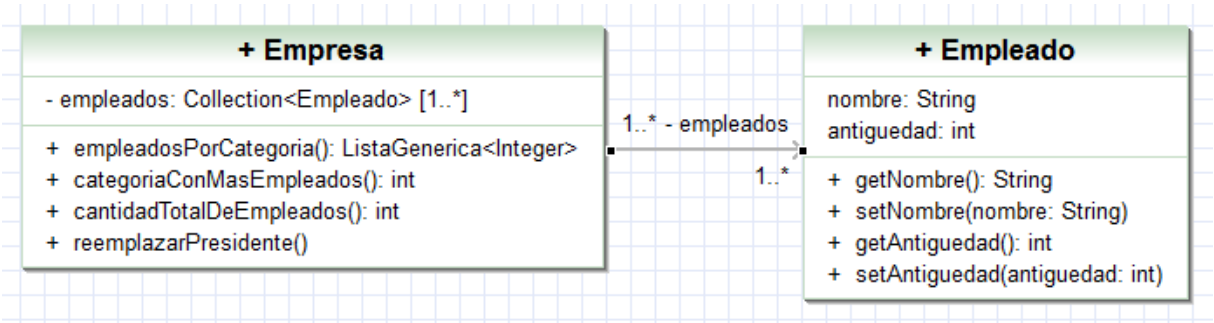

Figura 10: Ejercicio 9

Debe modificar el árbol para que el nodo represente a cada empleado. Luego, los hijos del nodo representan la relación "es subordinado de"

a) Debe implementar un recorrido por niveles como lo hizo en 2.c. Debe totalizar para cada uno de los niveles.

b) A partir del punto anterior, se debe quedar con la mayor cantidad.

c) A partir de a, debe realizar la suma

d) Debe tomar los hijos de la raíz, y buscar el de mayor antigüedad de los hijos. Sin modificar la estructura, pase el mayor de los hijos a la raíz, y se envía el mensaje al hijo promovido. Cuando el hijo promovido no tenga hijos, se lo debe eliminar.

10. Implemente una clase en Java llamada `Imagen` que permita crear imágenes en blanco y negro que utilice como representación interna una matriz. Una forma de comprimir una imagen es transformarla a un árbol 4-ario. El algoritmo es el siguiente. Si toda la matriz tiene un mismo color, se debe definir un nodo con ese color. En caso contrario, se divide la matriz en cuatro partes, se define un nodo con 4 hijos, y cada hijo es la conversión de cada una de las partes de la matriz. Realice un método de instancia llamado `imagenComprimida()` que devuelva la representación de la imagen en su árbol correspondiente.

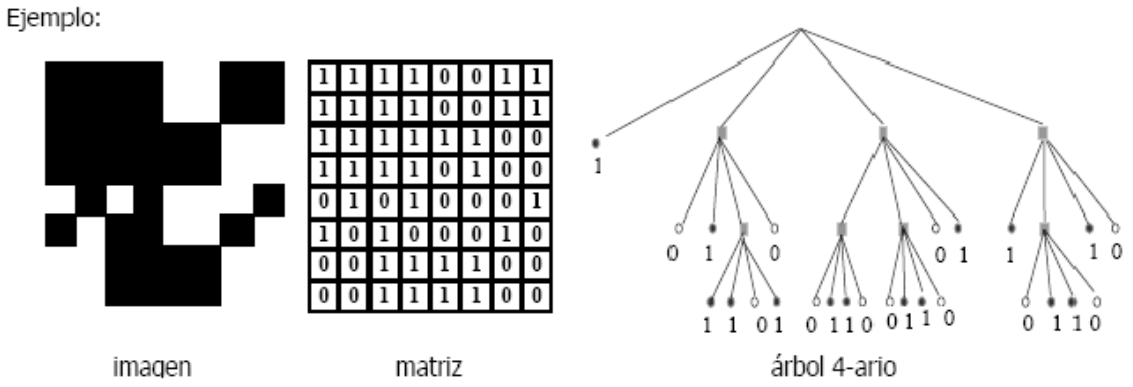

Figura 11: Ejercicio 10

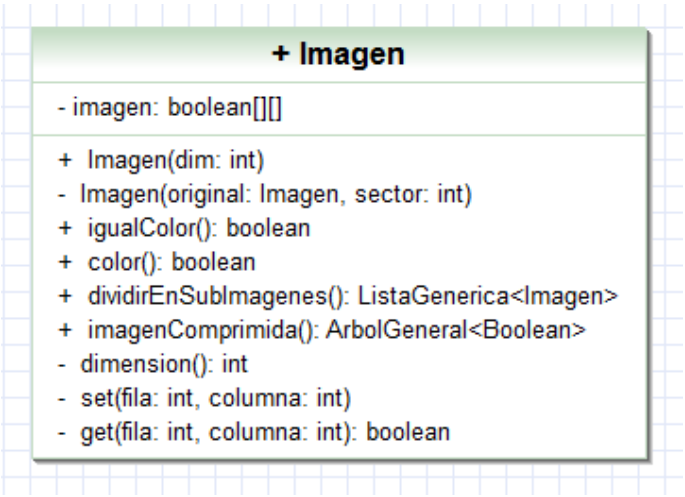

Figura 12: Ejercicio 10

Teniendo en cuenta que el color blanco lo representamos con el valor falso y el negro con verdadero, implemente los métodos `igualColor()`, `color()` y `dividirEnSubimagenes()` en la

clase `Imagen`. El método `dividirEnSubimagenes()` devuelve una Lista con 4 `Imágenes`. Para comprimir la imagen, debe verificar si la misma posee igual color, en ese caso debe crear y devolver un `ArbolGeneral` con un único nodo. En cambio, si la imagen no posee el mismo color, debe `dividirEnSubImagenes` y enviar el mensaje `imagenComprimida()` a cada una.

11. Modelizar e implementar en Java una tabla de Contenidos de un libro. Un Libro está compuesto por capítulos y apéndices. Los mismos además del nombre tienen un tema y pagina inicial. A su vez cada uno de estos puede estar compuesto por secciones, que cuentan con la misma información: nombre, tema y página inicial; y cada sección podría contener más secciones. La tabla de contenidos comienza con el contenido destacado que es el nombre del libro. Para este ejercicio codifique métodos para resolver los siguientes problemas:

    a) Devolver todos los temas de las secciones que conforman un capitulo del libro.

    b) Devolver para cada capítulo que conforma el libro su tema, pagina inicial y pagina final.

    c) Retornar el/los capítulos que están compuestos por el mayor numero de anidamientos de secciones.

    d) Tetornar el/los temas que están compuestos por el mayor numero de subtemas.

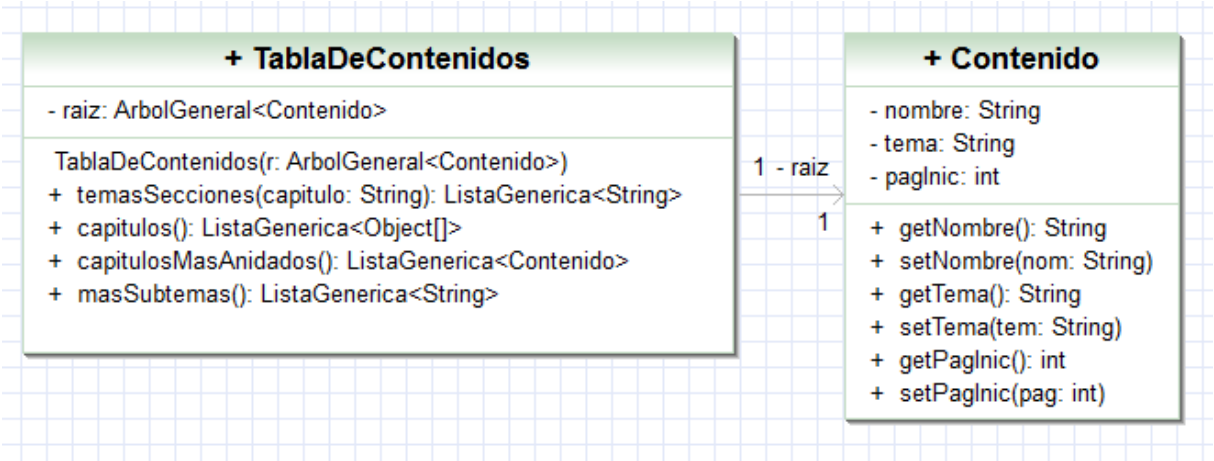

Figura 13: Ejercicio 11

Ejemplo:

a) Contenido 0. Contenido. Estructuras de datos.
    1) Contenido 1. Capítulo 1. “Listas”. Página 1.
        a′ Contenido 5. L1. “Estructuras estáticas y dinámicas”. Página 1.
        b′ Contenido 6. L2. “Listas con arrays”. Página 14.
            i- Contenido 8. L2.1. “Pilas”. Página 14.
            ii- Contenido 9. L2.2. “Colas”. Página 17.
            iii- Contenido 10. L2.3. “Comparación de representaciones”. Página 21.
            iv- Contenido 11. L2.4. “Sugerencias de codificación”. Página 24.
        c′ Contenido 7. L3. “Listas enlazadas”. Página 26.
    2) Contenido 2. Capítulo 2. “Árboles”. Página 32.
    3) Contenido 3. Capítulo 3. “Grafos”. Página 55.
    4) Contenido 4. Apéndice A. “Métodos matemáticos”. Página 74.

a) Debe buscar el capítulo. Una vez obtenido el mismo, debe recorrer su lista de hijos, que contiene las secciones del mismo y devolver los temas de las secciones en una la lista.
En el ejemplo debería devolver: "*Estructuras estáticas y dinámicas*", "*Listas con Arrays*", "*Listas Enlazadas*", si se recibe como parámetro el capítulo "*Capítulo1*".

b) Debe tomar los hijos de la raíz, que son los capítulos y apéndices del libro y de cada capítulo obtener el nombre, tema, página inicial y página final. Tener en cuenta que la página final de un capítulo es la anterior a la página inicial del siguiente capítulo. Se debe devolver la información de los capítulos únicamente.
En el ejemplo debería devolver: "*Capítulo 1, Listas, 1, 31*", "*Capítulo 2, Árboles, 32, 54*", "*Capítulo3, Grafos, 55, 73*"

c) Debe calcular la altura de cada capítulo utilizando el método implementado en 2ª y quedarse con todos los capítulos que igualen a la mayor altura.
En el ejemplo el árbol general cuyo nodo raíz es capitulo 1 tiene una altura de 2, mientras que capitulo 2 y capitulo 3 tienen ambos una altura 0, por lo tanto la lista contendrá en este caso solo "*Capitulo 1*"

d) Se debe recorrer toda la tabla de contenidos en profundidad o por niveles. Recorrer cada uno de los temas contabilizando para cada uno la cantidad de hijos que tiene, y manteniendo solo los que igualan al mayor.
En el ejemplo "*Estructura de Datos*" y "*Listas con Arrays*" tienen 4 subtemas; "*Listas*" tiene 3 y los restantes temas no están compuestos por subtemas; con lo cuál la lista a retornar tendría "*Estructura de Datos, Listas con Arrays*"

12. Implementacion de un TRIE (**TRIEs == re-trie-val trees**)

Definicion de un TRIE.

- Es una estructura de datos que permite representar conjuntos de cadenas de caracteres.
- Cada nodo de T, excepto la raíz, está etiquetado con un símbolo del alfabeto.
- Los hijos de un nodo interno de T están ordenados según el ordenamiento en el alfabeto.
- Cada hoja marca el final de una cadena.

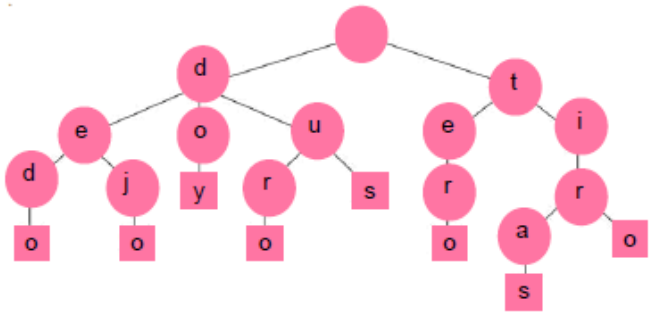

Figura 14: Ejercicio 12. TRIE.

Una aplicación frecuente de los TRIE es el almacenamiento de diccionarios, como los que se encuentran en los teléfonos móviles.

a) Indique de acuerdo a las estructuras de datos ya vistas cual correspondería a una posible implementacion de TRIE.

b) Implemente un TRIE con la siguiente operación:
`public StringBuffer palabrasIngresadas(String supuestaPalabra)`.