CS311 Yoshii Expression Trees

Expression trees are very common in computer science.

Given an arithmetic expression, we often want to build a tree representation of it.

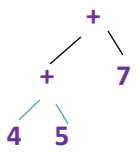
Let's see some examples:

4 + 5 + 7 is an expression.

We want to use parentheses to show which parts need to be done first.

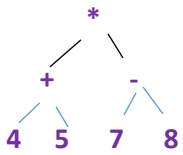
(4 + 5) + 7

Numbers will be the leaves. Operators will be the internal nodes. Parentheses indicate a sub-tree.



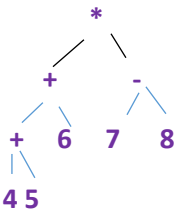
Notice that the numbers appear as leaves in the same order as in the expression.

Given (4 + 5) * (7 – 8) . you can tell that there will be two sub-trees with * are the root.



You can have nested parentheses for sub-sub-trees.

Given ((4 + 5) + 6) * (7 – 8)

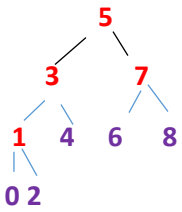


Why do we use trees for expressions?

The most common reason is so that the compiler can "parse" arithmetic expressions in such a way that if we did an in-order traversal of the tree, we can compute the expression.

Another reason is to view a tree in a way that is easy to manipulate as we do with AVL search trees.

For AVL, we do not really have arithmetic expressions. But given the following binary search tree



You notice that it has the same shape as our last arithmetic example. Red nodes correspond to the operators.

Therefore, the expression for this tree is:

((0 **1** 2) **3** 4) **5** (6 **7** 8))

Try drawing a tree for ((0 **1** 2) **3** (4 **5** 6))

Because it is a binary search tree, the numbers will always be "increasing" in the expression.