

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Binary Trees: Why They Matter, Their Properties, and How to Walk Them

Week 7 · Wednesday 18 November 2026 · Canning, Broder & Lafore Ch 8

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–10 min) Why trees: where they appear, and the vocabulary
2. (10–20 min) Binary trees: properties and types
3. (20–24 min) Building a binary tree in Python
4. (24–34 min) Traversals: why we need them, then four ways to walk a tree
5. (34–40 min) Expression trees, rebuilding a tree, practice

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Explain	why trees exist, and name real systems that use them
Apply	the properties: level capacity, height versus size, and counting leaves
Classify	a binary tree as full, perfect, complete or degenerate
Traverse	preorder, inorder, postorder and level order, by hand and in code; rebuild a tree from two traversals

Why Not Just Use a List?

IIT Madras Zanzibar

A list: one line, one next item

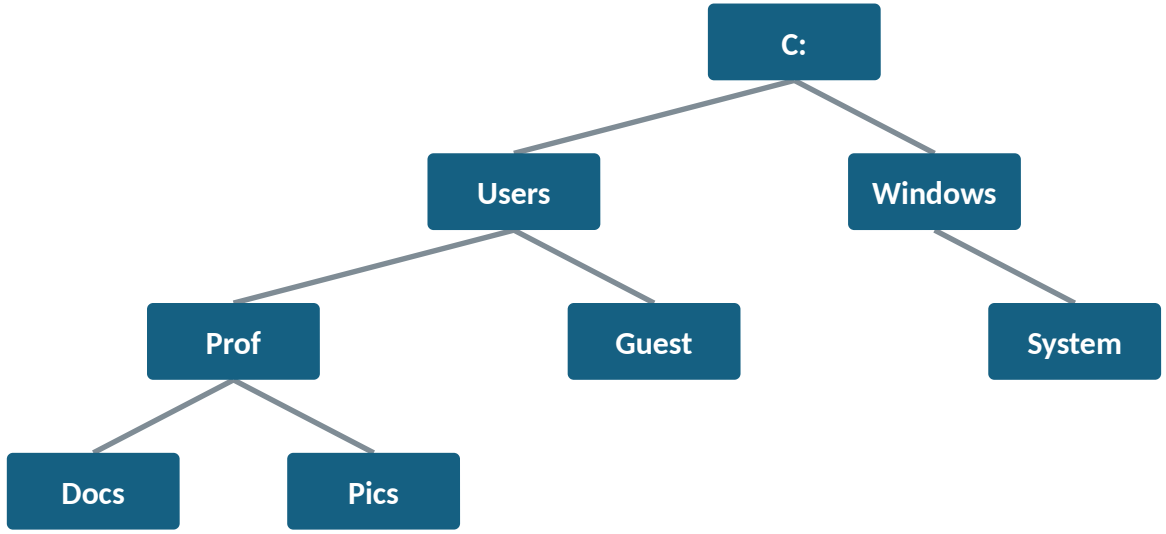
C:	Users	Windows	Prof	Docs	Pics	Guest	System

Where is Prof's folder? Scan from the left until you meet it.

Which items belong to Prof? The list cannot say.

How big is Users? No idea where it ends.

A tree: branches and parents



Every item has one parent: Docs and Pics sit under Prof

? **Think:** A laptop holds 300,000 files. Give two things a tree can do for you that a single flat list cannot.

Answer: Group related items so a search skips whole branches and give every item a path and an owner. A balanced search tree finds one of 300,000 items in about 19 steps.

Where Trees Appear

IIT Madras Zanzibar

1. **File systems**

   Folders hold files and folders; a path is a walk down from the root.

2. **Web pages**

   The browser keeps a page as a tree of elements and walks it to draw the page.

3. **Compilers**

   (3 + 4) × 2 is stored as an expression tree; evaluating it is a traversal.

4. **Databases**

   B-trees, a many-child cousin of binary trees, keep disk indexes sorted and shallow.

5. **Decision trees**

   A classifier asks a yes or no question at each node and follows a branch.

6. **Compression**

   Huffman coding builds a binary tree; the left and right path to a letter is its bit code.

7. **Heaps**

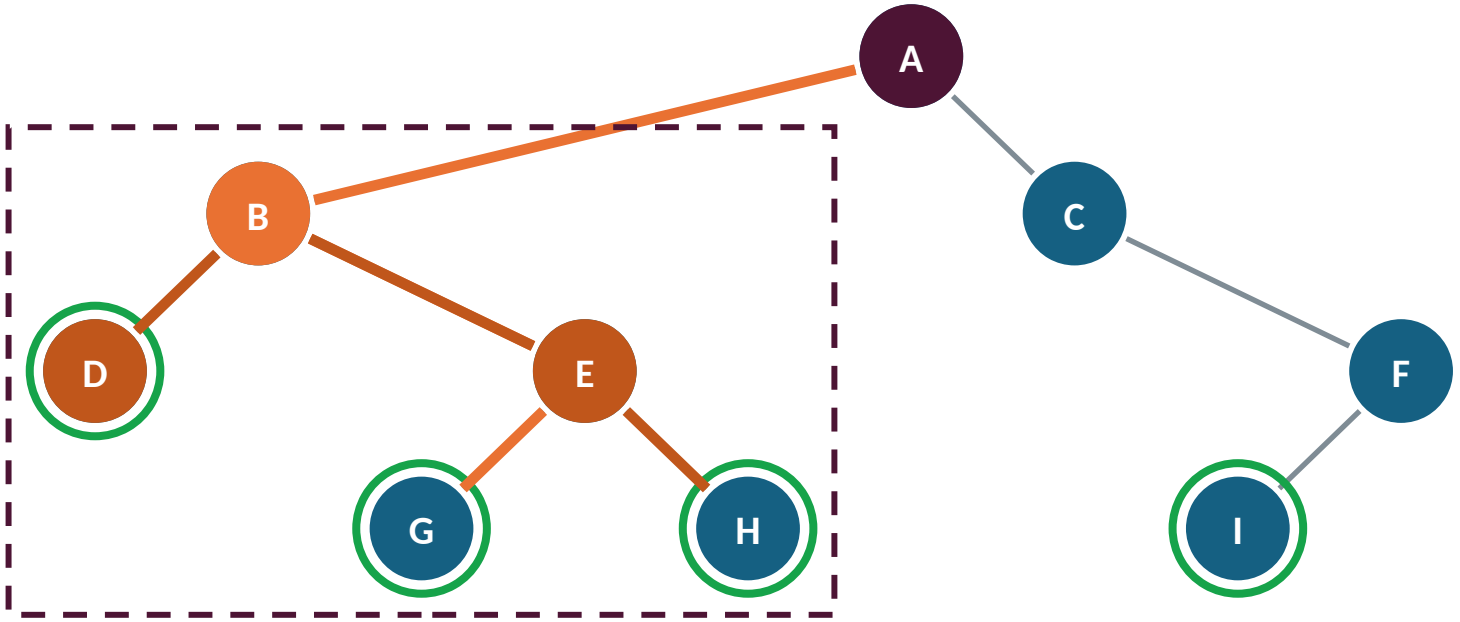
   A priority queue is a binary tree stored in an array. Next week.

8. **Search, games**

   Binary search trees on Thursday, and the move trees chess programs explore.

Pattern: most of these are binary trees or built from them, because two branches are enough for yes or no, smaller or larger.

A Tree, Named Piece by Piece

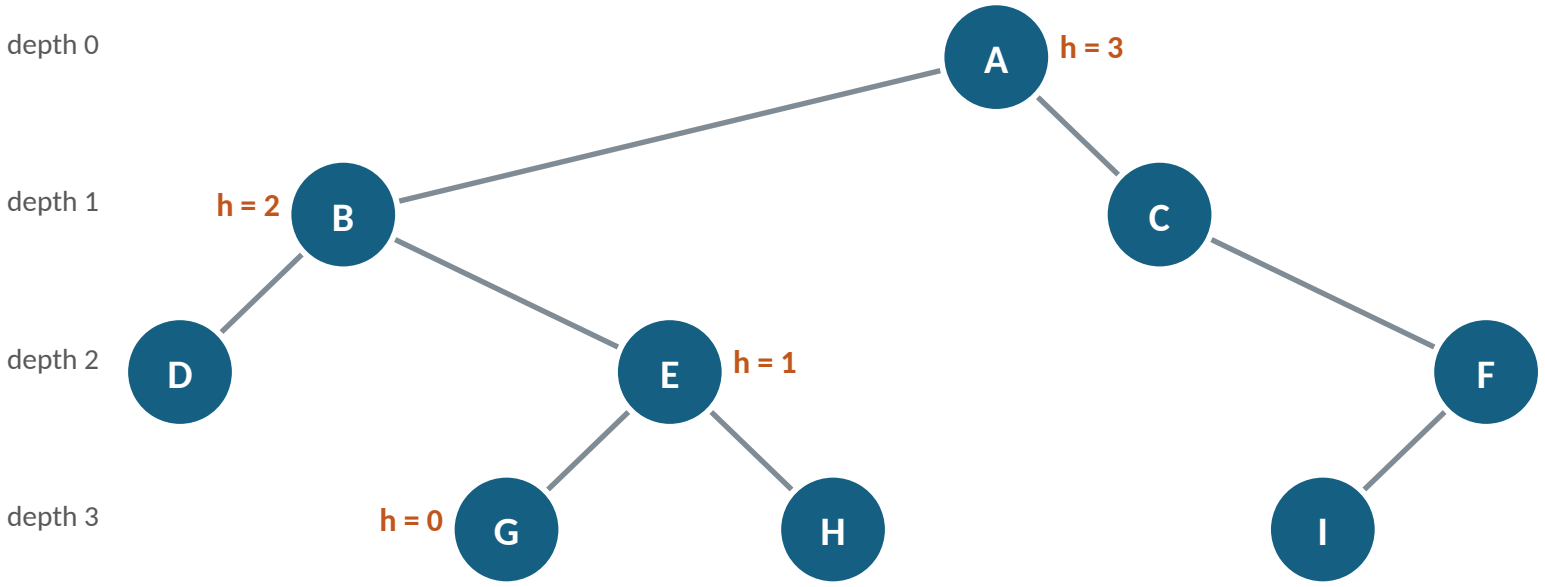


1. Root: the one node with no parent (A)
2. Parent and children: B is the parent of D and E; D and E are siblings
3. Leaf: a node with no children (D, G, H, I). The others are internal nodes
4. Ancestors of G: E, B, A, the nodes on the way up to the root
5. Subtree at B: B and everything below it (B, D, E, G, H)
6. Path: D, B, E, H has 3 edges. Any two nodes have exactly one path

? **Think:** Which nodes are the ancestors of I, and how many leaves does the subtree at C contain?

Answer: Ancestors of I: F, C and A (parent, grandparent, root). The subtree at C is C, F, I, and its only leaf is I.

Depth, Level, Height, Degree



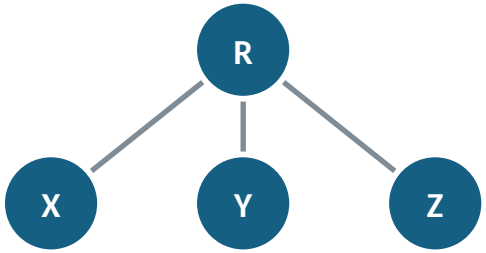
Definitions

1. depth(v): edges from the root down to v. Same as the level of v
2. height(v): edges on the longest path from v down to a leaf
3. height of the tree = height of the root (here 3)
4. empty tree: height −1, so one node has height 0
5. degree(v): number of children: 0, 1 or 2

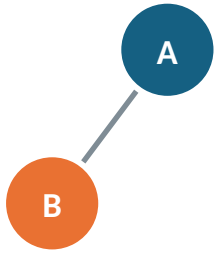
? **Think:** What is the height of node C and the depth of node H? Can a node's height ever exceed its depth?

Answer: Height of C is 2 (C, F, I). Depth of H is 3 (A, B, E, H). Yes: the root has depth 0 but height 3.

What Makes a Tree Binary

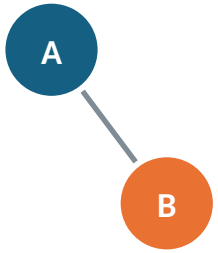


General tree: any number of children
not binary



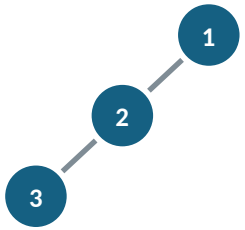
B is the LEFT child of A

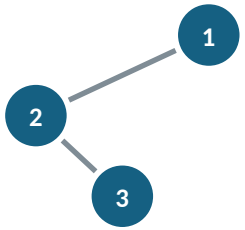
≠

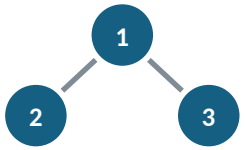


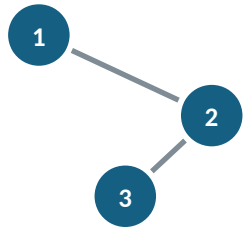
B is the RIGHT child of A

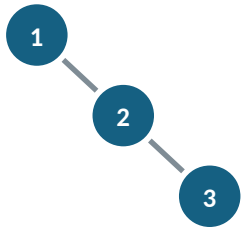
Definition: Binary tree: every node has at most two children, a left child and a right child, and the two positions are different.









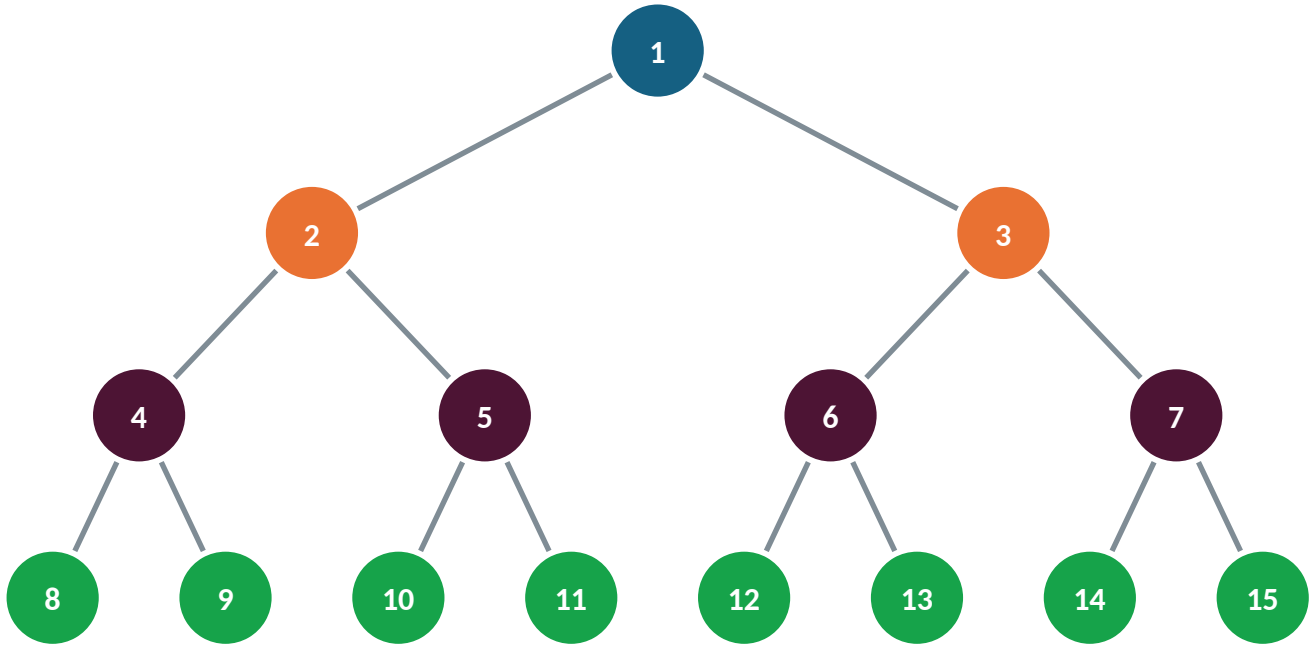


? **Think:** How many different binary trees can be drawn with exactly 3 nodes? Draw them all before the answer appears.

Answer: Five. For 1, 2, 3, 4 nodes the counts are 1, 2, 5, 14 (the Catalan numbers).

Property 1: Each Level Can Hold Twice as Many

IIT Madras Zanzibar



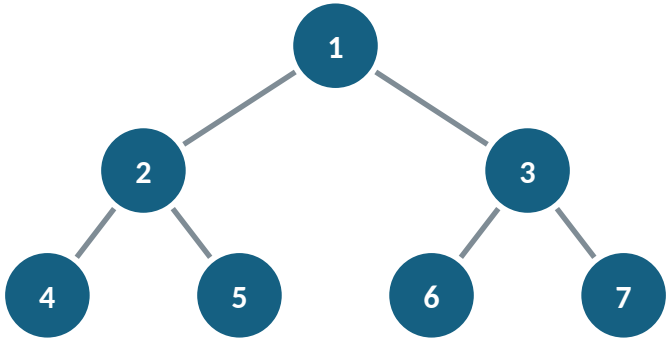
level	at most	so far
0	2^0 = 1	1
1	2^1 = 2	3
2	2^2 = 4	7
3	2^3 = 8	15

Formulas: level L holds at most 2^L nodes; height h holds at most 2^(h+1) − 1 nodes.

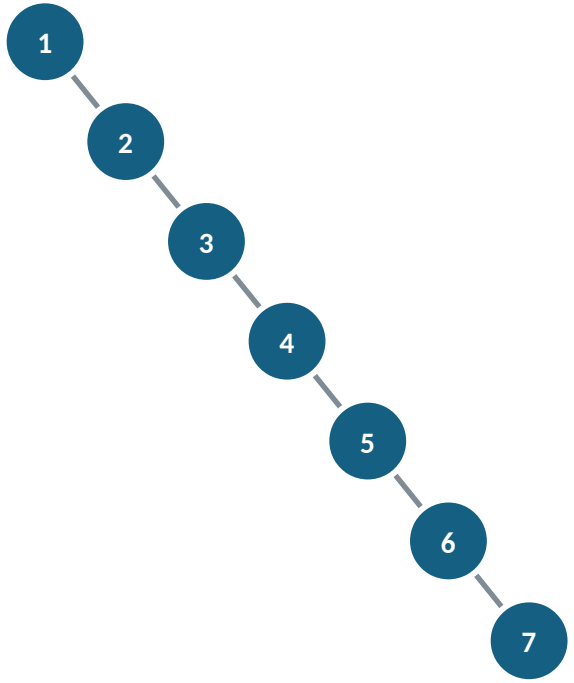
? **Think:** A perfect binary tree has 255 nodes. What is its height, and how many of the nodes are leaves?

Answer: 2^(h+1) − 1 = 255 gives h = 7. The bottom level holds 2^7 = 128 nodes, all leaves: just over half of the tree.

Property 2: Height Versus Number of Nodes



packed: n = 7, height 2



chain: n = 7, height 6

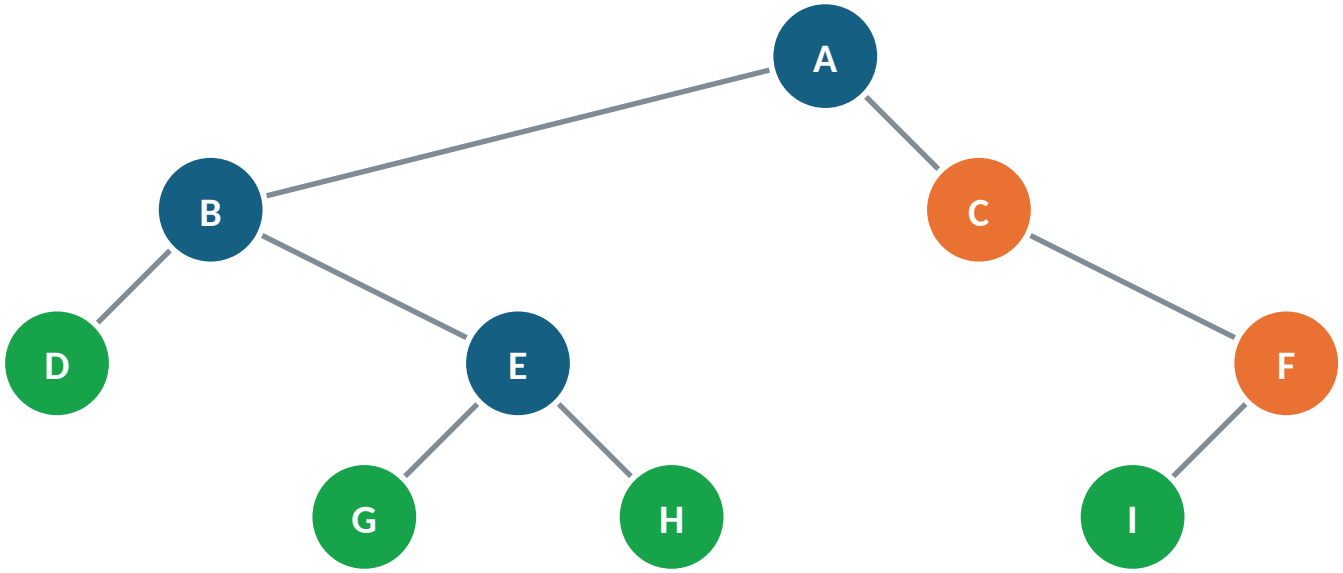
nodes n	min height	max height
7	2	6
15	3	14
1,000	9	999
1,000,000	19	999,999
1,000,000,000	29	999,999,999

min = $\lceil \log_2(n+1) \rceil - 1$
max = n − 1

? **Think:** What are the smallest and the largest possible heights of a binary tree with 100 nodes?

Answer: Smallest 6: height 5 holds only 63 nodes, height 6 holds up to 127. Largest 99: a chain. Same 100 nodes, depth 6 or 99.

Property 3: Counting Edges and Leaves



leaves: L = 4 (D, G, H, I)

one child: D1 = 2 (C, F)

two children: D2 = 3 (A, B, E)

1. Count the nodes by kind: n = L + D1 + D2. Here 4 + 2 + 3 = 9 nodes.

2. Count the edges two ways. Child side: every node except the root has one edge up, so n − 1 = 8. Parent side: D1 + 2·D2 = 2 + 6 = 8.

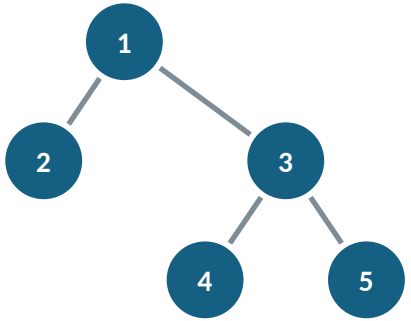
3. The two counts are equal: L + D1 + D2 − 1 = D1 + 2·D2. Cancel D1: L = D2 + 1. Check: 4 = 3 + 1. One-child nodes do not matter.

Idea: a one-child node continues a branch; a two-child node splits it in two. Every branch ends in one leaf.

? **Think:** A binary tree has 8 leaves and 5 nodes with exactly one child. How many nodes does it have in total, and how many edges?

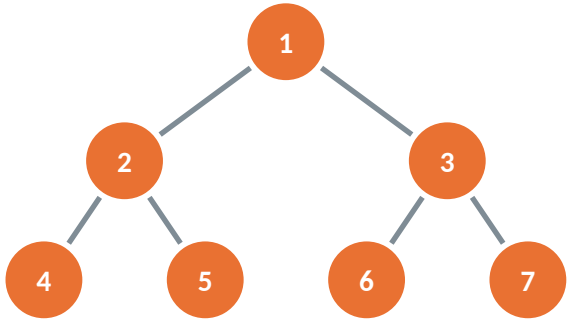
Answer: Two-child nodes = leaves − 1 = 7. Total nodes = 8 leaves + 5 one-child + 7 two-child = 20. Edges = nodes − 1 = 19.

Types of Binary Trees



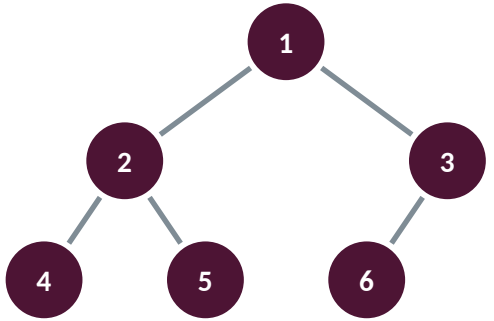
full

Every node has 0 or 2 children, never 1.



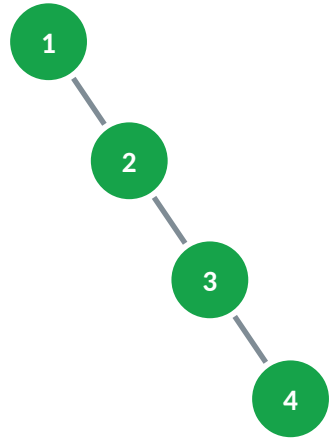
perfect

Full, with all leaves on one level: 2^(h+1) − 1 nodes.



complete

All levels full except the last, which fills from the left. Heaps use this.

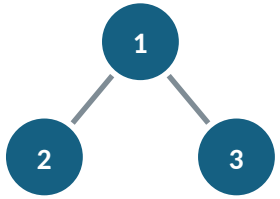


degenerate

Every node has at most one child: a linked list in tree clothing, height n − 1.

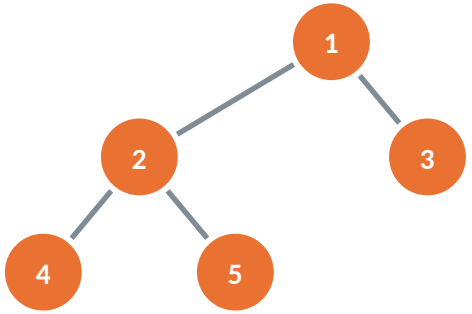
Also: balanced means that at every node the heights of the left and right subtrees differ by at most 1. Every perfect tree is full and complete; the reverse is false.

Classify These Five Trees



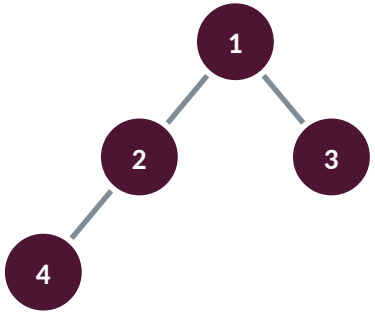
P

perfect (so also full and complete)



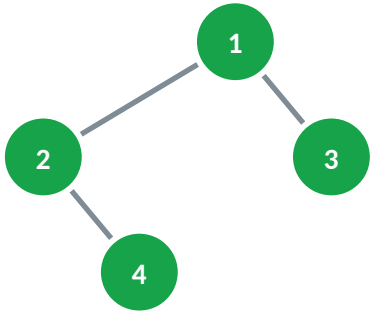
Q

full and complete, not perfect: leaf 3 is above the last level



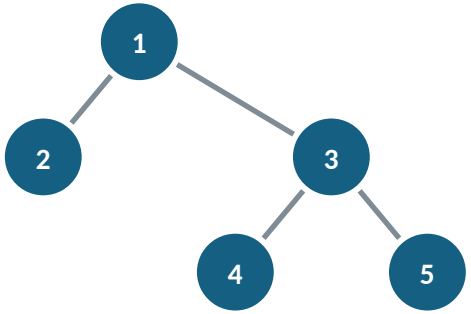
R

complete, not full: node 2 has one child



S

neither: 4 is a right child but 2 has no left child



T

full, not complete: 4 and 5 sit under 3 while 2 is a leaf

? **Think:** for each tree decide full, complete, perfect. Then explain why every perfect tree is full and complete, but not the other way round.

Storing a Binary Tree: the Node Class

IIT Madras Zanzibar

```
1  class Node:
2      def __init__(self, value):
3          self.value = value
4          self.left = None
5          self.right = None
```

Line by line

Line 1 a new kind of object called Node

Line 2 the constructor: runs when we write Node(7)

Line 3 store the value inside this node

Line 4–5 two arrows, left and right, both None: no children yet, so a new node is a leaf

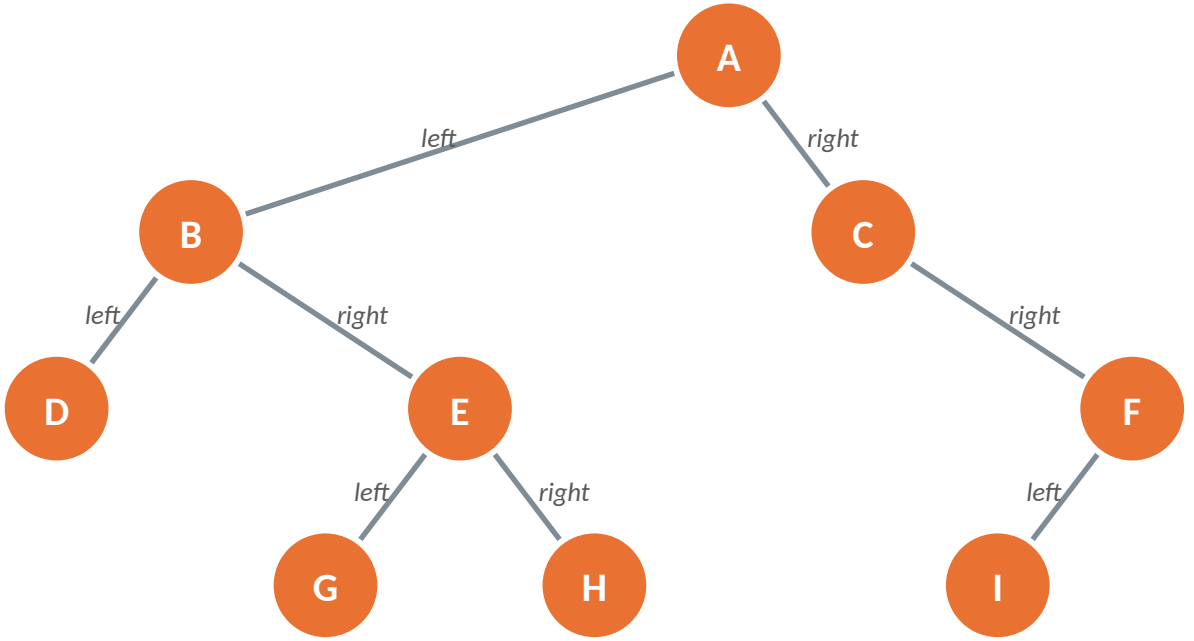
? **Think:** t = Node(1); u = t; u.left = Node(2). Does t have a left child now? Explain.

Answer: Yes. t and u are two names for one Node object, so u.left is t.left. Variables hold references, not copies.

Building the Tree, Line by Line

IIT Madras Zanzibar

```
root = Node("A")
root.left = Node("B")
root.right = Node("C")
root.left.left = Node("D")
root.left.right = Node("E")
root.left.right.left = Node("G")
root.left.right.right = Node("H")
root.right.right = Node("F")
root.right.right.left = Node("I")
```



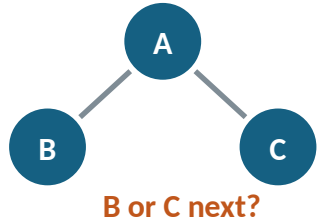
Think: Write the one line that makes J the right child of D. What is the height of the tree afterwards?

Answer: root.left.left.right = Node("J"). J sits on level 3, as deep as G, H and I, so the height stays 3.

Why Walk a Tree?

IIT Madras Zanzibar

The problem: a list has an obvious next item. A tree does not: after the root, which child comes next? A traversal is a rule that fixes one order and visits every node exactly once.



1 Copy a tree

Build a duplicate, node by node, with every link intact.

parent BEFORE children

2 Total folder size

Add up every file below a folder, subfolders included.

children BEFORE parent

3 Delete a tree

Free every node without losing a link we still need.

children BEFORE parent

4 Evaluate (3+4)×2

The expression is a tree of operators and numbers.

children BEFORE parent

5 Sorted output

Print the keys of a search tree from smallest to largest.

left, parent, right

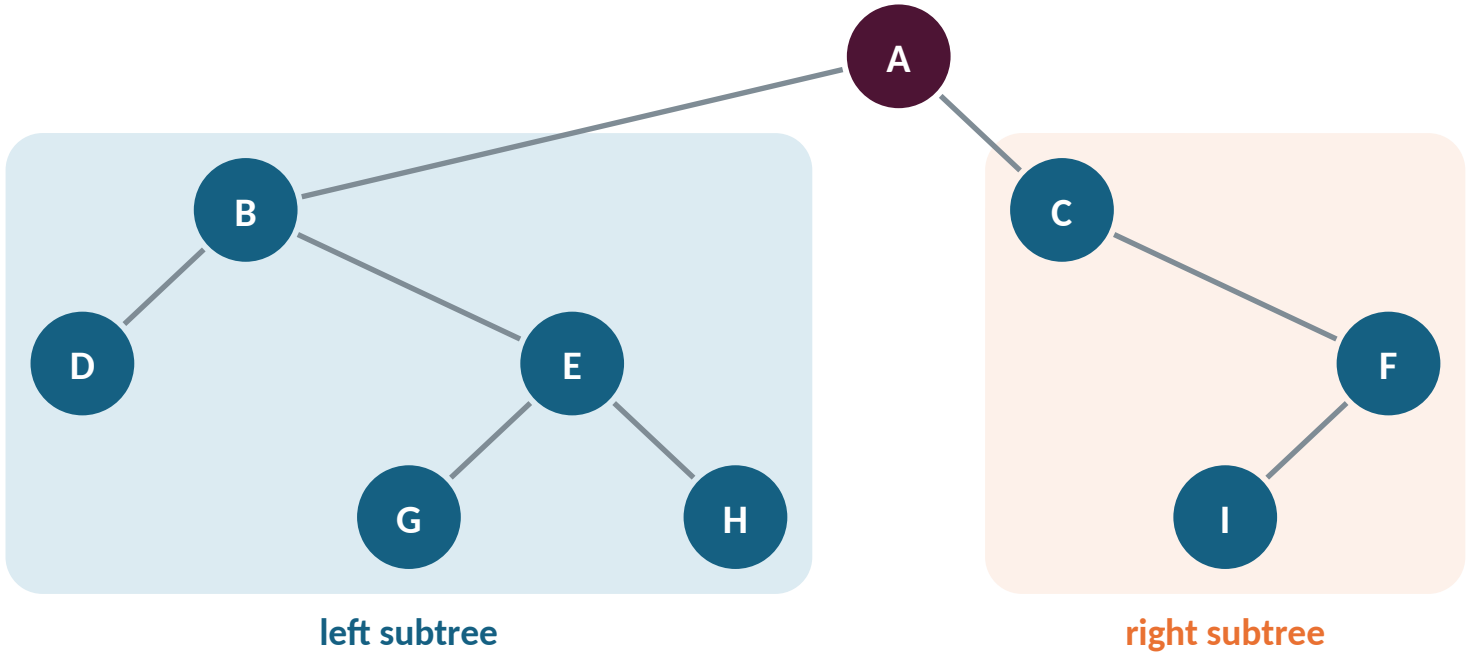
6 Nearest first

Visit the root, then its children, then their children.

level by level

? **Think:** for each task, decide when the parent is visited: before its children, after them, or between them.

Three Jobs at Every Node: V, L, R



Every subtree is a smaller tree, so one rule works at every node. Three jobs:

- **V** visit the node
- **L** walk the left subtree
- **R** walk the right subtree

preorder	V	L	R
inorder	L	V	R
postorder	L	R	V

? **Think:** V, L and R can be ordered in 3! = 6 ways. Why do we only name three traversals?

Answer: We always go left before right. That leaves three places for V: first (preorder), between (inorder), last (postorder).

Three Orders, One Code Shape

IIT Madras Zanzibar

```
def preorder(node):
    if node is None: return
    print(node.value)
    preorder(node.left)
    preorder(node.right)
```

node, LEFT, RIGHT

```
def inorder(node):
    if node is None: return
    inorder(node.left)
    print(node.value)
    inorder(node.right)
```

LEFT, node, RIGHT

```
def postorder(node):
    if node is None: return
    postorder(node.left)
    postorder(node.right)
    print(node.value)
```

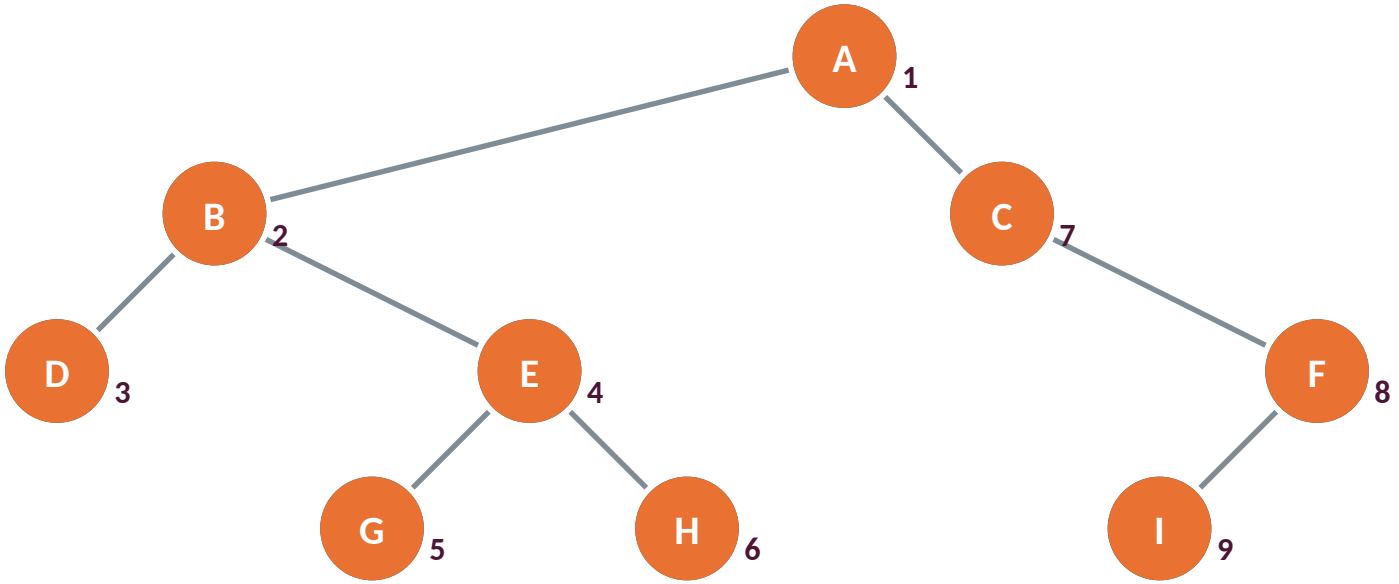
LEFT, RIGHT, node

Cost: time O(n), since every node is visited once with constant work; extra memory is the recursion stack, as deep as the height of the tree.

? **Think:** How would you change inorder so that it prints a search tree from the largest key to the smallest?

Answer: Swap the two recursive calls: right subtree, then print, then left subtree. The right side holds the larger keys.

Preorder: Node, Left, Right



output

A	B	D	E	G	H	C	F	I

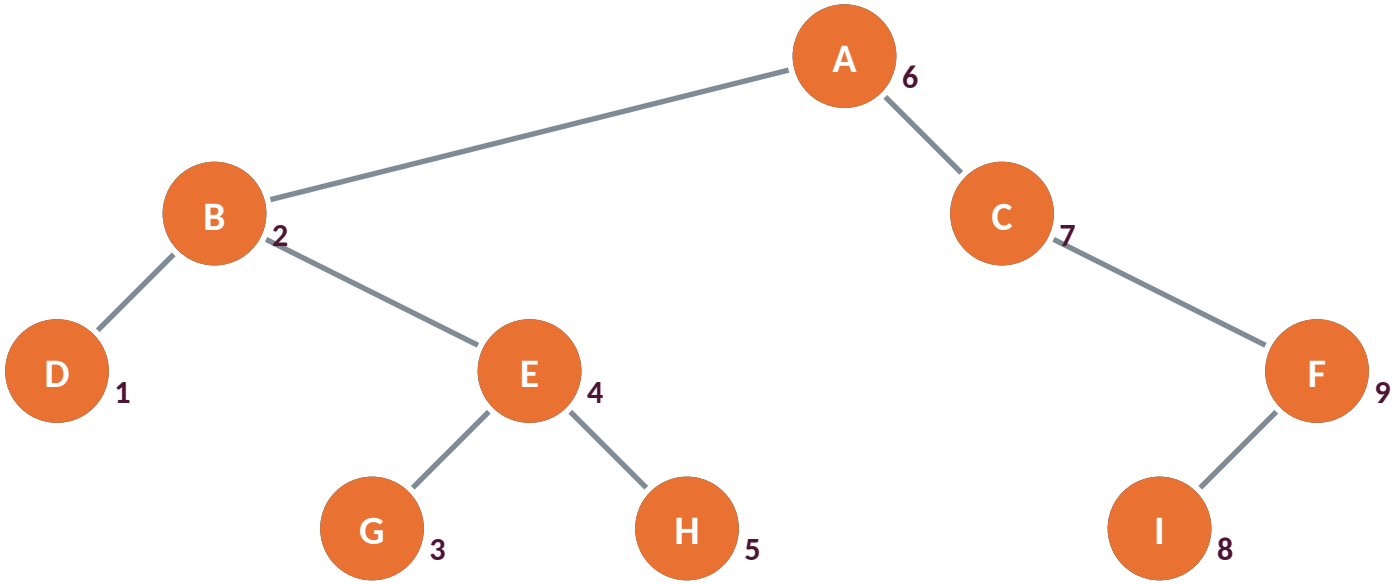
preorder: the NODE first, then the left subtree, then the right subtree

1. start at the root: print A
2. go left to B: print B
3. go left to D: print D
4. left side of B is finished; go right to E: print E
5. go left to G: print G
6. left side of E is finished; go right to H: print H
7. left side of A is finished; go right to C: print C
8. C has no left child; go right to F: print F
9. go left to I: print I

? **Think:** In the preorder A B D E G H C F I, how can you spot the root of every subtree using only the output?

Answer: A subtree is one consecutive block of the output, and its first item is its root: B D E G H starts with B, C F I starts with C.

Inorder: Left, Node, Right



inorder: the LEFT subtree, then the NODE, then the right subtree

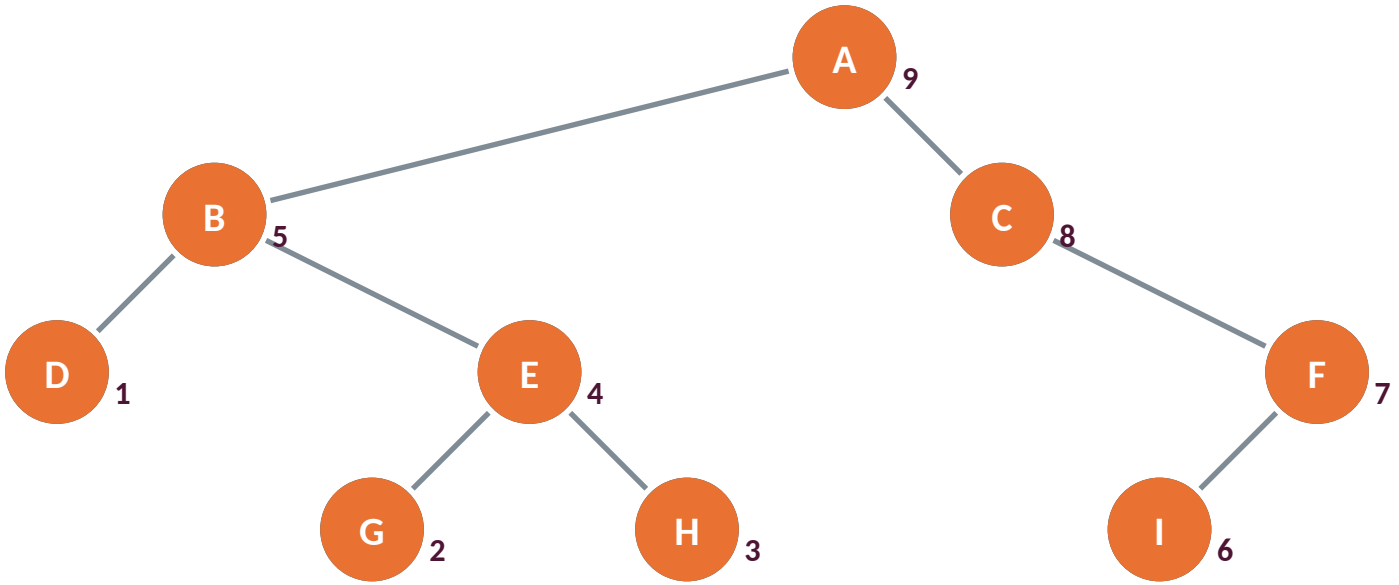
1. dive left A, B, D; D has no left child: print D
2. left side of B is finished: print B
3. G has no left child: print G
4. left side of E is finished: print E
5. H has no left child: print H
6. left side of A is finished: print A
7. C has no left child: print C
8. I has no left child: print I
9. left side of F is finished: print F

output	D	B	G	E	H	A	C	I	F

? **Think:** In the inorder D B G E H A C I F, which node comes immediately before the root A and which comes immediately after? Describe each in words.

Answer: Before: H, the rightmost node of the left subtree. After: C, the leftmost node of the right subtree. Friday's delete uses these neighbors.

Postorder: Left, Right, Node



output

D	G	H	E	B	I	F	C	A

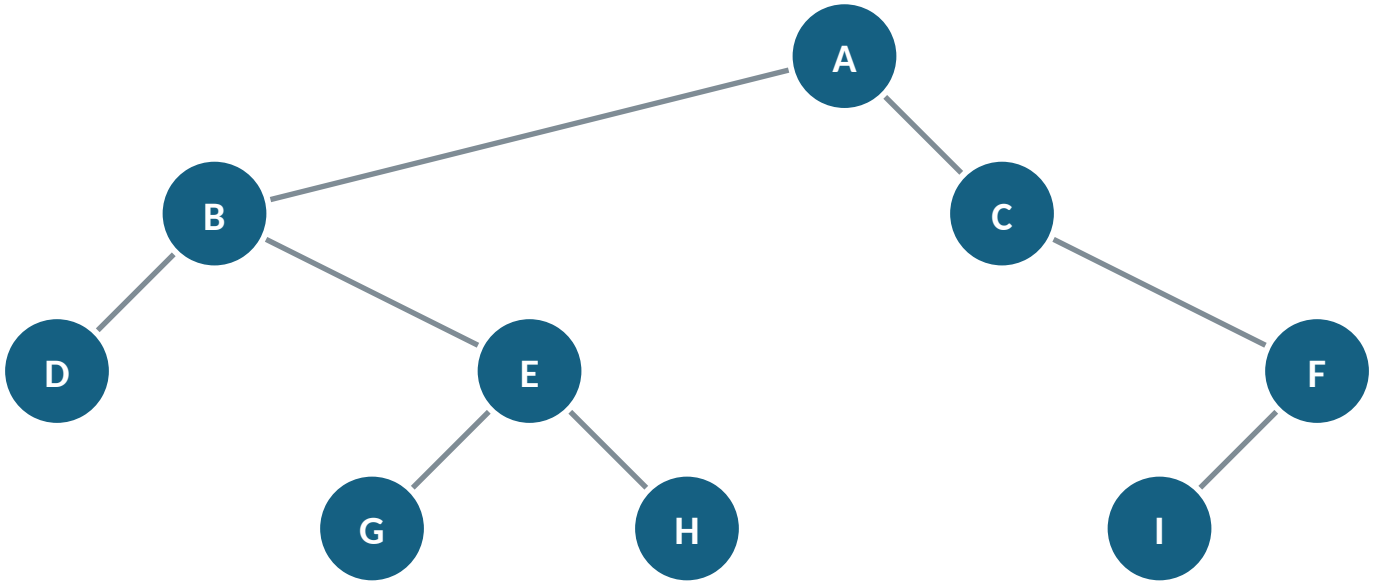
postorder: the LEFT subtree, the RIGHT subtree, then the NODE last

1. D is a leaf: print D
2. G is a leaf: print G
3. H is a leaf: print H
4. both sides of E are finished: print E
5. both sides of B are finished: print B
6. I is a leaf: print I
7. both sides of F are finished: print F
8. both sides of C are finished: print C
9. both sides of A are finished: print A

? **Think:** Which traversal can safely delete a tree, freeing a node only after its children? Why does preorder fail?

Answer: Postorder: children are freed before their parent. Preorder frees the parent first and loses the links to its children.

The Finger Trick: Walk Around the Tree



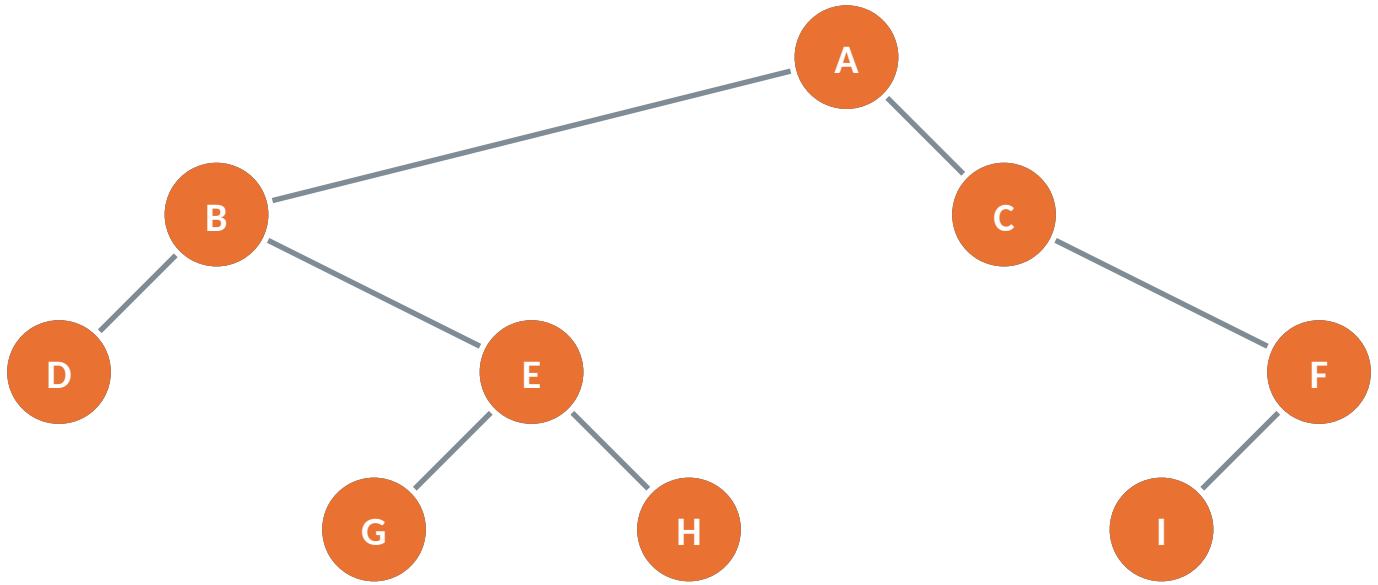
node	pre	in	post
A	1	6	9
B	2	2	5
D	3	1	1
E	4	4	4
I	9	8	6

Method: trace the outline counterclockwise. Write the node at its LEFT side for preorder, UNDERNEATH for inorder, RIGHT side for postorder.

? **Think:** Use the trick to find the inorder position of H and the postorder position of F.

Answer: Inorder D B G E H A C I F: H is 5th. Postorder D G H E B I F C A: F is 7th.

Level Order: Across the Tree with a Queue



take	add	queue: front → back
A	B,C	B C
B	D,E	C D E
C	F	D E F
D	none	E F
E	G,H	F G H
F	I	G H I
G	none	H I
H	none	I
I	none	

output: A B C D E F G H I

Think: How many nodes can be waiting in the queue at the same moment during a level-order walk of a perfect tree with 15 nodes?

Answer: Eight: once level 2 is processed, the whole bottom level of 8 leaves is waiting. A queue may hold about half the nodes; recursion holds only height-many.

Recursion on Trees: Count, Height, Leaves

IIT Madras Zanzibar

```
 1  def count(node):
 2      if node is None: return 0
 3      return 1 + count(node.left) + count(node.right)
 4  def height(node):
 5      if node is None: return -1
 6      return 1 + max(height(node.left), height(node.right))
 7  def leaves(node):
 8      if node is None: return 0
 9      if node.left is None and node.right is None: return 1
10      return leaves(node.left) + leaves(node.right)
```

Line by line

Line 2–3 count: 0 if empty, else 1 + left + right. Our tree: 9

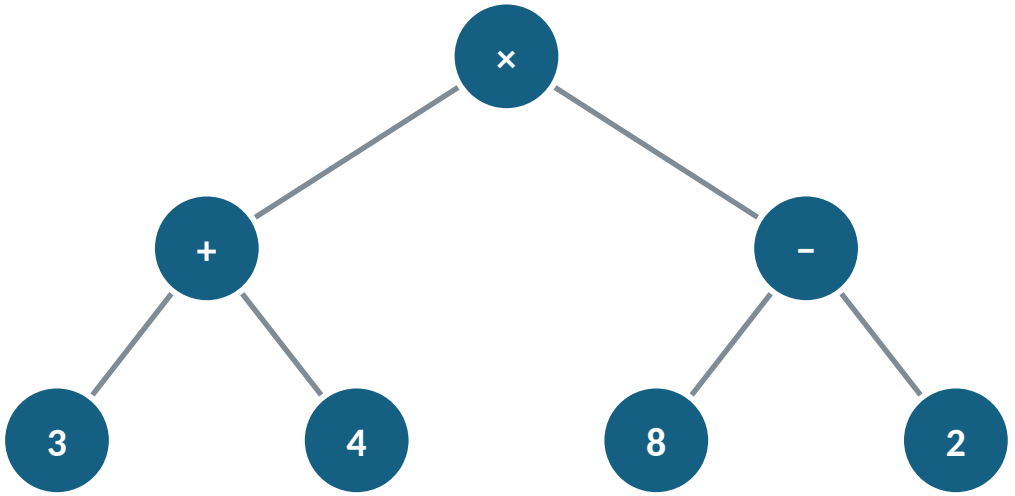
Line 5–6 height: −1 if empty, else 1 + the larger height. Our tree: 3

Line 8–10 leaves: 0 if empty, 1 if no children, else add both sides. Our tree: 4

? **Think:** Write total(node) that adds up all values in a tree of numbers. What does it return for the empty tree?

Answer: if node is None: return 0; else return node.value + total(node.left) + total(node.right). Empty tree: 0.

An Expression Tree: Postorder Evaluates It



prefix: `× + 3 4 − 8 2`
infix: `3 + 4 × 8 − 2` `(brackets lost)`
postfix: `3 4 + 8 2 − ×`

token → stack

```
3  →  [3]
4  →  [3, 4]
+  →  [7]
8  →  [7, 8]
2  →  [7, 8, 2]
−  →  [7, 6]
×  →  [42]
```

? **Think:** Give the postfix form of 2 × (3 + 4) − 5 and evaluate it with a stack.

Answer: 2 3 4 + × 5 −. Stack: 2; 2 3; 2 3 4; 2 7; 14; 14 5; 9. The value is 9.

Rebuilding a Tree from Two Traversals

IIT Madras Zanzibar

preorder: A B D E G H C F I **inorder:** D B G E H A C I F

PREORDER gives the ROOT: its first item.

INORDER gives the SPLIT: left of the root = left subtree.

subtree	1. preorder: first item = root	2. inorder: split at the root	3. sizes, then the next preorder items
whole tree	[A] B D E G H C F I	D B G E H | [A] | C I F	left 5 items, right 3: preorder after A: B D E G H go left, C F I go right
left of A	[B] D E G H	D | [B] | G E H	left 1 item, right 3: preorder after B: D goes left, E G H go right
right of B	[E] G H	G | [E] | H	left 1, right 1: G goes left, H goes right
right of A	[C] F I	(empty) | [C] | I F	no left child; right has 2: preorder F I: root F, inorder I | F: I is its left child

? **Think:** Can the preorder and postorder alone rebuild a tree? Try preorder A B with postorder B A.

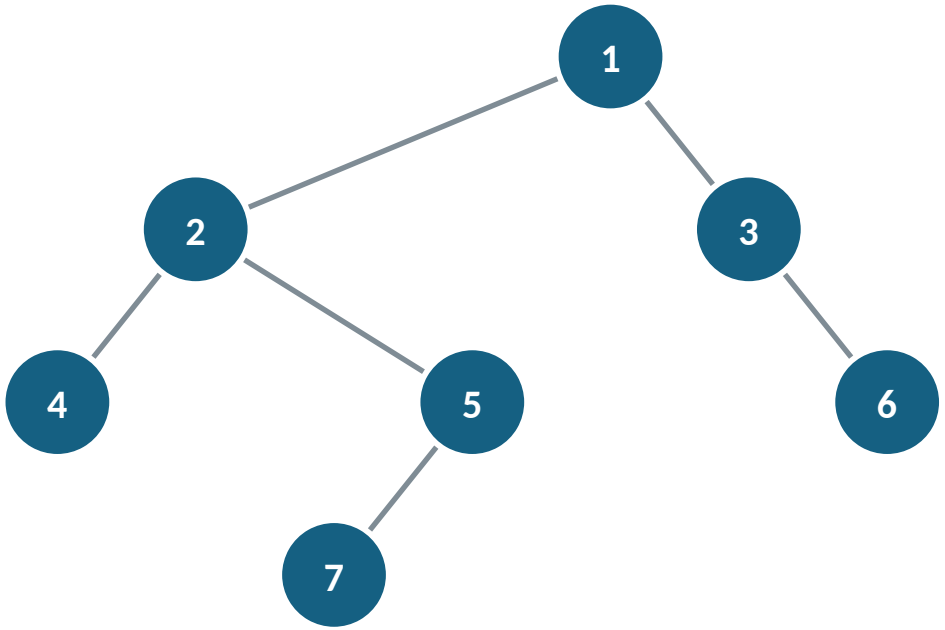
Answer: No: A with left child B and A with right child B both give pre A B and post B A. Inorder separates left from right.



In-Class Exercise (6 minutes, in pairs)

IIT Madras Zanzibar

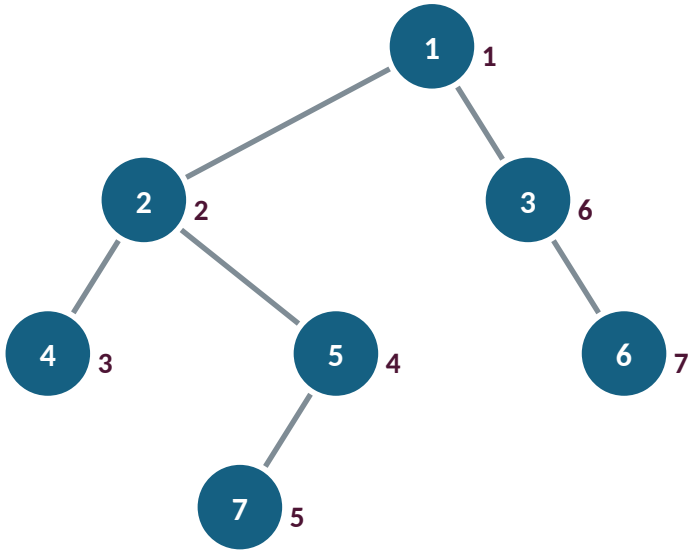
IN-CLASS TASK: traverse, rebuild, evaluate, count



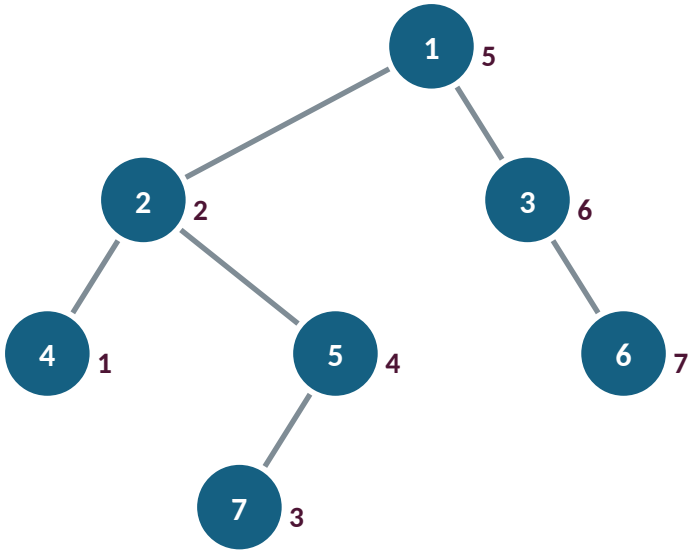
Four tasks (paper)

1. Tree on the left: write preorder, inorder, postorder and level order. Then count the nodes and leaves and find the height.
2. Rebuild the tree with preorder A B D E C F and inorder D B E A F C. Write its postorder.
3. Write the postfix form of (2 + 3) × 4 − 6 and evaluate it with a stack.
4. A binary tree has 12 leaves and 3 nodes with exactly one child. How many nodes does it have?

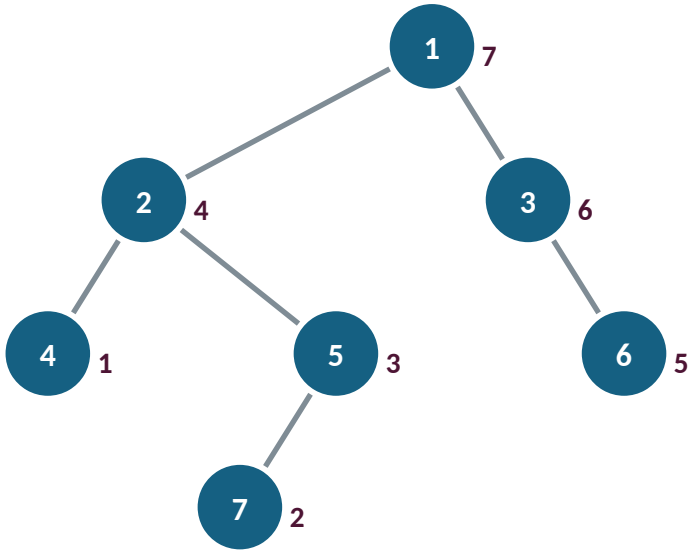
Solution 1: the Four Traversals



preorder: 1 2 4 5 7 3 6



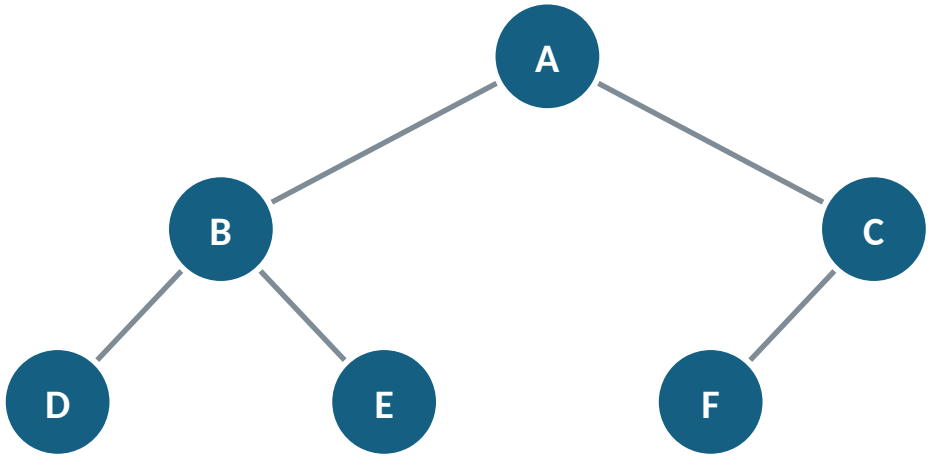
inorder: 4 2 7 5 1 3 6



postorder: 4 7 5 2 6 3 1

Also: level order 1 2 3 4 5 6 7; 7 nodes; leaves 4, 7, 6; height 3 (path 1, 2, 5, 7).

Solution 2, 3 and 4: Rebuild, Evaluate, Count

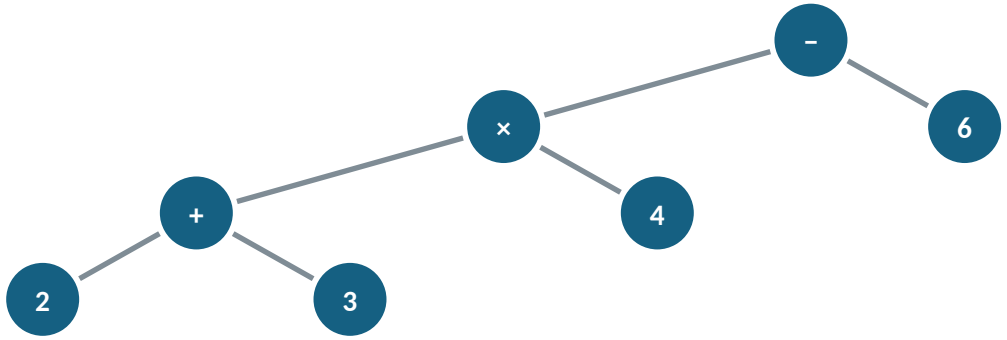


rebuilt tree; postorder: D E B F C A

round	preorder part	inorder part	root
1	A B D E C F	D B E A F C	A
2 (left)	B D E	D B E	B
3 (right)	C F	F C	C

`stack: [2] [2,3] [5] [5,4] [20] [20,6] [14]`

(2 + 3) × 4 − 6: postfix 2 3 + 4 × 6 − = 14



Task 4: leaves = two-child nodes + 1, so 11 two-child nodes. Total = 12 + 3 + 11 = 26.

Summary and Exam Checkpoints

IIT Madras Zanzibar

Why trees

data that branches: file systems, web pages, expressions, databases, heaps, search

Properties

level L holds at most 2^L; height h holds at most 2^(h+1) − 1; leaves = two-child nodes + 1

Types

full, perfect, complete, degenerate; balanced keeps the height near log n

Traversals

preorder, inorder, postorder differ only in the print line; level order uses a queue; all O(n)

Next

Thu 19 Nov: binary search trees: search and insert; Lab 5 on sorting and trees

Fri 20 Nov: BST delete and balance

Next week: heaps and priority queues

References and Further Reading

[1] Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 8.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Section 10.3; Chapter 12.

[3] Goodrich, M. T., Tamassia, R. & Goldwasser, M. H. (2013). Data Structures and Algorithms in Python. Wiley. Chapter 8.