1) Realizar un método que dado un árbol de expresiones cuyos operandos puede ser números reales o variables y un **Map** que contiene los valores de las variables, retorna una lista con la expresión postfija del mismo.

/\*

\* Retorna una lista con la expresión postfija de un árbol de expresiones

\* Si el operando es una variable busca su valor en el Map v

\* Si la variable no existe lanza la excepción: ArithmeticException indicando

\* el nombre de la variable que no existe.

\*/

**public** **static** List<String> postfixExpression(LinkedBinaryTree<String> t,

Map<String, Double> v) {

Realizar una aplicación que pruebe el método implementado con el árbol de expresiones que representa la siguiente expresión aritmética:

(B+C)∗(A−1) Donde: A = 2.0, B = 5.0 y C = 8.0

Elimine una de las variables del **Map** y atrape la excepción lanzada.

2) a) Agregar a la clase **LinkedBinaryTree** el siguiente método:

/\*\*

\* Retorna un Map donde la clave es la profundidad y el valor una lista con

\* las posiciones de todos los nodos que tienen esa profundidad.

\*

\* **@return** Map clave: profundidad, valor: lista de nodos con esa profundidad

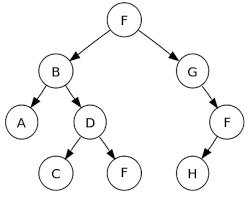
\*

\*/

**public** TreeMap<Integer, List<E>> mapDepth()

b) Realizar un programa de prueba que cargue un árbol con letras. Llamar el método implementado y mostrar el resultado retornado.

Por ejemplo, dado el árbol:



El resultado mostrado por consola sería similar al siguiente:

Profundidad: 0 Letra: [F]

Profundidad: 1 Letra: [B, G]

Profundidad: 2 Letra: [A, D, F]

Profundidad: 3 Letra: [C, F, H]

3) Agregar a la clase **LinkedBinaryTree** el siguiente método:

/\*\*

\* Reemplaza todas las ocurrencias de los elementos del árbol que está en

\* el Map

\*

\* **@param** replace

\* Map que contiene los elementos a reemplazar

\*

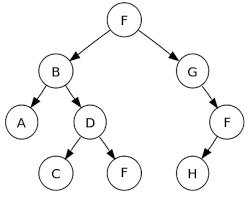
\* **@return** cantidad de elementos reemplazos

\*/

**public** **int** replaceAll(Map<E, E> replace)

Realizar un programa de prueba que cargue un árbol con letras y las reemplace utilizando el método implementado. Mostrar sus resultados antes de reemplazarlas y después de reemplazarlas recorriendo el árbol por niveles.

Por ejemplo, dado el árbol:



Si se pasa un map donde una clave es la letra “F” con el valor de reemplazo “Z” y otra clave es la letra “C” con el valor de reemplazo “W” el resultado por la consola sería similar al siguiente:

FBGADFCFH

Cantidad de reemplazos: 4

ZBGADZWZH

4) Agregar a la clase **LinkedBinaryTree.java** el siguiente método:

/\*\*

\* Lista de los nodos que representan un camino.

\*

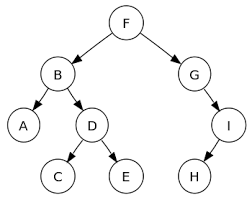
\* **@return** Lista con las posiciones de los nodos que representan un camino desde la posición

\* from a la posición to.

\*/

**public** List<Position<E>> path(Position<E> from, Position<E> to)

Por ejemplo, dado el siguiente árbol:



El método retorna una lista con los las posiciones de los siguientes elementos:

a) path (A, I) = [ A, B, F, G, I ]

b) path (E, A) = [ E, D, B, A ]

c) path (B, B) = [ B ]

d) path (D, B) = [ D, B ]