Previous Next

108. Convert Sorted Array to BST

Nov. 6, 2019 | 29K views

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1.

Example:

```
Given the sorted array: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height bal

      0
     / \
   -3   9
   /   /
 -10  5
```

Solution

How to Traverse the Tree. DFS: Preorder, Inorder, Postorder; BFS.

There are two general strategies to traverse a tree:

- *Depth First Search* (`DFS`)

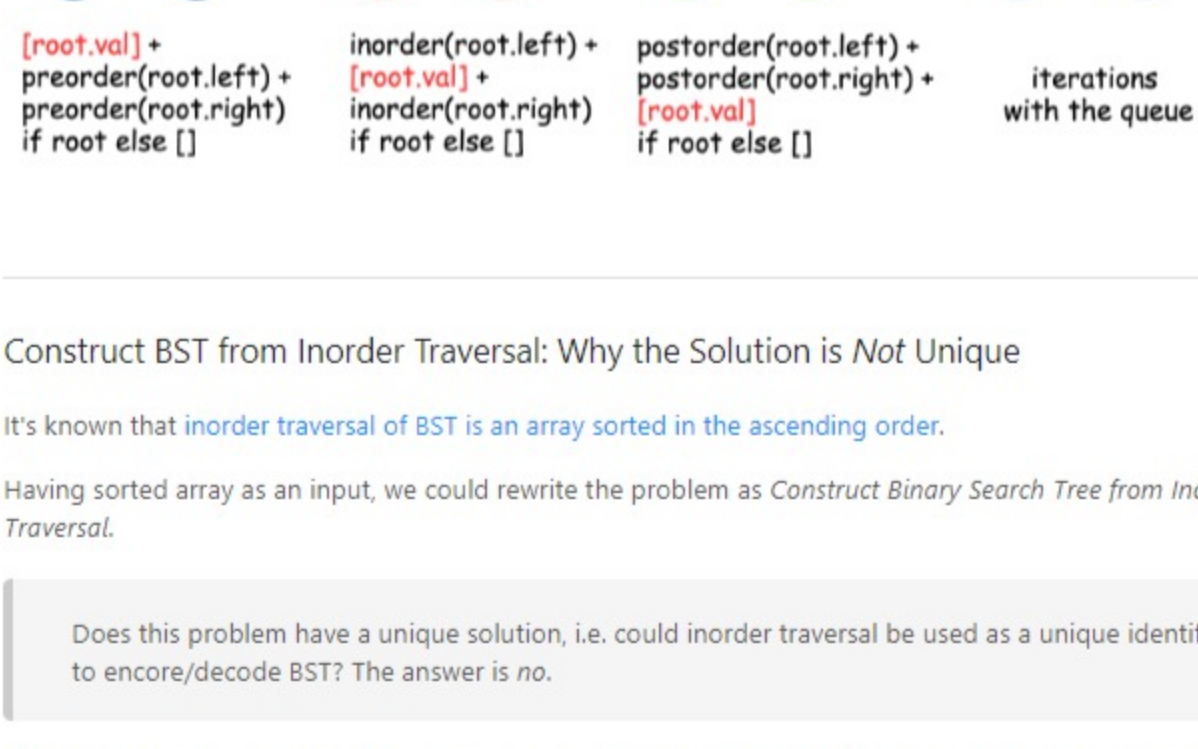
  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

  The DFS strategy can further be distinguished as `preorder`, `inorder`, and `postorder` depending on the relative order among the root node, left node and right node.

- *Breadth First Search* (`BFS`)

  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

On the following figure the nodes are enumerated in the order you visit them, please follow `1-2-3-4-5` to compare different strategies.



Construct BST from Inorder Traversal: Why the Solution is *Not* Unique

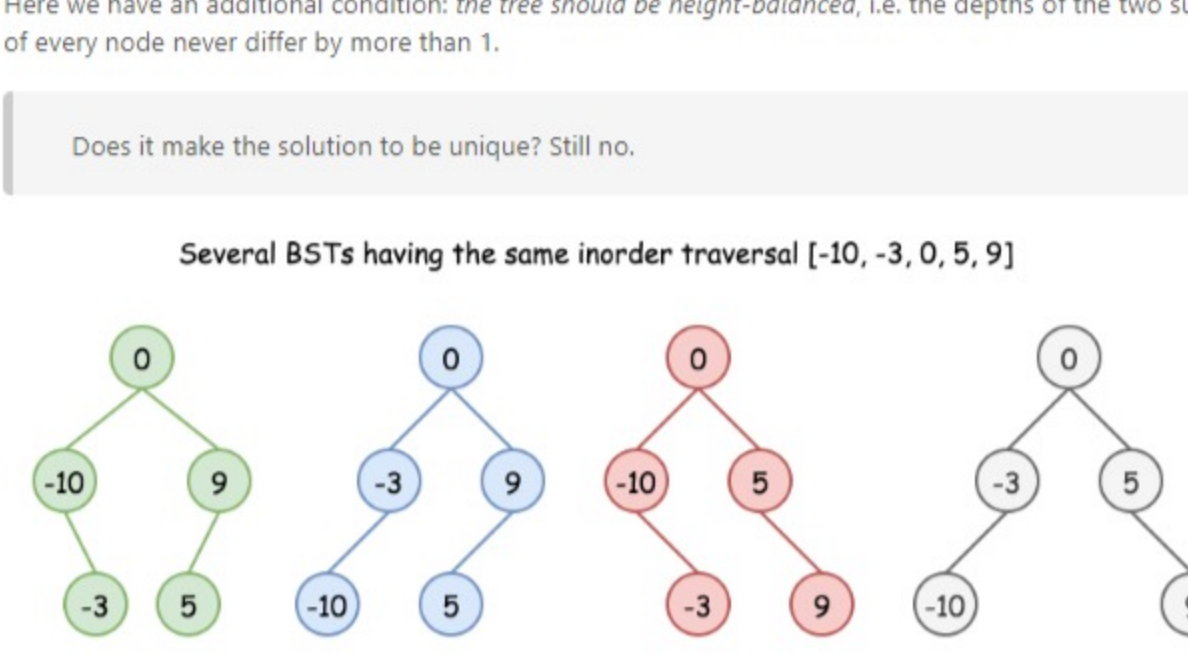
It's known that inorder traversal of BST is an array sorted in the ascending order.

Having sorted array as an input, we could rewrite the problem as *Construct Binary Search Tree from Inorder Traversal.*

> Does this problem have a unique solution, i.e. could inorder traversal be used as a unique identifier to encore/decode BST? The answer is *no*.

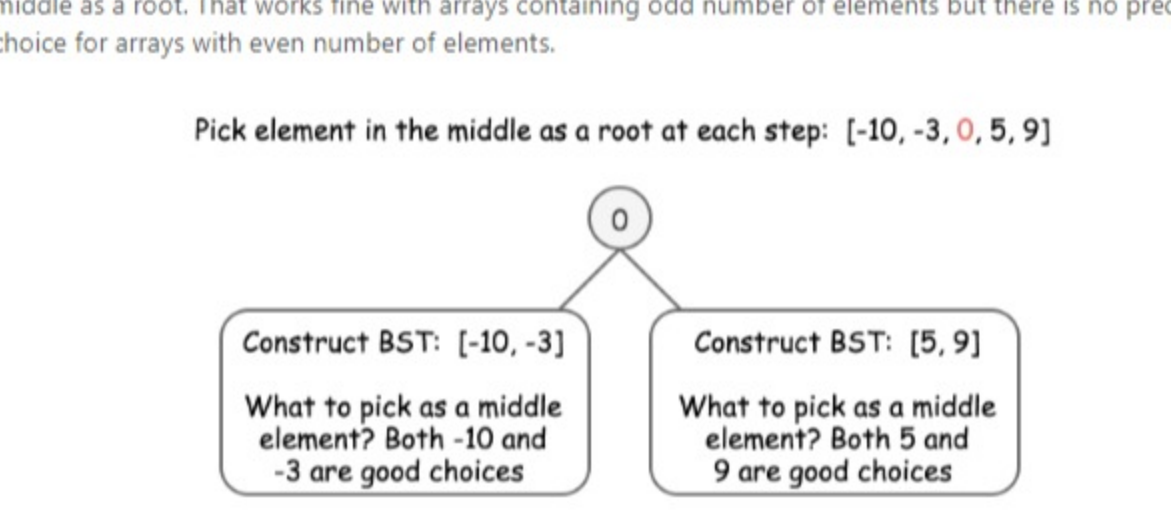
Here is the funny thing about BST. Inorder traversal is *not* a unique identifier of BST. At the same time both preorder and postorder traversals *are* unique identifiers of BST. From these traversals one could restore the inorder one: `inorder = sorted(postorder) = sorted(preorder)`, and inorder + postorder or inorder + preorder are both unique identifiers of whatever binary tree.

So, the problem "sorted array -> BST" has multiple solutions.

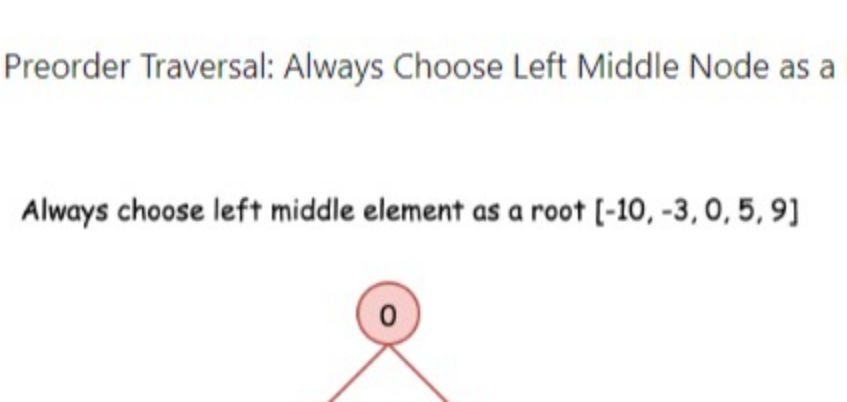


Here we have an additional condition: *the tree should be height-balanced*, i.e. the depths of the two subtrees of every node never differ by more than 1.

> Does it make the solution to be unique? Still no.



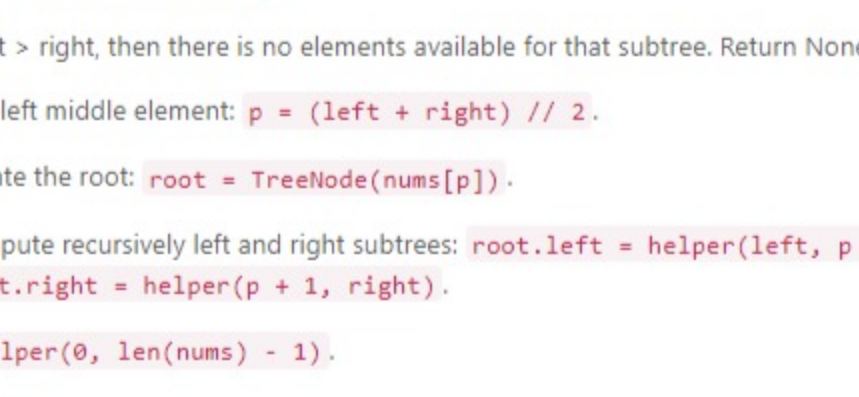
Basically, the height-balanced restriction means that at each step one has to pick up the number in the middle as a root. That works fine with arrays containing odd number of elements but there is no predefined choice for arrays with even number of elements.



One could choose left middle element, or right middle one, and both choices will lead to *different* height-balanced BSTs. Let's see that in practice: in Approach 1 we will always pick up left middle element, in Approach 2 - right middle one. That will generate *different* BSTs but both solutions will be accepted.

Approach 1: Preorder Traversal: Always Choose Left Middle Node as a Root

Algorithm



- Implement helper function `helper(left, right)`, which constructs BST from nums elements between indexes `left` and `right`:
  - If left > right, then there is no elements available for that subtree. Return None.
  - Pick left middle element: `p = (left + right) // 2`.
  - Initiate the root: `root = TreeNode(nums[p])`.
  - Compute recursively left and right subtrees: `root.left = helper(left, p - 1)`, `root.right = helper(p + 1, right)`.
- Return `helper(0, len(nums) - 1)`.

Implementation

Java | Python3 | Copy

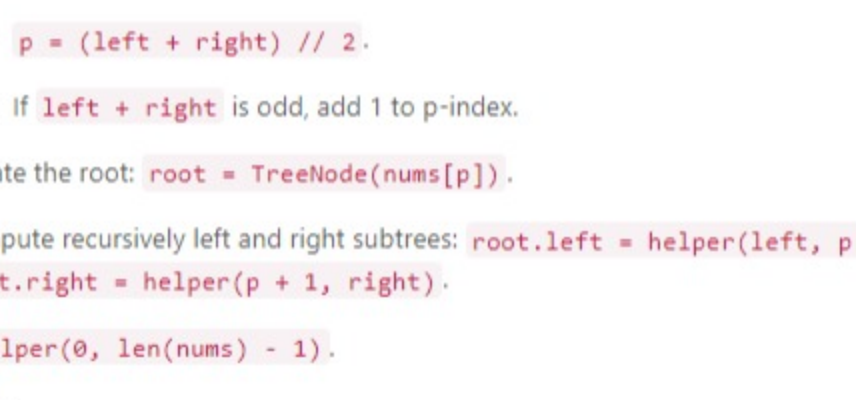
```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        def helper(left, right):
            if left > right:
                return None

            # always choose left middle node as a root
            p = (left + right) // 2

            # preorder traversal: node -> left -> right
            root = TreeNode(nums[p])
            root.left = helper(left, p - 1)
            root.right = helper(p + 1, right)
            return root

        return helper(0, len(nums) - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$ since we visit each node exactly once.
- Space complexity: $\mathcal{O}(N)$. $\mathcal{O}(N)$ to keep the output, and $\mathcal{O}(\log N)$ for the recursion stack.

Approach 2: Preorder Traversal: Always Choose Right Middle Node as a Root

Algorithm



- Implement helper function `helper(left, right)`, which constructs BST from nums elements between indexes `left` and `right`:
  - If left > right, then there is no elements available for that subtree. Return None.
  - Pick right middle element:
    - `p = (left + right) // 2`.
    - If `left + right` is odd, add 1 to p-index.
  - Initiate the root: `root = TreeNode(nums[p])`.
  - Compute recursively left and right subtrees: `root.left = helper(left, p - 1)`, `root.right = helper(p + 1, right)`.
- Return `helper(0, len(nums) - 1)`.

Implementation

Java | Python3 | Copy

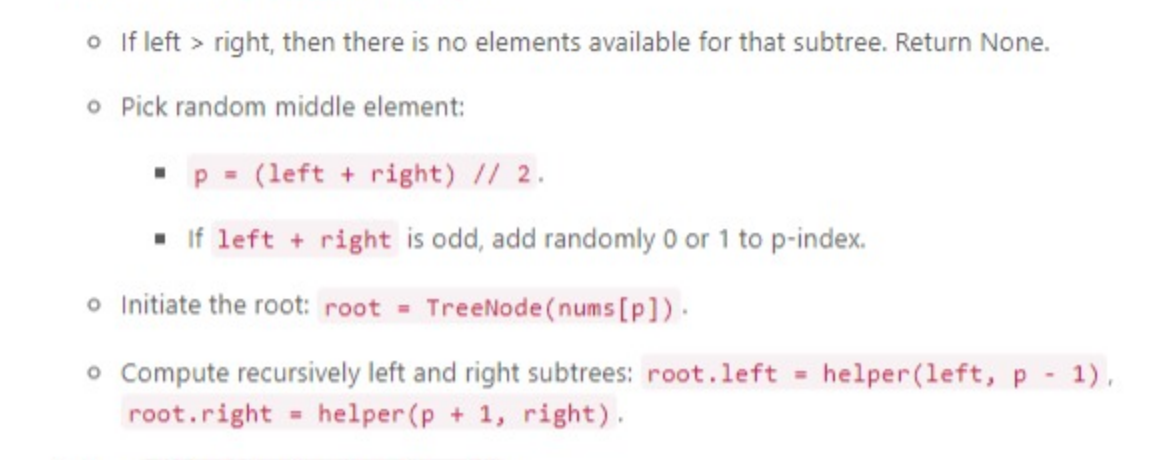
```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        def helper(left, right):
            if left > right:
                return None

            # always choose right middle node as a root
            p = (left + right) // 2
            if (left + right) % 2:
                p += 1

            # preorder traversal: node -> left -> right
            root = TreeNode(nums[p])
            root.left = helper(left, p - 1)
            root.right = helper(p + 1, right)
            return root

        return helper(0, len(nums) - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$ since we visit each node exactly once.
- Space complexity: $\mathcal{O}(N)$. $\mathcal{O}(N)$ to keep the output, and $\mathcal{O}(\log N)$ for the recursion stack.

Approach 3: Preorder Traversal: Choose Random Middle Node as a Root

This one is for fun. Instead of predefined choice we will pick randomly left or right middle node at each step. Each run will result in different solution and they all will be accepted.



Algorithm

- Implement helper function `helper(left, right)`, which constructs BST from nums elements between indexes `left` and `right`:
  - If left > right, then there is no elements available for that subtree. Return None.
  - Pick random middle element:
    - `p = (left + right) // 2`.
    - If `left + right` is odd, add randomly 0 or 1 to p-index.
  - Initiate the root: `root = TreeNode(nums[p])`.
  - Compute recursively left and right subtrees: `root.left = helper(left, p - 1)`, `root.right = helper(p + 1, right)`.
- Return `helper(0, len(nums) - 1)`.

Implementation

Java | Python3 | Copy

```
from random import randint
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        def helper(left, right):
            if left > right:
                return None

            # choose random middle node as a root
            p = (left + right) // 2
            if (left + right) % 2:
                p += randint(0, 1)

            # preorder traversal: node -> left -> right
            root = TreeNode(nums[p])
            root.left = helper(left, p - 1)
            root.right = helper(p + 1, right)
            return root

        return helper(0, len(nums) - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$ since we visit each node exactly once.
- Space complexity: $\mathcal{O}(N)$. $\mathcal{O}(N)$ to keep the output, and $\mathcal{O}(\log N)$ for the recursion stack.

Previous Next

Comments: 12 Sort By

Type comment here... (Markdown is supported)

Preview Post

xuanbryant ★ 66 December 30, 2019 3:31 PM
These approaches should be titled as preorder instead of inorder traversal. It always starts to construct BST from root node.
48 Share Reply
SHOW 2 REPLIES

klyxx ★ 230 December 15, 2019 11:18 PM
It should be universally accepted to use `int mid = lo + (hi - lo) / 2;` for overflow protection, or that bit-shifting one
40 Share Reply

hanyrostom ★ 26 April 6, 2020 5:18 PM
Did not feel like an Easy :|
14 Share Reply

eightdevelopers ★ 9 April 10, 2020 7:54 AM
Seems like the difficulty is at least Medium.
9 Share Reply

fmusk ★ 2 November 15, 2019 11:36 PM
Thanks a lot!
2 Share Reply

hdobhal ★ 1 May 24, 2020 3:40 AM
In approach 2 why you do this
`// always choose right middle node as a root int p = (left + right) / 2; if ((left + right) % 2 == 1) ++p;`
when p = left + (right - left)/2 is the correct way to find mid for binary search
1 Share Reply

Mrmagician ★ 319 May 1, 2020 5:32 AM Report
My Python Submission

```
def recursive(self, nums):
    def rec(nums, start, end):
        if start <= end:
```

Read More
1 Share Reply

hacker0007 ★ 1 February 18, 2020 2:06 AM Report
Can you explain bit more details or example why "Inorder traversal is not a unique identifier of BST. At the same time both preorder and postorder traversals are unique identifiers of BST"?

I go through all the links mentioned, still it's confusing...

Read More
1 Share Reply
SHOW 1 REPLY

savochkin ★ 2 November 17, 2019 2:15 PM
Your solutions do not pass the test case [1,2,2,2,2].
1 Share Reply
SHOW 5 REPLIES

ashwinjosep94 ★ 0 2 days ago
Does the space complexity O(N) to keep the output means, the tree nodes for storing each value?
0 Share Reply

‹ 1 2 ›