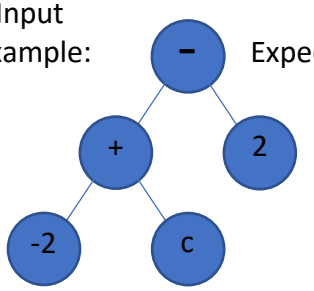
Info1105 Assignment2 Report Canhong Cai SID:460312484

Description and justification of the tests

- testTree2prefix

Description: the input of this method is a correct structure of tree for Arithmetic Expression.

Input Example:



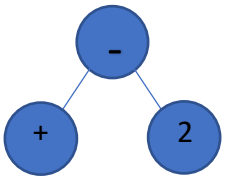
Expected output: - + -2 c 2 which is a String

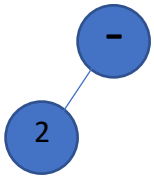
Purpose of the test: testing the function can return the correct prefix arithemetic expression when the tree contains negative integer nodes, letter nodes and non-negative nodes.

- testTree2prefixException

Description: the input of this method is an invalid structure of tree for Arithmetic Expression.

Input Example: Expected output: throw an IllegalArgumentException



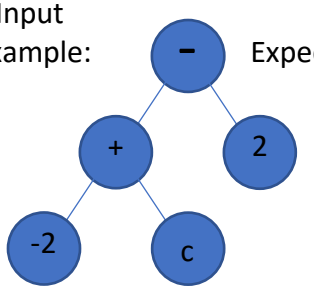


Purpose of the test: testing the function can detect the Illegal Arithmetic Expression in tree structure, which are two examples above, and throw an IllegalArgumentException to remind users.

- testTree2infix

Description: the input of the method is a correct structure of tree for Arithmetic Expression.

Input Example:



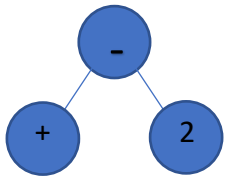
Expected output: ((-2+c)-2) which is a String

Purpose of the test: testing the function can return the correct infix arithemetic expression when the tree contains negative integer nodes, letter nodes and non-negative nodes.

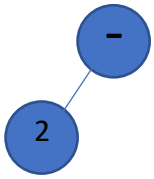
- testTree2infixException

Description: the input of this method is an invalid structure of tree for Arithmetic Expression.

Input Example:



Expected output: throw an IllegalArgumentException

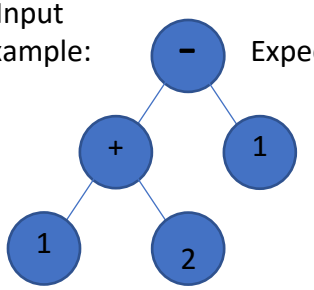


Purpose of the test: testing the function can detect the Illegal Arithmetic Expression in tree structure, which are two examples above, and throw an IllegalArgumentException to remind users.

- testSimplify1

Description: the input of the method is a simple LinkedBinaryTree, and the output is the simplify tree by following the arithmetic expression.

Input Example:



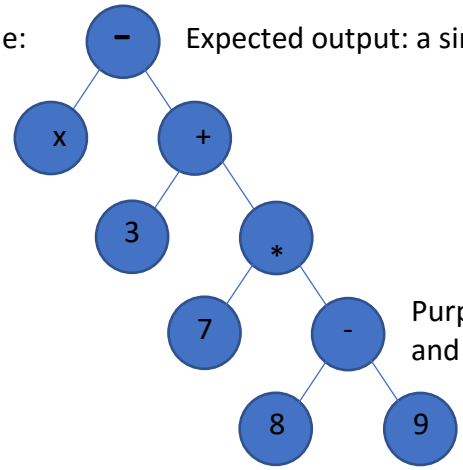
Expected output: a simlify LinkedBinaryTree

Purpose of the test: testing a simple tree to make sure the method work in basic situation.

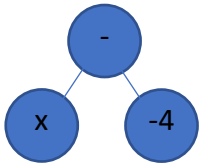
- testSimplify2

Description: the input of the method is a LinkedBinaryTree, and the output is the simplify tree by following the arithmetic expression.

Input Example:



Expected output: a simlify LinkedBinaryTree



Purpose: testing every operators can work well and tesing the rule that simplify cannot simplify a letter.

- testSimplify3 ---- testSimplify 8

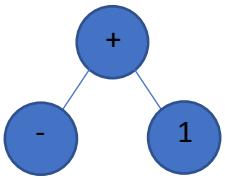
Decrption: the input of each method is a LinkedBinaryTree, and the output is the simplify tree by following the arithmetic expression.

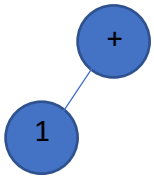
Purpose: testing the method can handle more complicated trees such as tree with negative integer, positive integer, letter, three operators.

- testSimplifyException

Description: the inputs of the method are invalid LinkedBinaryTree(tree==null, and two trees below)

Input Example: Expected output: throw an IllegalArgumentException





Purpose of the test: testing the function can detect the Illegal Arithmetic Expression in tree structure,null input tree, and throw an IllegalArgumentException.

- testSimplifyFancy1---testSimplifyFancy7

Description: * 1 x == x i.e. (1*x)==x input: left hand side(x can be letter or tree)
* x 1 == x (x*1)==x output: right hand side
* 0 x == 0 (0*x)==0 it is a general representation of my test input
* x 0 == 0 (x*0)==0 in order to minimize the length of the report
+ 0 x == x (0+x)==x
+ x 0 == 0 (x+0)==x
- x 0 == x (x-0)==x
- x x == 0 (x-x)==0
- 0 x == - 0 x cannot simplify

purpose: check the method can work perfectly follow by the simplifyFancy rules

- testSimplifyFancy8-9

Description: input: complicated tree containing letter, non-negative and negative integer

Output: really simple tree

Purpose: make sure that the method will work when simplify the tree with several different Fancy rules and with various kinds of nodes.

- testSubstitute

Description: input: linked binary tree with letters and Integer

Output: new tree with choosing letter substitute by Integer value

Purpose: testing the method can substitute choosing letter with non-negative or negative Integer

- test SubstituteException1—4

Description: input: an invalid tree expression, null tree, an invalid variable (e.g. operators, null, Integer), and a value

Output: throw IllegalArgumentException

Purpose: testing the method can correctly handle all the invalid input

- testSubtituteMap

Description: input: linked binary tree with several different letter and Integer, a HeapMap
Output: a substitute tree with every selected letters are substituted by given corresponding values
Purpose: testing method can substitute all variables correctly following the map

- testSubstituteMapException

Description: input: an invalid tree expression or null map
Output: throw IllegalArgumentException
Purpose: testing method can handle this kind of problem and throw exception correctly

- testisArithmeticException

Description: input: invalid Linked BinaryTree expression
Output: if the expression is right return true, else return false
Purpose: testing the method can return the right Boolean value for valid and invalid tree

- testMixed

  Description: input: linked binary tree and map with variable and value
  Output: simplify and substitute tree
  Purpose: testing methods can use at the same time without any errors

Analysis run time cost

- public static String tree2prefix(LinkedBinaryTree<String> tree) throws IllegalArgumentException

Algorithm: I implemented the method recursively. The base case of the recursion is that If the node is a leaf, the recursion should stop and return the element in that node, since no more nodes exist after the leaf. I used a private helper method with one more parameter called Position<String> root to update the node that we choose for each recursion than the original function.

```
tree2prefix(LinkedBinaryTree<String> tree) throws IllegalArgumentException
    if tree is null
        throw IllegalArugmentException
  return tree2prefix(tree, tree.root())
tree2prefix(LinkedBinaryTree<String> tree, Position<String> root) throws
IllegalArgumentException
Input: linked binary tree, root          Output: Arithmetic expression prefix in String
If tree.isExternal(root) is False
    If node only has one child  (tree.numChildren(root)=1)
        Throw IllegalArgumentException
  Return root.getelment()+" "+ tree2prefix(tree, tree.left(root))+" "+tree2prefix(tree,
  tree.right(root))
if tree.isExternal(root) is True
    if root is operator(+,-,*)
        Throw IllegalArgumentException
```

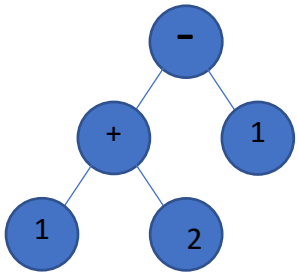
Return the element of the leaf

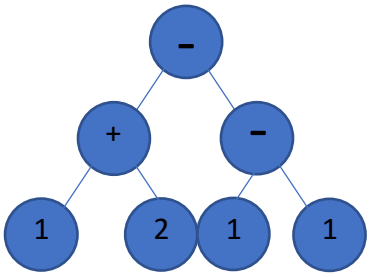
Big O worst case runtime and justification: →O(n)

root.getelment()+" "+ tree2prefix(tree, tree.left(root))+" "+tree2prefix(tree, tree.right(root))

O(1)+T(n-1)+O(1) (first graph) O(1)+T(n/2)+(T/2)(second graph)

Since Other lines are O(1)





T(n-1)+O(1)=O(n) 2T(n/2)+O(1)=O(n)

- public static String tree2infix(LinkedBinaryTree<String> tree) throws IllegalArgumentException

Algorithm: The idea of the method is basically same as the previous method tree2prefix, the difference is that the output is in infix expression.

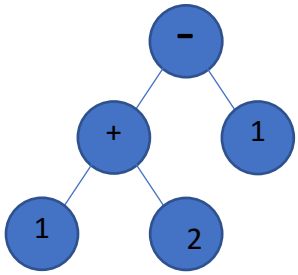
tree2infix(LinkedBinaryTree<String> tree) throws IllegalArgumentException
if tree is null
throw IllegalArugmentException
return tree2infix(tree, tree.root())
tree2infix(LinkedBinaryTree<String> tree, Position<String> root) throws IllegalArgumentException
Input: linked binary tree, current node Output: Arithmetic expression infix in String
If tree.isExternal(root) is False
If node only has one child (tree.numChildren(root)=1)
Throw IllegalArgumentException
Return root.getelment()+" "+ tree2infix(tree, tree.left(root))+" "+tree2infix(tree, tree.right(root))
if tree.isExternal(root) is True
if root is operator(+,-,*)
Throw IllegalArgumentException
Return the element of the leaf

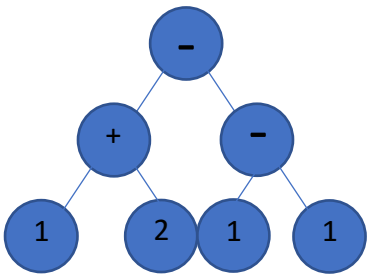
Big O worst case runtime and justification: →O(n)

root.getelment()+" "+ tree2infix(tree, tree.left(root))+" "+tree2infix(tree, tree.right(root))

O(1)+T(n-1)+O(1) (first graph) O(1)+T(n/2)+(T/2)(second graph)

Since Other lines are O(1)





T(n-1)+2O(1)= T(n-1)+O(1)=O(n)

2T(n/2)+O(1)=O(n)

- public static LinkedBinaryTree<String> simplify(LinkedBinaryTree<String> tree) throws IllegalArgumentException

Algorithm: I implement the method recursively. Traversing the the binary tree in preorder. I design a helper method that have one more input called Position<String> root than original method for recursion.

simplify(LinkedBinaryTree<String> tree) throws IllegalArgumentException
if tree is null
throw IllegalArugmentException
return simplify(tree, tree.root())
simplify(LinkedBinaryTree<String> tree, Position<String> root) throws IllegalArumentException
Input: linked binary tree and current node Output: a simplify tree
LinkedBinaryTree<String> newtree=new LinkedBinaryTree<String>();

If tree.isExternal(root) is False
LinkedBinaryTree<String> lefttree= simplify(tree,tree.left(root));
LinkedBinaryTree<String> righttree=simplify(tree,tree.right(root));
If root equals "+"
If left child (simplify(tree, tree.left(root)) of current root is a tree with only a root and the root is a letter
newtree.addRoot("+");
newtree.attach(newtree.root(),lefttree,righttree)
return newtree
If right child(simplify(tree, tree.right(root)) of current root is a tree with only a root and the root is a letter
newtree.addRoot("+");
newtree.attach(newtree.root(),lefttree,righttree)
return newtree
If left child of current root is a tree has number of nodes greater than 1
newtree.addRoot("+");
newtree.attach(newtree.root(),lefttree,righttree)
return newtree
If right child of current root is a tree has number of nodes greater than 1
newtree.addRoot("+");
newtree.attach(newtree.root(),lefttree,righttree)
return newtree

If left child(simplify(tree, tree.left(root)) of current root is a tree with only a root and is a negative Integer&&right child(simplify(tree, tree.right(root)) of current root is a tree with only a root and is a negative Integer

```
result=Integer.parseInt(lefttree.root().getElement().substring(1))+Integer.parseInt(righttree.root().getElement().substring(1));
newtree.addRoot("-"+result+"");
return newtree;
```

If left child(simplify(tree, tree.left(root)) of current root is a tree with only a root and is a negative Integer

```
int result=Integer.parseInt(righttree.root().getElement())
-Integer.parseInt(lefttree.root().getElement().substring(1));
newtree.addRoot(result+"");
return newtree;
```

If right child(simplify(tree, tree.right(root)) of current root is a tree with only a root and is a negative Integer

```
int result=Integer.parseInt(lefttree.root().getElement())
-Integer.parseInt(righttree.root().getElement().substring(1));
newtree.addRoot(result+"");
return newtree;
```

```
//if the left tree and the right tree of the current root does not satisfy above conditions, then go to the normal case
result=Integer.parseInt(lefttree.root().getElement())+Integer.parseInt(righttree.root().getElement());
newtree.addRoot(result+"");
return newtree;
```

Notice: since there are three operators in the assignment, the method simplify should cover all conditions, which makes the method too long. In order to minimize the length of the report, I omit the "-" and "*" part of algorithm. The algorithm of the these two operator basically is same as "+" algorithm, except for some unique cases such as the 1 - -2 , -1 * 2 etc. I write comments for each line of my code to make my code readable.

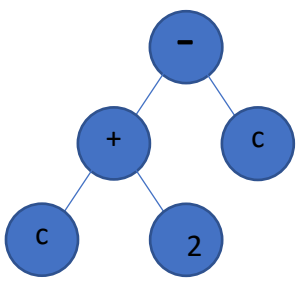
```
if tree.isExternal(root) is True
    newtree.addRoot(root.getElement());
    return newtree
```

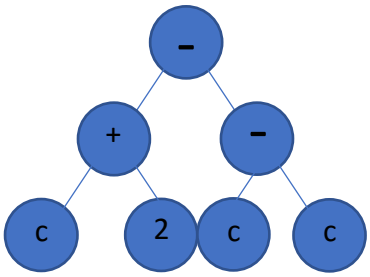
Big O worst case runtime and justification: →O(n)

```
LinkedBinaryTree<String> lefttree= simplify(tree,tree.left(root));
LinkedBinaryTree<String> righttree=simplify(tree,tree.right(root));
```

T(n-1)+O(1) (graph one) T(n/2)+T(n/2)(graph 2)

Since Other lines are O(1)





T(n-1)+O(1)=O(n) 2T(n/2)+O(1)=O(n)

- public static LinkedBinaryTree<String> simplifyFancy(LinkedBinaryTree<String> tree) throws IllegalArgumentException

Algorithm: since the method is a update function of simplify, the algorithm of simplifyFancy is similar to simplify, except for some special rules that we should add in each operator if statements.

```
LinkedBinaryTree<String> lefttree= simplifyFancy(tree,tree.left(root));
LinkedBinaryTree<String> righttree=simplifyFancy(tree,tree.right(root));
```

Regarding to "+":

If left tree of current root only have one node is 0 or right tree of current root only have one node is 0

Return lefttree// if right tree equals 0

or Return righttree// if left tree equals 0

//in the code the algorithm above is implemented by using two if statement

Regarding to "-":

If left tree of current root only have one node is 0

```
newtree.addRoot("-");
newtree.attach(newtree.root(),lefttree,righttree);
return newtree;
```

If right tree of current root only have one node is 0

Return simplifyFancy(tree.tree.left(root))

If left tree and right tree of current root is completely equal

newtree.addRoot("0");

return newtree;

Regarding to "*":

If left tree of current root only have one node is 1 or right tree of current root only have one node is 1

Return lefttree// if right tree equals 1

or Return righttree// if left tree equals 1

//in the code the algorithm above is implemented by using two if statement

If left tree of current root only have one node is 0 or right tree of current root only have one node is 0

newtree.addRoot("0");

return newtree;

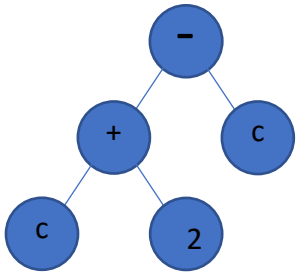
Big O worst case runtime and justification:

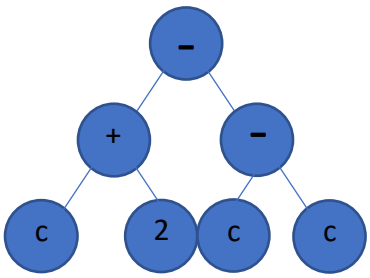
Except for operator "-", the worst case runtime **is O(n)**

```
LinkedBinaryTree<String> lefttree= simplifyFancy(tree,tree.left(root));
LinkedBinaryTree<String> righttree=simplifyFancy(tree,tree.right(root));
```

T(n-1)+O(1) (graph one) T(n/2)+T(n/2)(graph 2)

Since Other lines are O(1)



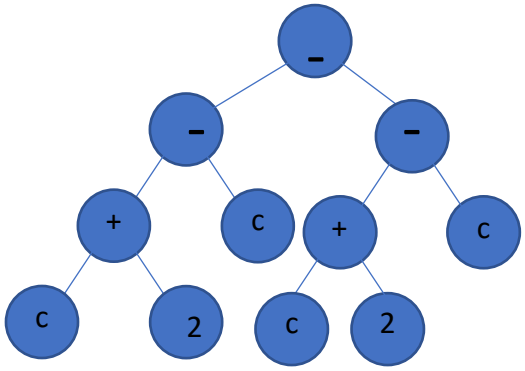


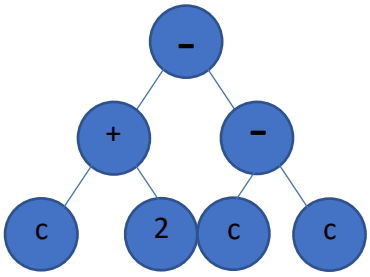
T(n-1)+O(1)=O(n) 2T(n/2)+O(1)=O(n)

In the case of operator "-", the worst case runtime **is O(n^2)**

The special case x-x, the method use equals(treeA,treeB) to compare those two subtree whether completely same, which equals spend O(n) to compare each node of the linked binary tree with height O(n)

Since other lines are O(1),





T(n-1)+O(n)=O(n^2) 2T(n/2)+O(n)=O(nlogn)

- public static LinkedBinaryTree<String> substitute(LinkedBinaryTree<String> tree, String variable, int value) throws IllegalArgumentException

Algorithm: since we cannot substitute an operator and substitute leaf with null or operator, or Integer, the leaves are the nodes we can substitute Letter with Integer. Helper method same as above helper method add one more parameter Position<String> root. The base case is the root is a leaf.

substitute(LinkedBinaryTree<String> tree, String variable, int value) throws IllegalArgumentException

if tree is null or variable is null or variable is Integer or operator

throw IllegalArugmentException

return substitute(tree, tree.root(),variable,value)

substitute(LinkedBinaryTree<String> tree, Position<String> root, String variable, int value) throws IllegalArumentException

Input: linked binary tree ,current node, variable, value Output: a substitute tree

LinkedBinaryTree<String> newtree=new LinkedBinaryTree<String>();

If tree.isExternal(root) is False

If node only has one child (tree.numChildren(root)=1)

Throw IllegalArgumentException

newtree.addRoot(root.getElement())

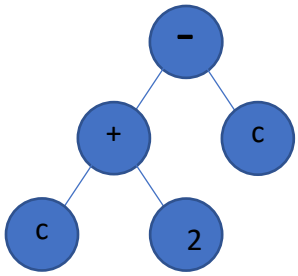
```
            newtree.attach(newtree.root(),substitute(tree,tree.left(root),variable,value),
            substitute(tree,tree.right(root),variable,value));
            return newtree;
        else
            if root is operator(+,-,*)
                    Throw IllegalArgumentException
            If root.getElment equals variable
                        newtree.addRoot(value+"");
                                return newtree;
             else

                        newtree.addRoot(root.getElement());
                                return newtree;
```

Big O worst case runtime and justification:. →O(n)

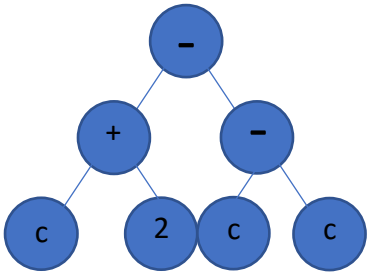
newtree.attach(newtree.root(), substitute(tree,tree.left(root),variable,value), substitute(tree,tree.right(root),variable,value));

O(1)+O(1)+T(n-1)+O(1)(first graph) O(1)+O(1)+T(n/2)+T(n/2)(second graph)

other lines are O(1)



T(n-1)+O(1)=O(n)



2T(n/2)+O(1)=O(n)

- public static LinkedBinaryTree<String> substitute(LinkedBinaryTree<String> tree, HashMap<String, Integer> map) throws IllegalArgumentException

Algorithm: The basic idea of the method is similar to the previous substitute method, except for the method can substitute several String with Integer instead of one pair in the previous substitute.

In the base case, adding a condition Map.containsKey(root.getElement()) to check the current root whether contained in the map or not. If map does contain the root, replace it with provided value, if not changing nothing.

```
            if(map.containsKey(root.getElement())){
                        if(map.get(root.getElement())==null){
                                throw new IllegalArgumentException();
                        }
                        newtree.addRoot(map.get(root.getElement())+"");
                        return newtree;
                }
```

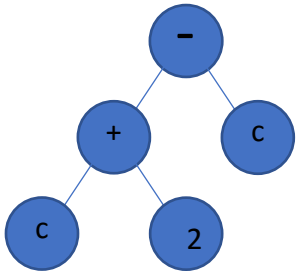
```
newtree.addRoot(root.getElement());
return newtree;
```

Big O worst case runtime and justification: →O(n)

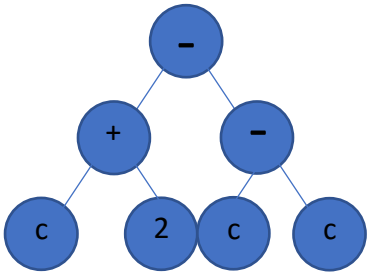
```
newtree.attach(newtree.root(), substitute(tree,tree.left(root),variable,value),
substitute(tree,tree.right(root),variable,value));
```

O(1)+O(1)+T(n-1)+O(1)(first graph) O(1)+O(1)+T(n/2)+T(n/2)(second graph)

since other lines are O(1)



T(n-1)+O(1)=O(n)



2T(n/2)+O(1)=O(n)

- public static boolean isArithmeticExpression(LinkedBinaryTree<String> tree)

  Algorithm:

  1. The tree provided is null
  2. A node only have one child
  3. Leaf is an operator

     When the method reach one of the three conditions above, method should return false.

  Implemented the method recursively. Base case is that the current node is a leaf, and if the lead is an operator, method return false, else return true

  If tree.isExternal(root) is False

  If node only has one child (tree.numChildren(root)=1)

  return false;

  only if isArithmeticExpression(tree,tree.left(root)) and isArithmeticExpression(tree,tree.right(root)) all return true, method will return true

  else

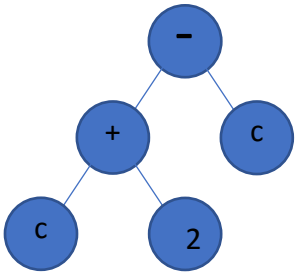
  return false

Big O worst case runtime and justification: →O(n)

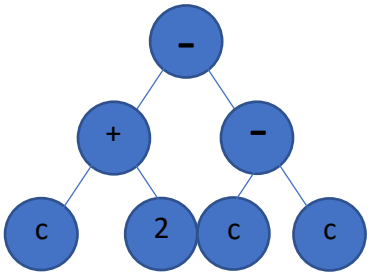
```
isArithmeticExpression(tree,tree.left(root))&&isArithmeticExpression(tree,tree.right(root))
```

T(n-1)+O(1) (first graph) T(n/2)+T(n/2)(second graph)

Since other lines are O(1)



T(n-1)+O(1)=O(n)



2T(n/2)+O(1)=O(n)