Binary Trees I

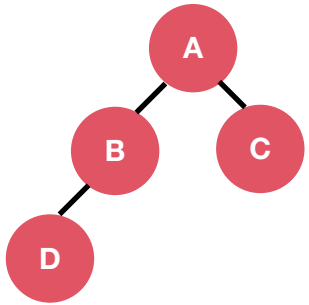
AGENDA & ANNOUNCEMENTS

- Binary Trees (45 mins)
- Breakout sessions (60 mins)
- Breakout session recap, questions (15 mins)
- ***Please post questions in Slack channel se103-s1-help-jun21***

Binary Trees

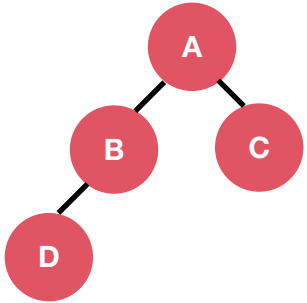
BINARY TREES

- Comprised of nodes
- Holds a value (usually a string or number)
- Each node can have [0, 2] children
- A node with no children is called a *leaf*
- Can have different properties (BST, balanced, complete, perfect etc.)



BINARY TREES CLASS DEFINITION

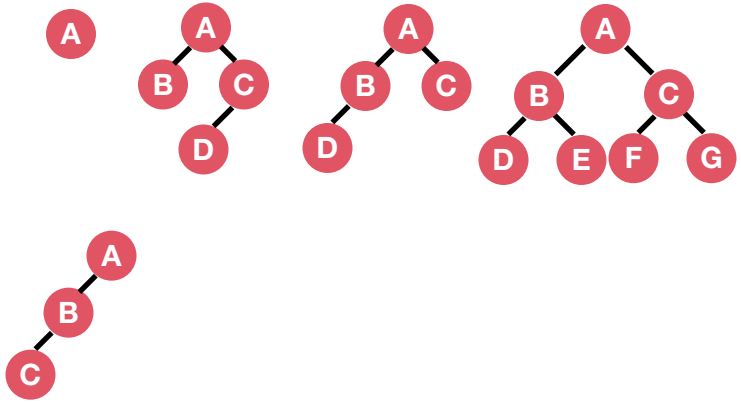
- Initialize a node with a value
- Optional left and right properties
- This is usually how a binary tree is represented in interviews




```
1  class BinaryTreeNode:
2      def __init__(self, value):
3          self.value = value
4          self.left = None
5          self.right = None
6  a = BinaryTreeNode('a')
7  b = BinaryTreeNode('b')
8  c = BinaryTreeNode('c')
9  d = BinaryTreeNode('d')
10 a.left = b
11 a.right = c
12 b.left = d
```

BINARY TREES

- The appearance of a binary tree can look many different ways
- Nodes can also have duplicate or negative values
- Realizing this is helpful when creating test cases!



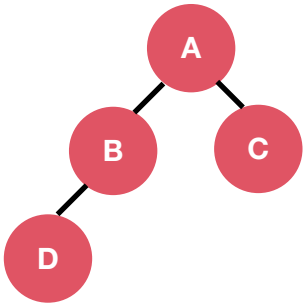
Tree Traversals

TREE TRAVERSALS

- Binary Tree problems require you to traverse the tree and/or manipulate/get values from it
- Two ways to traverse:
  - Depth-first search (DFS)
    - In-order, Pre-order, Post-order
  - Breadth-first search (BFS)
    - Level-order

DEPTH-FIRST SEARCH (DFS)

- In-order (left, current, right)
  - *Hint: In-order = Left to right*
  - *This traversal in a BST gives you the sorted order*
  - Reverse In-order (right, current, left)
- Pre-order (current, left, right)
  - *Hint: Pre = Before*
  - Reverse Pre-order (current, right, left)
- Post-order (left, right, current)
  - *Hint: Post = After*
  - Reverse Post-order (right, left, current)
- *Keywords: max, deepest, longest*



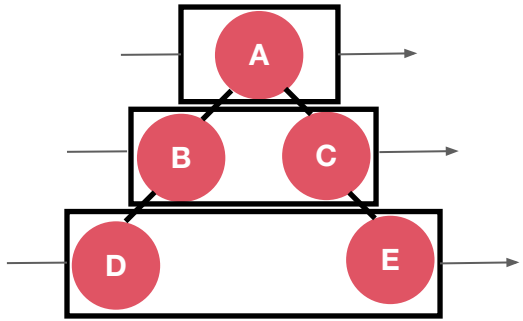
In-order: [D, B, A, C]
Pre-order: [A, B, D, C]
Post-order: [D, B, C, A]

IMPLEMENTING DFS

- Can be implemented iteratively or recursively
- At the very minimum, be able to do pre/post/in-order traversals
- Implement In-order Traversal
- Implement Preorder Traversal
- Implement Postorder Traversal

BREADTH-FIRST SEARCH (BFS)

- Traverse the tree in a level-order fashion
- Can be implemented using a queue
- *Keywords: level, row, closest, minimum, width, diameter*



Level-order: [A, B, C, D, E]

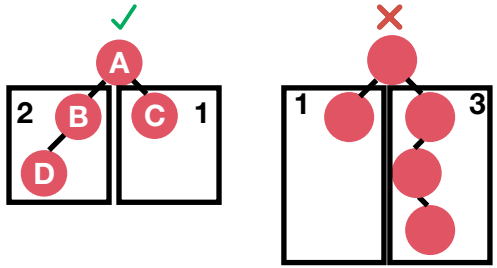
Different Properties of Binary Trees

DIFFERENT PROPERTIES OF BINARY TREES

- Most common:
  - Balanced/Non-balanced
  - Binary Search Tree (BST)
- Not as common:
  - Full, Perfect, Complete, Degenerate etc.
- Binary Trees can have multiple properties
  - Balanced BST, Non-balanced BST, Complete BST etc.
- *Hint: Tree properties are usually hints! Always ask your interviewer if a tree has special properties*

BALANCED BINARY TREES

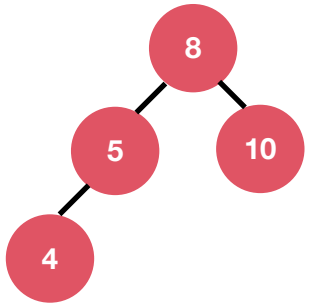
- Height of left and right subtree of every node differs at most by 1
- Height = The max distance of any node from the root
- *Hint: Balanced/non-balanced trees impact the runtime complexity for trees, especially BSTs*



https://xlinux.nist.gov/dads/HTML/height.html

BINARY SEARCH TREES (BST)

- Left subtree contains children <= root
- Right subtree contains children > root
- Better lookup and insert performance than non-BSTs
  - *Caveat: only if they are balanced*
- *Hint: The properties of a BST are usually needed to get the optimal solution*



COOL PROPERTIES OF BSTs

- A ***balanced*** BST has O(logn) performance for get, insert, delete operations
- Traversing a BST using an in-order traversal gets the ascending order. How would you get the descending order?

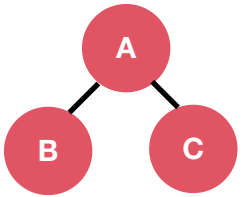
TIME/SPACE COMPLEXITY

- Time complexities usually differ depending on:
  - If the tree is a BST or not
  - If the tree is balanced/non-balanced
- **Remember your time and space complexities, interviewers will ask!**
- Always preface your assumptions
  - *"If this BST was balanced/non-balanced, the runtime would be..."*

	Non-balanced BST	Balanced BST
Get	O(n)	O(logn)
Insert	O(n)	O(logn)
Remove	O(n)	O(logn)

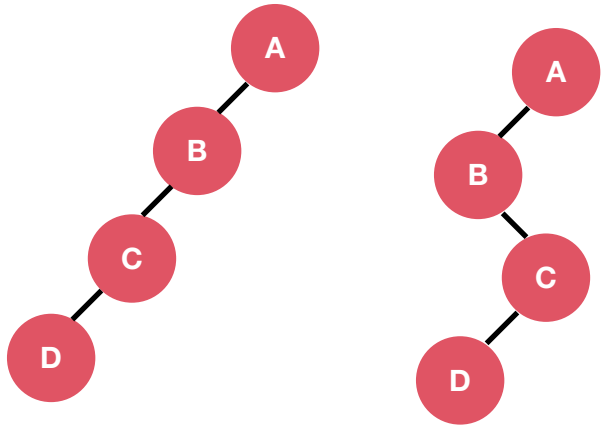
PERFECT BINARY TREES

- Every level is completely filled
- Every node has either 2 children or none
- A perfect binary tree is also balanced



DEGENERATE BINARY TREES

- Also called a *pathological* tree
- Every node has at most 1 child
- This is usually why runtime/space complexity is O(n) for trees (especially BSTs)



REVERSE LEVEL ORDER TRAVERSAL

- Leetcode Link

Breakout Sessions

BREAKOUT SESSION RECAP

- Min-Depth of Binary Tree
  - DFS vs. BFS
- Binary Tree Pruning
  - Recursive post-order traversal

See you on Saturday!

Please take < 3 mins to complete our feedback survey here.

This helps us improve the class for you!