ANSWERS

A4.5 Trees

Trees and graphs · A level · OCR H446 1.4.2, AQA 7517 4.2.5.1, Eduqas A500QS 1.1 · about 25 min

BugBotLab

What this lesson is about

Trees as connected graphs with no cycles, rooted trees and their vocabulary, binary trees and typical uses.

- A tree is a graph
- Rooted trees
- Where rooted trees are used
- A rooted tree in Python
- Is it a tree?
- Binary trees

Questions

6 marks in all

1. Which is the definition of a tree? [1 mark]

- ☐ **A** A connected, undirected graph with no cycles
- ☐ **B** A directed graph in which every vertex has two children
- ☐ **C** Any graph with a root
- ☐ **D** An undirected graph in which every vertex has the same degree

> **Answer:** A. A tree does not need a root. A rooted tree is a tree with one vertex designated as the root.

2. A tree has 20 vertices. How many edges does it have? [1 mark]

> **Answer:** 19. A tree with n vertices always has n - 1 edges.

3. In a rooted tree, which node has no parent? [1 mark]

- ☐ **A** The root
- ☐ **B** Every leaf
- ☐ **C** Every internal node
- ☐ **D** The node with the most children

> **Answer:** A. Every other node has exactly one parent and is a descendant of the root.

4. Which of these are typical uses of rooted trees? [1 mark]

Tick every answer that is true.

- ☐ **A** A file system of folders and files
- ☐ **B** An expression tree for an arithmetic expression
- ☐ **C** A binary search tree for fast searching
- ☐ **D** A circular queue of print jobs

> **Answer:** A, B, C. A circular queue is a linear structure, not a tree.

5. What is a binary tree? [1 mark]

- ☐ **A** A rooted tree in which each node has at most two children
- ☐ **B** A tree with exactly two leaves
- ☐ **C** A tree that stores only 0s and 1s
- ☐ **D** A graph with two roots

> **Answer:** A. Each node has a left child, a right child, both or neither.

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

```
CHILDREN = {"r": ["a", "b"], "a": ["c", "d", "e"], "b": ["f"]}
def leaves(node):
    kids = CHILDREN.get(node, [])
    if not kids:
        return 1
    return sum(leaves(k) for k in kids)
print(leaves("r"), leaves("a"))
```

> **Answer:**
>
> ```
> 4 3
> ```
>
> The leaves are c, d, e and f. The subtree rooted at a holds three of them.

The task: facts about a tree

`TREE` is a rooted tree stored as a dictionary: each key is a node that has children, and its value is the list of those children. Leaves appear only inside children lists, never as keys, and the keys are in no particular order, so the root is not necessarily first. Write a recursive function `height(node)` that returns the height of the subtree rooted at `node`, which is 0 for a leaf. Then print five lines: `root:` and the root; `nodes:` and the number of nodes; `edges:` and the number of edges; `leaves:` and every leaf in alphabetical order, separated by single spaces; and `height:` and the height of the whole tree. Work everything out from `TREE`.

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

TREE = {
    "fetch": ["approach", "grip"],
    "mission": ["find", "fetch", "return"],
    "approach": ["align", "creep"],
    "find": ["scan", "turn"],
    "return": ["drive", "drop"],
}

def height(node):
    return 0
```

The hint students can ask for: The root is the one node that is nobody's child. A leaf is a node with no children. The height of a node is 0 for a leaf, otherwise one more than the tallest of its children.

A solution

```
from bugbot import *
connect()

TREE = {
    "fetch": ["approach", "grip"],
    "mission": ["find", "fetch", "return"],
    "approach": ["align", "creep"],
    "find": ["scan", "turn"],
    "return": ["drive", "drop"],
}

children_seen = []
for parent in TREE:
    children_seen = children_seen + TREE[parent]
nodes = sorted(set(TREE) | set(children_seen))
root = [n for n in nodes if n not in children_seen][0]

def kids(node):
    return TREE.get(node, [])

def height(node):
    if not kids(node):
        return 0
    return 1 + max(height(c) for c in kids(node))

print("root:", root)
print("nodes:", len(nodes))
print("edges:", len(children_seen))
```

```
print("leaves:", " ".join(n for n in nodes if not kids(n)))
print("height:", height(root))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.