WORKSHEET

A4.7 Tree traversals

Trees and graphs · A level · OCR H446 2.3.1, AQA 7517 4.3.2.1, Eduqas A500QS 1.1 · about 25 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

Pre-order, in-order and post-order, the outline method, expression trees and what each traversal is for.

- Three orders
- The outline method
- What each traversal is for
- Expression trees
- Breadth-first on a tree

Questions

6 marks in all

1. Which traversal outputs the keys of a binary search tree in ascending order? [1 mark]

- ☐ **A** In-order
- ☐ **B** Pre-order
- ☐ **C** Post-order
- ☐ **D** Breadth-first

2. Which traversal of an expression tree produces Reverse Polish notation? [1 mark]

- ☐ **A** Post-order
- ☐ **B** Pre-order
- ☐ **C** In-order
- ☐ **D** Breadth-first

3. Put the items of the expression tree for (2 + 3) * 4 in post-order. [1 mark]

Number the lines 1 to 5 to put them in the right order.

___ `*`
___ `4`
___ `2`
___ `+`
___ `3`

4. Which traversal is used to copy a tree, so that inserting the keys into a new tree rebuilds the same shape? [1 mark]

- ☐ **A** Pre-order
- ☐ **B** In-order
- ☐ **C** Post-order
- ☐ **D** Any of them gives the same shape

5. What does this program print? [1 mark]

```
class Node:
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right

def walk(node):
    if node is None:
        return ""
    return node.item + walk(node.left) + walk(node.right)

tree = Node("M", Node("F", Node("B"), Node("H")), Node("T", None, Node("W")))
print(walk(tree))
```

6. A traversal is written: traverse left, traverse right, then output the node. Which traversal is it, and what is it used for? [1 mark]

- ☐ **A** Post-order, used to empty a tree
- ☐ **B** Pre-order, used to copy a tree
- ☐ **C** In-order, used to output a binary search tree in order
- ☐ **D** Breadth-first, used to find shortest paths

The task: three ways round a tree

`TREE` holds the expression (5 - 1) * (2 + 3 * 4) as a binary tree of `Node` objects. Each node's `item` is an operator (`+`, `-` or `*`) or a single digit, both as strings, and a leaf has `left` and `right` set to `None`. Write recursive functions `preorder(node)`, `inorder(node)` and `postorder(node)` that each return a list of the items in that order, and an empty list for `None`. Print `pre-order: `, `in-order: ` and `post-order: `, each followed by that list's items separated by single spaces, one per line in that order. Then work out the value of the expression from the tree, without `eval`, and print `value: ` followed by the whole number.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

class Node:
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right

# (5 - 1) * (2 + 3 * 4)
TREE = Node("*", Node("-", Node("5"), Node("1")), Node("+", Node("2"), Node("*", Node("3"),
Node("4"))))

def preorder(node):
    return []
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a4-7-tree-traversals/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Write `infix(node)` for the task's tree, with brackets round every operator's subtree, and check it gives the same value as the original expression.
2. Draw the expression tree for 8 / (4 - 2) + 1 and write down all three traversals by hand. Which one is the RPN?
3. Evaluate the task's post-order list with a stack: push numbers, and on an operator pop two values and push the result. Does it match `value`?