Welcome to Leetcode Study Group!

Before we begin...

- Session materials: https://github.com/WomenWhoCode/WWCodePython

- Set your chat to “All panelists and attendees” and share your thoughts there

- Ask any questions using the Q&A button

- Have fun and make some coding friends!

WOMEN WHO CODE PYTHON

WELCOME

WOMEN WHO CODE

Our Mission

Inspiring women to excel in technology careers.

WOMEN WHO CODE PYTHON

Our Vision

A world where diverse women are better represented as engineers and tech leaders

WOMEN WHO CODE PYTHON

Our Values

- + Focus on the mission
- + Live Leadership
- + Punch above your weight
- + Inclusion at the core

WOMEN WHO CODE PYTHON

Our Target

Engineers with two or more years of experience looking for support and resources to strengthen their influence and levelup in their careers.

WOMEN WHO CODE PYTHON

290,000

Members

70 networks in 20 countries
122+ countries
14K+ events
$1025 daily Conference tickets
$2M Scholarships
Access to jobs + resources
Infinite connections

WOMEN WHO CODE PYTHON

OUR MOVEMENT

As the world changes, we can be a connecting force that creates a sense of belonging while the world is being asked to isolate.

WOMEN WHO CODE PYTHON

Code of Conduct

WWCode is an inclusive community, dedicated to providing an empowering experience for everyone who participates in or supports our community, regardless of gender, gender identity and expression, sexual orientation, ability, physical appearance, body size, race, ethnicity, age, religion, socioeconomic status, caste, creed, political affiliation, or preferred programming language(s).

Our events are intended to inspire women to excel in technology careers, and anyone who is there for this purpose is welcome. We do not tolerate harassment of members in any form. Our Code of Conduct applies to all WWCode events and online communities.

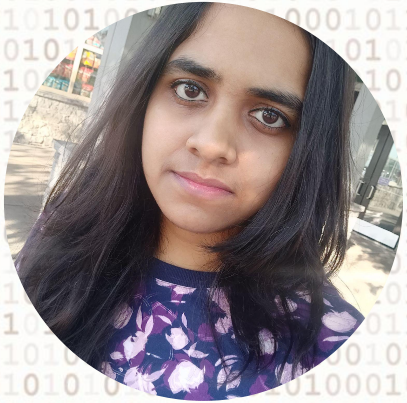
Read the full version and access our incident report form at womenwhocode.com/codeofconduct

WOMEN WHO CODE PYTHON

LeetCode Study Group

WOMEN WHO CODE PYTHON

Meet Your Team!



Chethana

Lead / Associate Software Engineer

Karen

Lead / Programmer

Today's Agenda

1. What have we seen so far?
2. Iterative tree traversal
   a. Intro
   b. Problem: Validate BST
   c. Problem: Binary Tree Right Side View
3. Fast and slow pointers
   a. Intro
   b. Problem & quick live coding: Linked List Cycle
4. Q&A

So far?

Quick recap

Our journey so far...

S1 - Two pointers

S2 - Sliding Window

S3 - Binary Search

S4 - Greedy approach

S5 - Hash tables

S6 - DFS & BFS

S7 - Backtracking

(Videos can be found in this playlist)

Bonus patterns #1

Iterative Tree Traversal

What?

- **DFS** being done iteratively using a **stack**
- **BFS** done using a **queue**

Why?

- Recursion shrouds some logic that we can see easily “see” using iteration
- “Seeing” how DFS and BFS is done (with a stack and queue respectively) unlocks newer thinking patterns

Problem: Validate BST

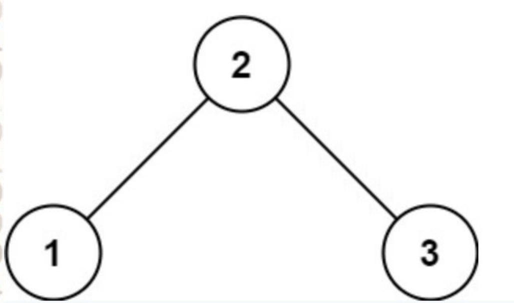
[Link to problem]

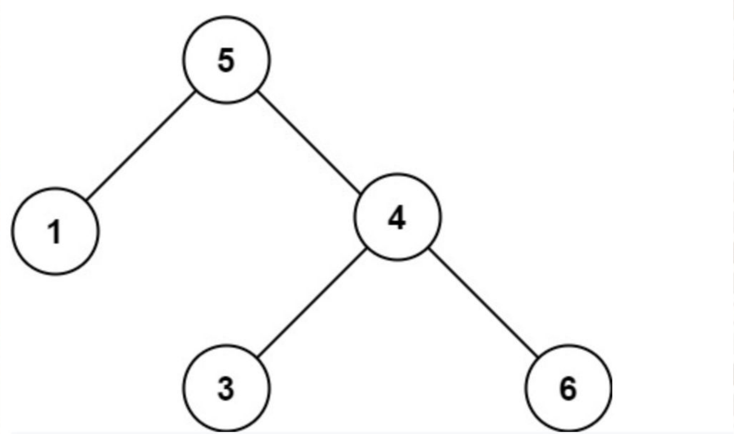
*"Given the **root** of a binary tree, determine if it is a valid binary search tree (BST).*

*A **valid BST** is defined as follows:*

- *The left subtree of a node contains only nodes with keys **less than** the node's key.*
- *The right subtree of a node contains only nodes with keys **greater than** the node's key.*
- *Both the left and right subtrees must also be binary search trees."*

Examples





Which one is a valid BST?

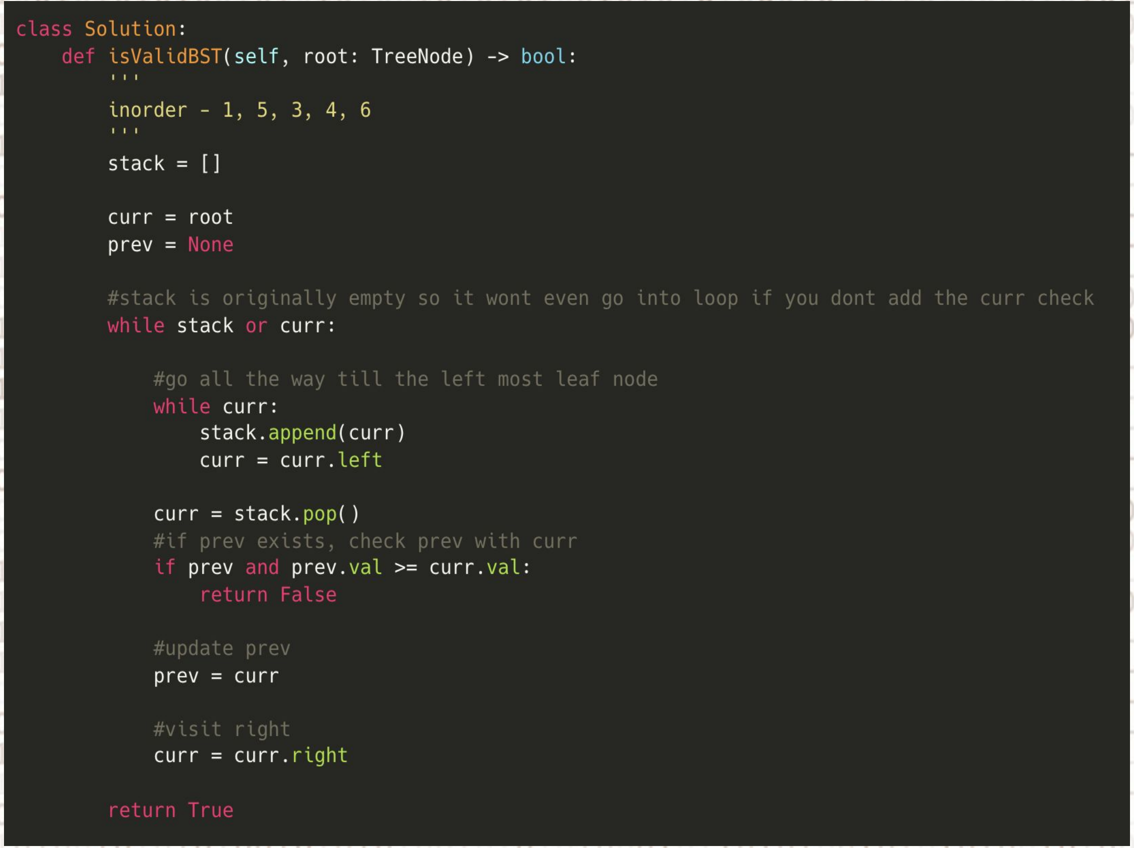
How can we “see” this?

Let’s try doing an **inorder** traversal on both (Left -> Node -> Right)

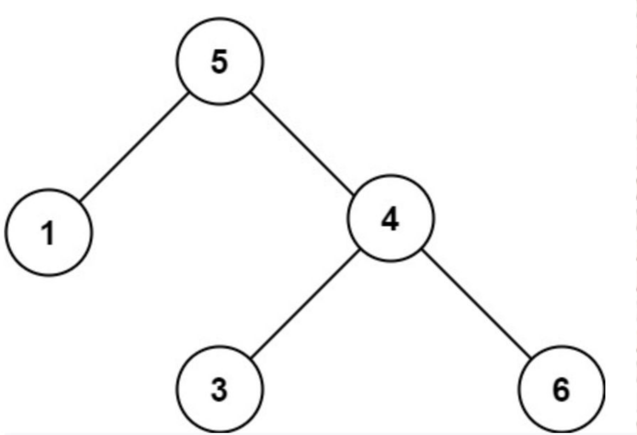
#1 => [1, 2, 3]
#2 => [1, 5, 3, 4, 6]

What difference do you notice in these traversals?

Approach: Validate BST - Iterative DFS



```
class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        '''
        inorder - 1, 5, 3, 4, 6
        '''
        stack = []

        curr = root
        prev = None

        #stack is originally empty so it wont even go into loop if you dont add the curr check
        while stack or curr:

            #go all the way till the left most leaf node
            while curr:
                stack.append(curr)
                curr = curr.left

            curr = stack.pop()
            #if prev exists, check prev with curr
            if prev and prev.val >= curr.val:
                return False

            #update prev
            prev = curr

            #visit right
            curr = curr.right

        return True
```

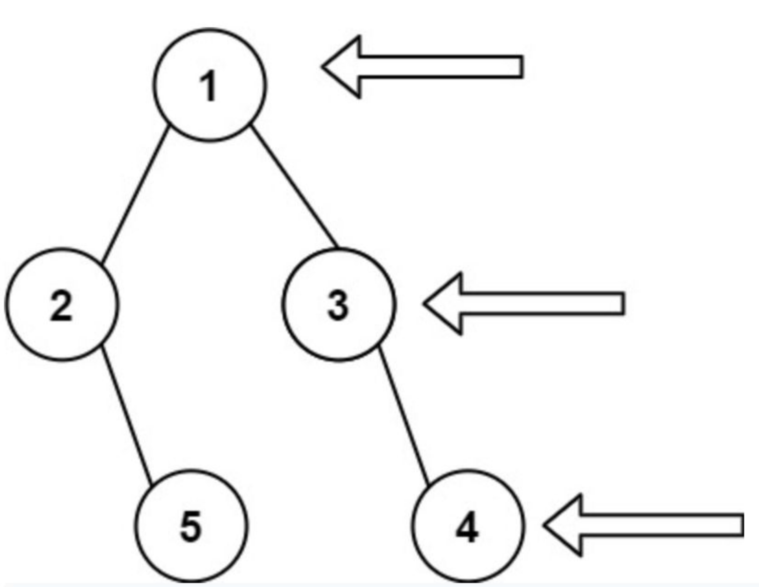


Problem: Binary Tree Right Side View

Link to problem

*“Given the **root** of a binary tree, imagine yourself standing on the **right side of it**, return the values of the nodes you can see ordered from **top to bottom**.”*

Example and approach discussion



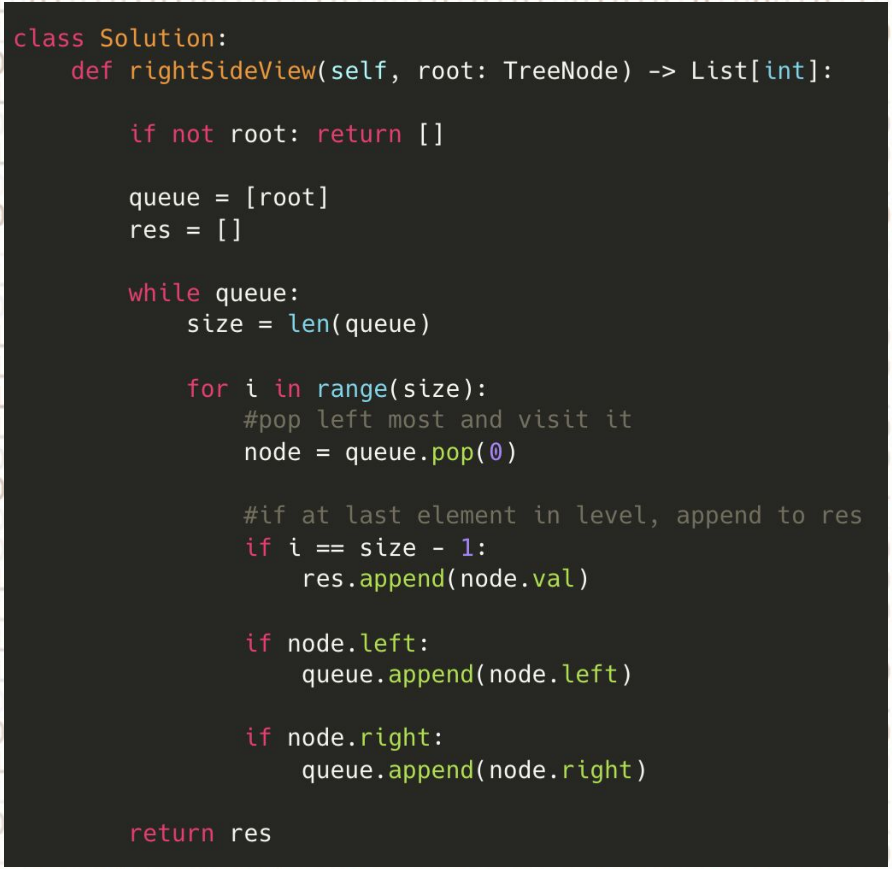
Output? [1, 3, 4]

How to do this?

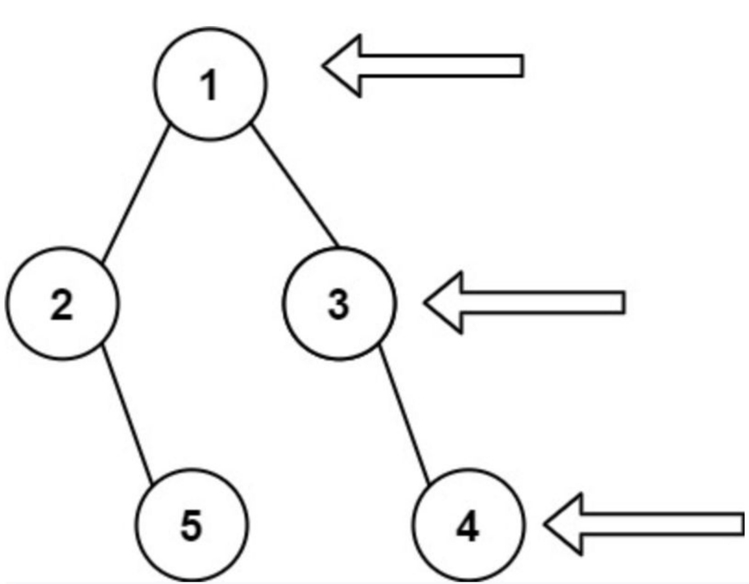
Look at levels,
Level #1 => [1]
Level #2 => [2, 3]
Level #3 => [5, 4]

Add each level to queue, pop left and if at the last element in that level, append it to result

Approach snapshot



```python
class Solution:
    def rightSideView(self, root: TreeNode) -> List[int]:

        if not root: return []

        queue = [root]
        res = []

        while queue:
            size = len(queue)

            for i in range(size):
                #pop left most and visit it
                node = queue.pop(0)

                #if at last element in level, append to res
                if i == size - 1:
                    res.append(node.val)

                if node.left:
                    queue.append(node.left)

                if node.right:
                    queue.append(node.right)

        return res
```



Bonus patterns #2

Fast and slow pointers

The idea

- There are 2 pointers => `slow`, `fast`
- Both start at the same position
- They move at different speeds (`fast` may move 1 or 2 steps faster than `slow`)
- Naturally there would be a point where they meet
- At this point, based on our problem - we can discover a cycle, duplicate element, etc
- More generally, when you need to traverse a data structure and have a way of looping through it - to find an intersection

Problem: Linked List Cycle

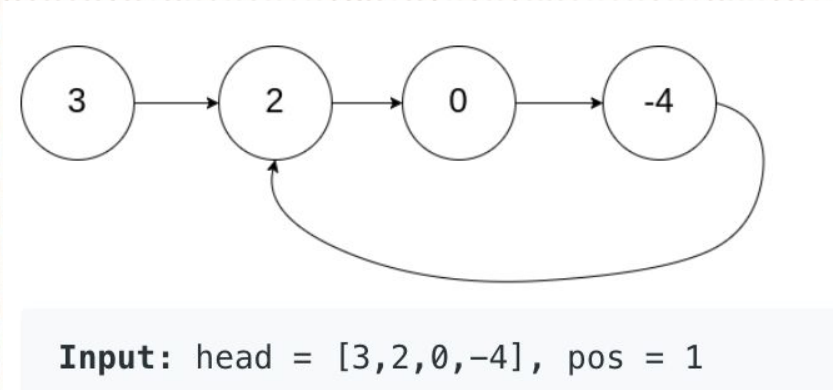
Link to problem

*"Given **head**, the head of a linked list, determine if the linked list has a cycle in it.*

*There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, **pos** is used to denote the index of the node that **tail**'s next pointer is connected to. Note that **pos** is not passed as a parameter.*

*Return **true** if there is a cycle in the linked list. Otherwise, return **false**"*

Simplifying that with example



Input: `head = [3,2,0,-4], pos = 1`

- Given a linked list - `head`

- `pos` indicates where the `tail` is connected to (ie the cycle) but we aren't given that info

- if cycle return True, else False

- How to figure out if cycle?

Let’s Code!

https://leetcode.com/problems/linked-list-cycle/

QnA Time!

WOMEN WHO
CODE
PYTHON

Useful Links

- Leetcode Study group repo
- Repl link
- Mock interview - Pramp
- Leetcode Weekly contest (and biweekly)

Stay Connected!

WOMEN WHO CODE PYTHON

JOIN US ON SOCIAL MEDIA!

@WWCODEPYTHON

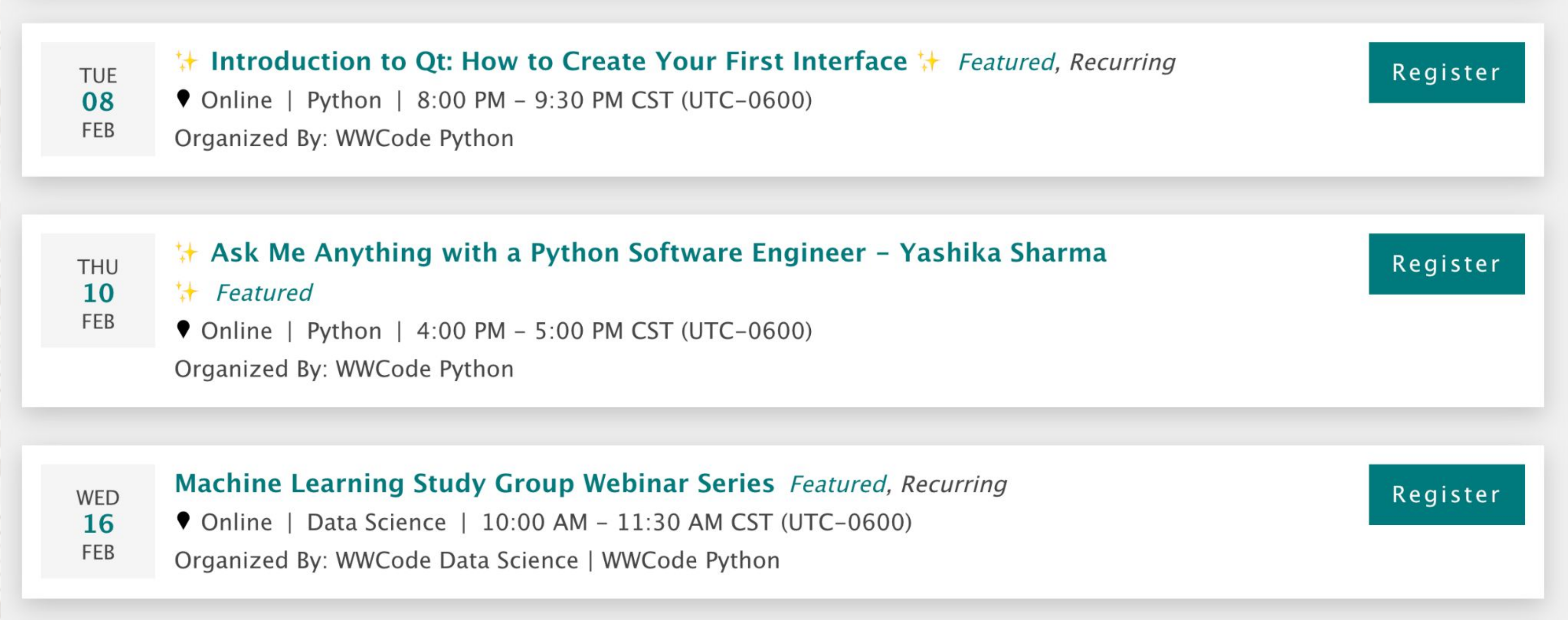
WOMENWHOCODE.COM/PYTHON

Last Session

➔ Feb 17 - Bonus session 2

(would be less of coding, more of sharing resources for certain algorithms)

Upcoming Events



TUE
08
FEB

✨ Introduction to Qt: How to Create Your First Interface ✨ *Featured, Recurring*

Online | Python | 8:00 PM – 9:30 PM CST (UTC-0600)

Organized By: WWCode Python

Register

THU
10
FEB

✨ Ask Me Anything with a Python Software Engineer – Yashika Sharma ✨ *Featured*

Online | Python | 4:00 PM – 5:00 PM CST (UTC-0600)

Organized By: WWCode Python

Register

WED
16
FEB

Machine Learning Study Group Webinar Series *Featured, Recurring*

Online | Data Science | 10:00 AM – 11:30 AM CST (UTC-0600)

Organized By: WWCode Data Science | WWCode Python

Register

Register at: https://www.womenwhocode.com/python/events

WOMEN WHO CODE PYTHON

Thank You for Joining!