Module5-HW

June 13, 2024

Problem1-LeetcodeQ704-Binary Search-Easy

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [-1,0,3,5,9,12], target = 9 Output: 4 Explanation: 9 exists in nums and its index is 4 Example 2:

Input: nums = [-1,0,3,5,9,12], target = 2 Output: -1 Explanation: 2 does not exist in nums so return -1

Constraints:

1 <= nums.length <= 104 -104 < nums[i], target < 104 All the integers in nums are unique. nums is sorted in ascending order.

0.0.1 pseudocode

[ ]:
```
Define 2 pointers
 left = 0
 right = length of nums - 1

while left <= right:
middle = (left + right) // 2

if nums[middle] < target:
    left = middle + 1
else if nums[middle] > target:
    right = middle - 1
else:
    return middle
```

0.0.2 Q704 code.py

[3]:
```
from typing import List

class Solution:
```

```python
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1

        while left <= right:
            middle = (left + right) // 2

            if nums[middle] < target:
                left = middle + 1
            elif nums[middle] > target:
                right = middle - 1
            else:
                return middle
        return -1

# Test case
nums = [-1, 0, 3, 5, 9, 12]
target = 9
solution = Solution()
output = solution.search(nums, target)
print(f'Input: nums = {nums}, target = {target}')
print(f'Output: {output}')
```

```
Input: nums = [-1, 0, 3, 5, 9, 12], target = 9
Output: 4
```

Problem2-LeetcodeQ74-Search a 2-D Matrix-Medium

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.

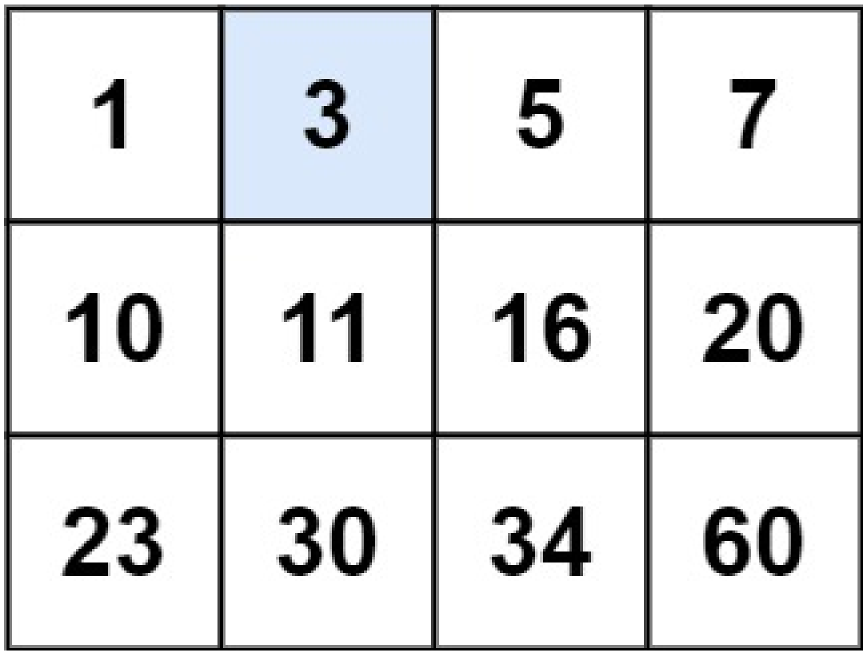
Given an integer target, return true if target is in matrix or false otherwise.

You must write a solution in O(log(m * n)) time complexity.

Constraints:

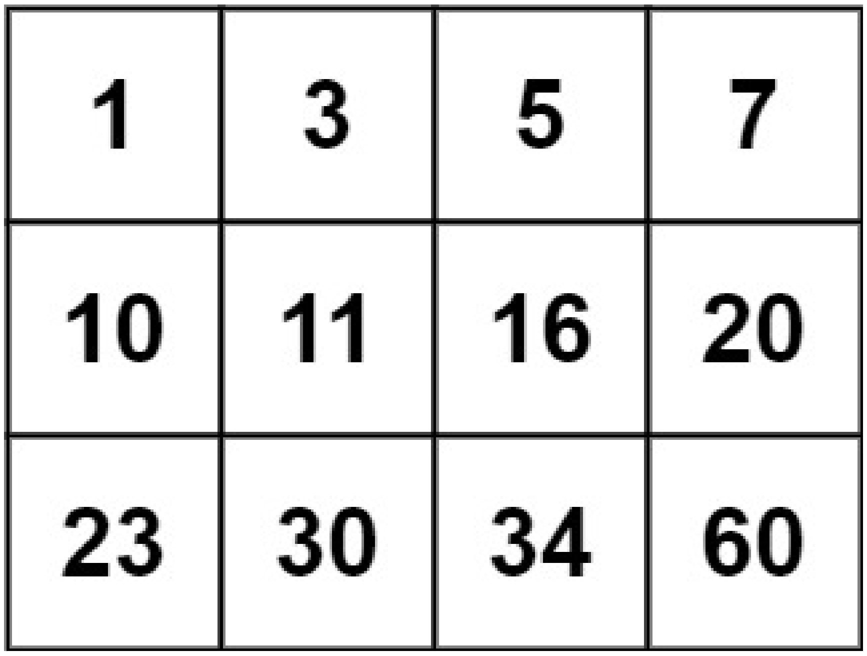
- m == matrix.length
- n == matrix[i].length

Example1: - Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3 - Output: true



1	3	5	7
10	11	16	20
23	30	34	60

- Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
- Output: false



1	3	5	7
10	11	16	20
23	30	34	60

0.0.3 pseudocode

[ ]:
```
initialize m and n to the number of rows and columns in the matrix
initialize left to -1 and right to the product of m and n
while left + 1 is less than right:
    calculate mid as the average of left and right
    set x to the element in the matrix at row mid // n and column mid % n
    if x is equal to target:
        return true
    if x is less than target:
        set left to mid
    else:
        set right to mid
return false
```

0.0.4 Q74 code

[6]:
```python
from typing import List

class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        m, n = len(matrix), len(matrix[0])
        left, right = -1, m * n
        while left + 1 < right:
            mid = (left + right) // 2
            x = matrix[mid // n][mid % n]
            if x == target:
                return True
            if x < target:
                left = mid
            else:
                right = mid
        return False

# Test case
matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]]
target = 3
solution = Solution()
output = solution.searchMatrix(matrix, target)
print(f'Input: matrix = {matrix}, target = {target}')
print(f'Output: {output}')
```

```
Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3
Output: True
```

Problem3-LeetcodeQ374. Guess Number Higher or Lower-Easy

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I will tell you whether the number I picked is higher or lower than your guess.

You call a pre-defined API int guess(int num), which returns three possible results:

- -1: Your guess is higher than the number I picked (i.e. num > pick).
- 1: Your guess is lower than the number I picked (i.e. num < pick).

- 0: your guess is equal to the number I picked (i.e. num == pick).
- Return the number that I picked.

Example 1:

- Input: n = 10, pick = 6
- Output: 6

Example 2:

- Input: n = 1, pick = 1
- Output: 1

Example 3:

- Input: n = 2, pick = 1
- Output: 1

0.0.5 pseudocode

[ ]:
```
initialize left to 1
initialize right to n
while left is less than or equal to right:
calculate mid as the average of left and right
call the guess function with mid and store the result in ret
if ret is equal to 0:
    return mid
if ret is equal to -1:
    set right to mid - 1
else:
    set left to mid + 1
```

0.0.6 code.py

[8]:
```python
def guess(num: int) -> int:
    pick = 6
    if num < pick:
        return 1
    elif num > pick:
        return -1
    else:
        return 0

from typing import List

class Solution:
    def guessNumber(self, n: int) -> int:
        left = 1
        right = n
        while left <= right:
            mid = (left + right) // 2
            ret = guess(mid)
            if ret == 0:
                return mid
            elif ret == -1:
                right = mid - 1
            else:
                left = mid + 1

# Test case
n = 10
solution = Solution()
output = solution.guessNumber(n)
print(f'Input: n = {n}, pick = 6')
print(f'Output: {output}')
```

```
Input: n = 10, pick = 6
Output: 6
```

Problem4-LeetcodeQ278. First Bad Version-Easy

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

- Input: n = 5, bad = 4
- Output: 4

Explanation: - call isBadVersion(3) -> false - call isBadVersion(5) -> true - call isBadVersion(4) -> true - Then 4 is the first bad version.

Example 2:

- Input: n = 1, bad = 1
- Output: 1

0.0.7 pseudocode

[ ]:
```
initialize i to 1
initialize j to n
while i is less than or equal to j:
calculate m as the average of i and j
if isBadVersion(m) is true:
    set j to m - 1
```

```
else:
    set i to m + 1
```

0.0.8 code.py

[11]:
```python
def isBadVersion(version: int) -> bool:
    bad = 4  # example value for testing
    return version >= bad

class Solution:
    def firstBadVersion(self, n: int) -> int:
        i, j = 1, n
        while i <= j:
            m = (i + j) // 2
            if isBadVersion(m):
                j = m - 1
            else:
                i = m + 1
        return i

# Test case
n = 5
solution = Solution()
output = solution.firstBadVersion(n)
print(f'Input: n = {n}, bad = 4')
print(f'Output: {output}')
```

```
Input: n = 5, bad = 4
Output: 4
```

Problem5-LeetcodeQ875. Koko Eating Bananas-Medium

Koko loves to eat bananas. There are n piles of bananas, the ith pile has piles[i] bananas. The guards have gone and will come back in h hours.

Koko can decide her bananas-per-hour eating speed of k. Each hour, she chooses some pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return the minimum integer k such that she can eat all the bananas within h hours.

Example 1:

- Input: piles = [3,6,7,11], h = 8
- Output: 4

Example 2:

- Input: piles = [30,11,23,4,20], h = 5
- Output: 30

Example 3:

- Input: piles = [30,11,23,4,20], h = 6
- Output: 23

0.0.9 pseudocode

[ ]:
```
initialize n to the length of piles
initialize left to 0
initialize right to the maximum value in piles
while left + 1 is less than right:
    calculate mid as the average of left and right
    if the sum of (p - 1) // mid for each p in piles is less than or equal to h␣
 ↪- n:
        set right to mid
    else:
        set left to mid
return right
```

0.0.10 code.py

[13]:
```python
from typing import List

class Solution:
    def minEatingSpeed(self, piles: List[int], h: int) -> int:
        n = len(piles)
        left = 0
        right = max(piles)
        while left + 1 < right:
            mid = (left + right) // 2
            if sum((p - 1) // mid for p in piles) <= h - n:
                right = mid
            else:
                left = mid
        return right

# Test case
piles = [30, 11, 23, 4, 20]
h = 5
solution = Solution()
output = solution.minEatingSpeed(piles, h)
print(f'Input: piles = {piles}, h = {h}')
print(f'Output: {output}')
```

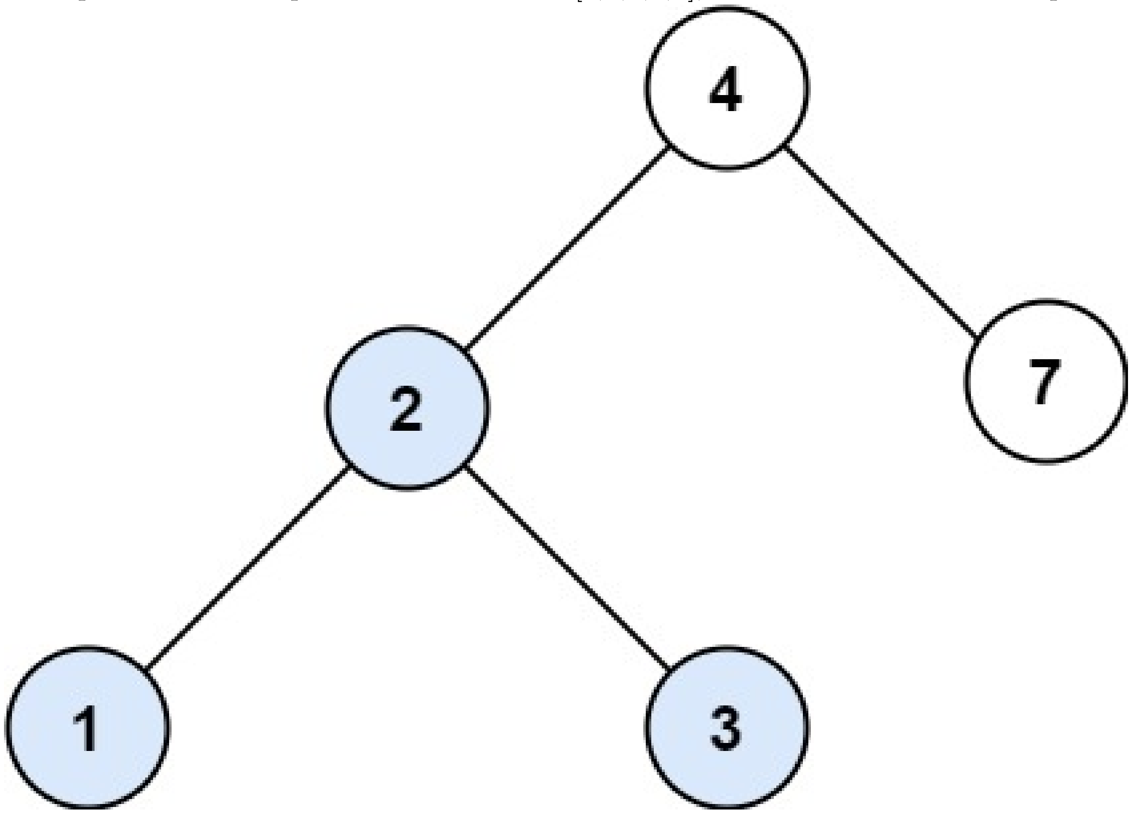
```
Input: piles = [30, 11, 23, 4, 20], h = 5
Output: 30
```

Problem6-LeetcodeQ700. Search in a Binary Search Tree-Easy

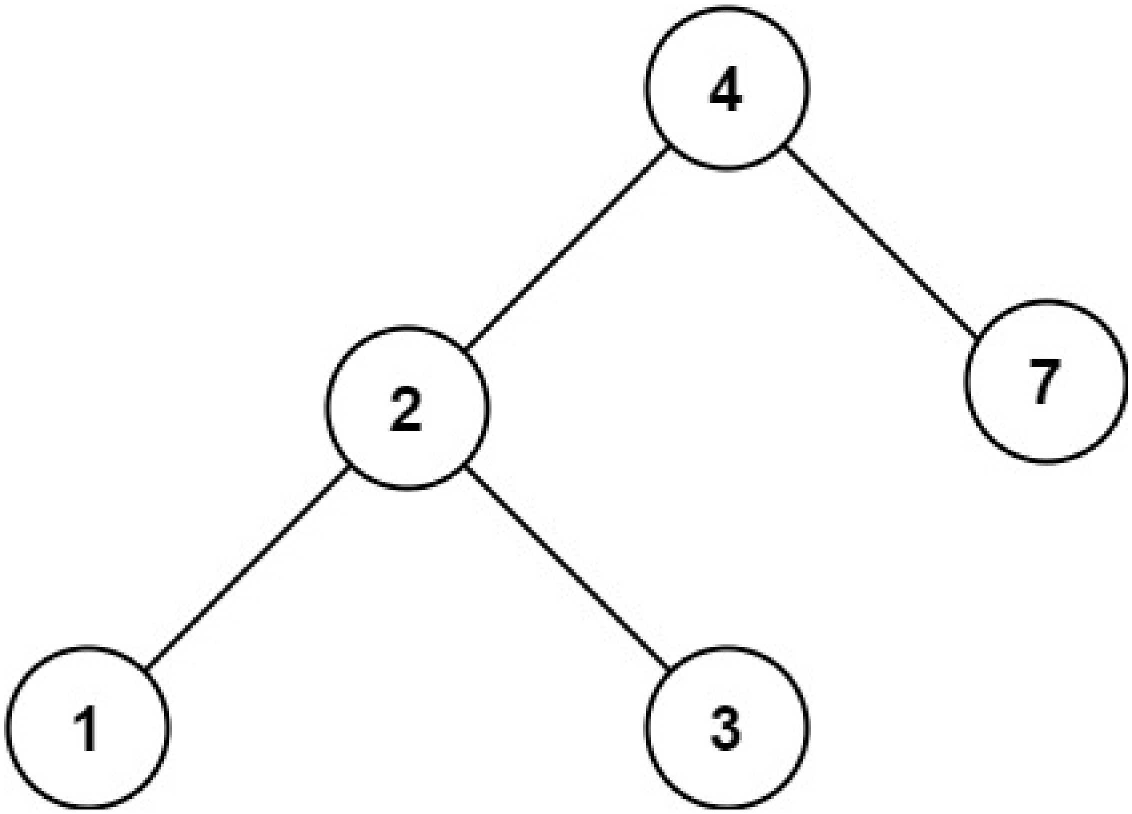
You are given the root of a binary search tree (BST) and an integer val.

Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null.

Example1: - Input: root = [4,2,7,1,3], val = 2 - Output: [2,1,3]



Example2: - Input: root = [4,2,7,1,3], val = 5 - Output: []



0.0.11 pseudocode

[ ]:
```
if root is null:
    return null
else:
    if root value is equal to val:
        return root
    else if root value is greater than val:
        return searchBST on root's left child with val
    else:
        return searchBST on root's right child with val
return null
```

0.0.12 code.py

[15]:
```python
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def searchBST(self, root: Optional[TreeNode], val: int) ->␣
↪Optional[TreeNode]:
        if not root:
            return None
        else:
            if root.val == val:
                return root
            elif root.val > val:
                return self.searchBST(root.left, val)
            else:
                return self.searchBST(root.right, val)
        return None


def build_tree(nodes, index=0):
    if index < len(nodes) and nodes[index] is not None:
        node = TreeNode(nodes[index])
        node.left = build_tree(nodes, 2 * index + 1)
        node.right = build_tree(nodes, 2 * index + 2)
        return node
    return None

def print_tree(node):
    if node:
        print_tree(node.left)
        print(node.val, end=' ')
        print_tree(node.right)

# Test case
root = build_tree([4, 2, 7, 1, 3])
val = 2
solution = Solution()
output = solution.searchBST(root, val)
print(f'Input: root = [4, 2, 7, 1, 3], val = {val}')
print(f'Output:', end=' ')
print_tree(output)
print()
```

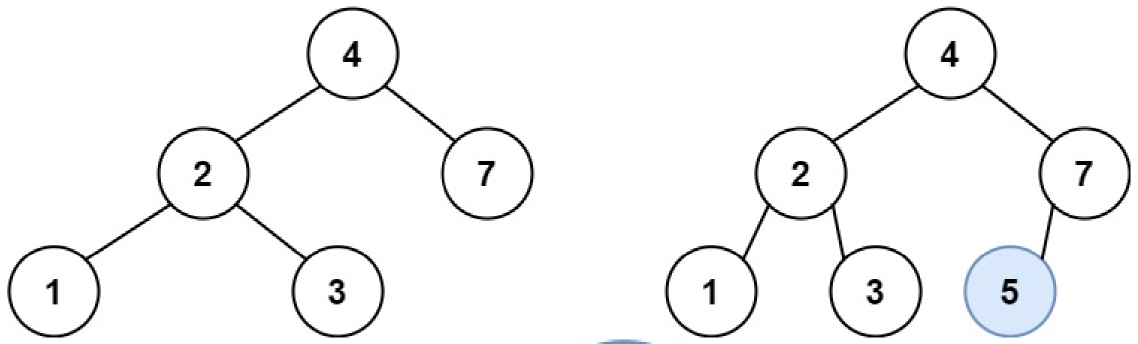
```
Input: root = [4, 2, 7, 1, 3], val = 2
Output: 1 2 3
```

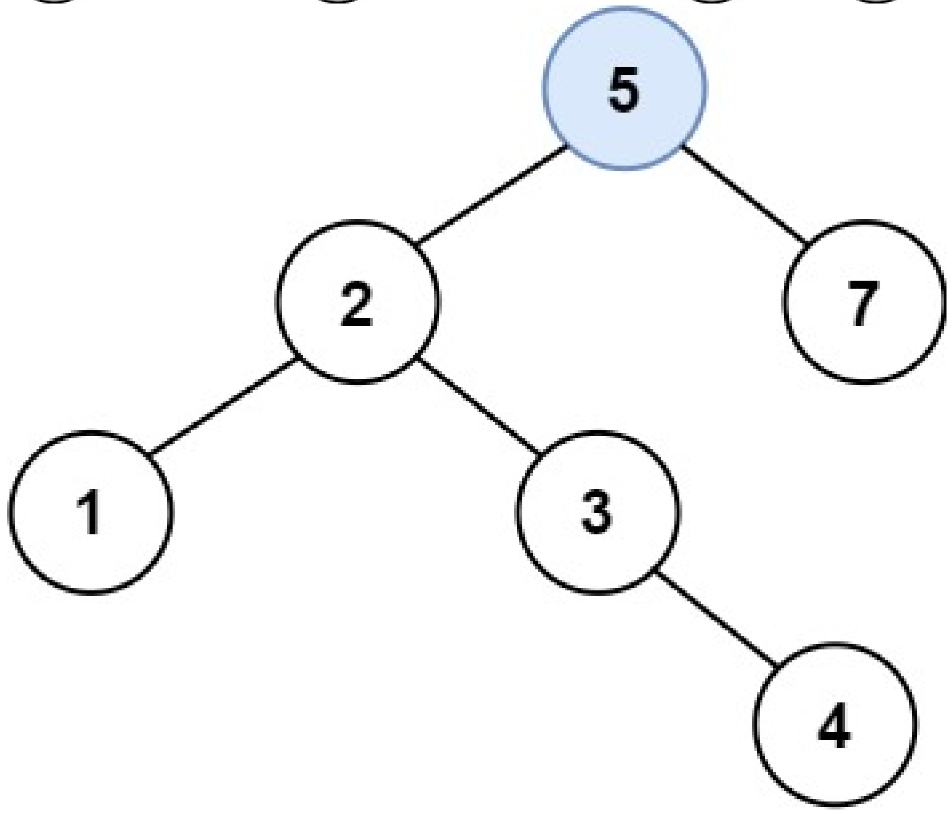
Problem7-LeetcodeQ701. Insert into a Binary Search Tree-Medium

You are given the root node of a binary search tree (BST) and a value to insert into the tree. Return the root node of the BST after the insertion. It is guaranteed that the new value does not exist in the original BST.

Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return any of them.

Example1: - Input: root = [4,2,7,1,3], val = 5 - Output: [4,2,7,1,3,5]





Example 2:

- Input: root = [40,20,60,10,30,50,70], val = 25
- Output: [40,20,60,10,30,50,70,null,null,25]

Example 3:

- Input: root = [4,2,7,1,3,null,null,null,null,null,null], val = 5
- Output: [4,2,7,1,3,5]

0.0.13 pseudocode

[ ]:
```
if root is null:
return new tree node with value val

define function find with parameters root, parent, and val:
if root is null:
    if val is less than parent's value:
    set parent's left child to new tree node with value val
    else:
    set parent's right child to new tree node with value val
    return
if val is less than root's value:
    call find with root's left child, root, and val
else:
    call find with root's right child, root, and val

call find with root, null, and val
return root
```

0.0.14 code.py

[17]:
```
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def insertIntoBST(self, root: Optional[TreeNode], val: int) -> TreeNode:
        if root is None:
            return TreeNode(val)

        def find(root: Optional[TreeNode], parent: Optional[TreeNode], val:␣
 ↪int):
            if root is None:
                if val < parent.val:
                    parent.left = TreeNode(val)
                else:
                    parent.right = TreeNode(val)
                return

            if val < root.val:
                find(root.left, root, val)
            else:
                find(root.right, root, val)

        find(root, None, val)
        return root

def build_tree(nodes, index=0):
    if index < len(nodes) and nodes[index] is not None:
        node = TreeNode(nodes[index])
        node.left = build_tree(nodes, 2 * index + 1)
        node.right = build_tree(nodes, 2 * index + 2)
        return node
    return None

def print_tree(node):
    if node:
        print_tree(node.left)
        print(node.val, end=' ')
        print_tree(node.right)

# Test case
root = build_tree([4, 2, 7, 1, 3])
val = 5
solution = Solution()
output = solution.insertIntoBST(root, val)
print(f'Input: root = [4, 2, 7, 1, 3], val = {val}')
print(f'Output:', end=' ')
print_tree(output)
print()
```

```
Input: root = [4, 2, 7, 1, 3], val = 5
Output: 1 2 3 4 5 7
```

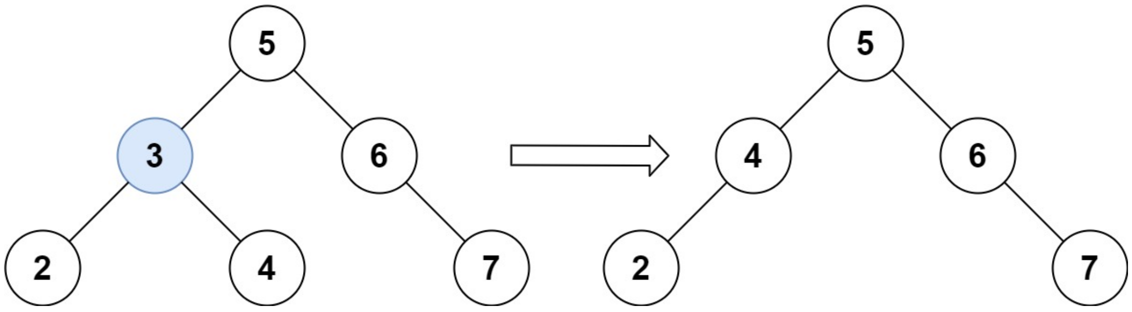
Problem8-LeetcodeQ450. Delete Node in a BST-Medium

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

- Search for a node to remove.
- If the node is found, delete the node.

Example 1:



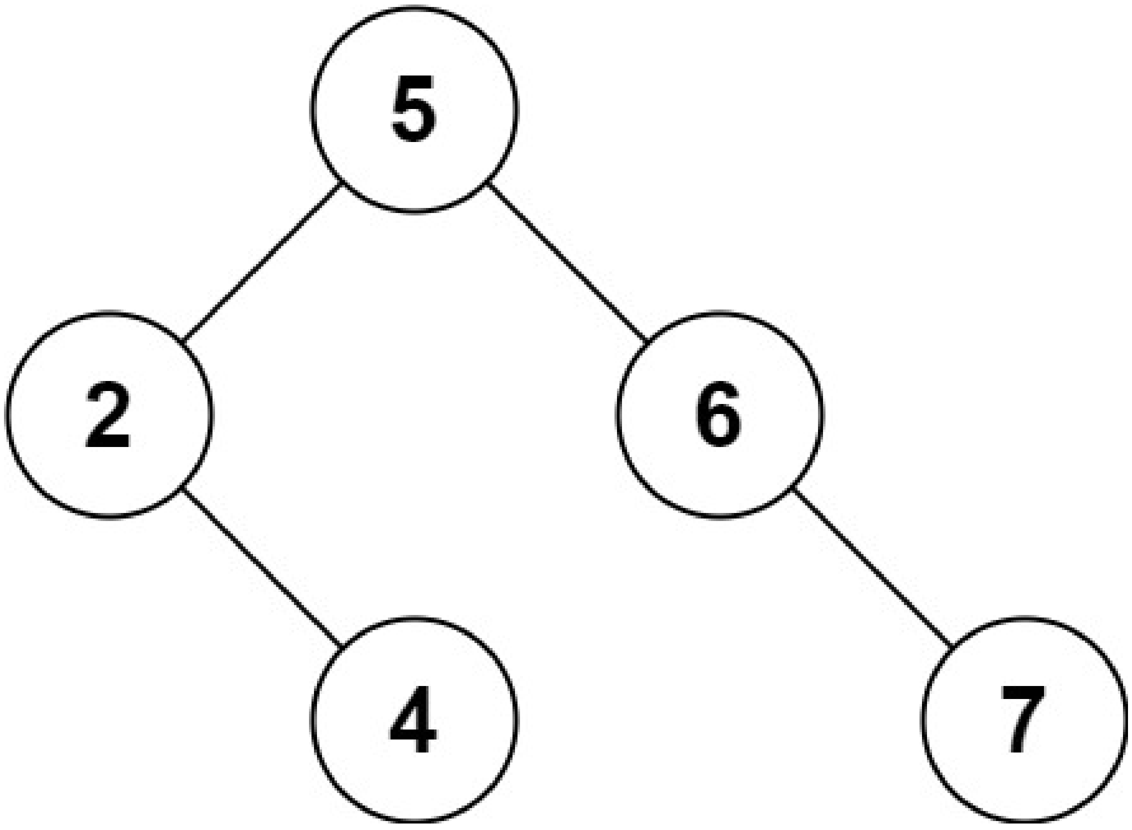
- Input: root = [5,3,6,2,4,null,7], key = 3
- Output: [5,4,6,2,null,null,7]

Explanation: Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the above BST.

Please notice that another valid answer is [5,2,6,null,4,null,7] and it's also accepted.

Example 2:



- Input: root = [5,3,6,2,4,null,7], key = 0
- Output: [5,3,6,2,4,null,7]

Explanation: The tree does not contain a node with value = 0.

Example 3:

- Input: root = [], key = 0
- Output: []

0.0.15 pseudocode

[ ]:
```
if root is not null:
if root value is less than key:
    set root's right child to the result of deleting the key from root's right␣
  ↪child
else if root value is greater than key:
    set root's left child to the result of deleting the key from root's left␣
  ↪child
else:
    if root has no left or right child:
        set root to root's left child if it exists, otherwise set to root's␣
  ↪right child
    else:
        initialize node to root's left child
        while node's right child exists:
            move to node's right child
        set node's left child to the result of deleting node's value from␣
  ↪root's left child
        set node's right child to root's right child
        set root to node
return root
```

0.0.16 code.py

[20]:
```python
from typing import Optional

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def deleteNode(self, root: Optional[TreeNode], key: int) ->␣
  ↪Optional[TreeNode]:
        if root:
            if root.val < key:
                root.right = self.deleteNode(root.right, key)
            elif root.val > key:
                root.left = self.deleteNode(root.left, key)
            else:
                if not root.left or not root.right:
                    root = root.left if root.left else root.right
                else:
                    node = root.left
                    while node.right:
                        node = node.right
                    node.left = self.deleteNode(root.left, node.val)
                    node.right = root.right
                    root = node
        return root


def build_tree(nodes, index=0):
    if index < len(nodes) and nodes[index] is not None:
        node = TreeNode(nodes[index])
        node.left = build_tree(nodes, 2 * index + 1)
        node.right = build_tree(nodes, 2 * index + 2)
        return node
    return None

def print_tree(node):
    if node:
        print_tree(node.left)
        print(node.val, end=' ')
        print_tree(node.right)

# Test case
root = build_tree([5, 3, 6, 2, 4, None, 7])
key = 3
solution = Solution()
output = solution.deleteNode(root, key)
print(f'Input: root = [5, 3, 6, 2, 4, null, 7], key = {key}')
print(f'Output:', end=' ')
print_tree(output)
print()
```

```
Input: root = [5, 3, 6, 2, 4, null, 7], key = 3
Output: 2 4 5 6 7
```

[ ]: