230. Kth Smallest Element in a BST

April 30, 2019 | 155.1K views

Given a binary search tree, write a function `kthSmallest` to find the **k**th smallest element in it.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

Follow up:
What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

Constraints:

- The number of elements of the BST is between `1` to `10^4`.
- You may assume `k` is always valid, `1 ≤ k ≤ BST's total elements`.

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Depth First Search* (`DFS`)

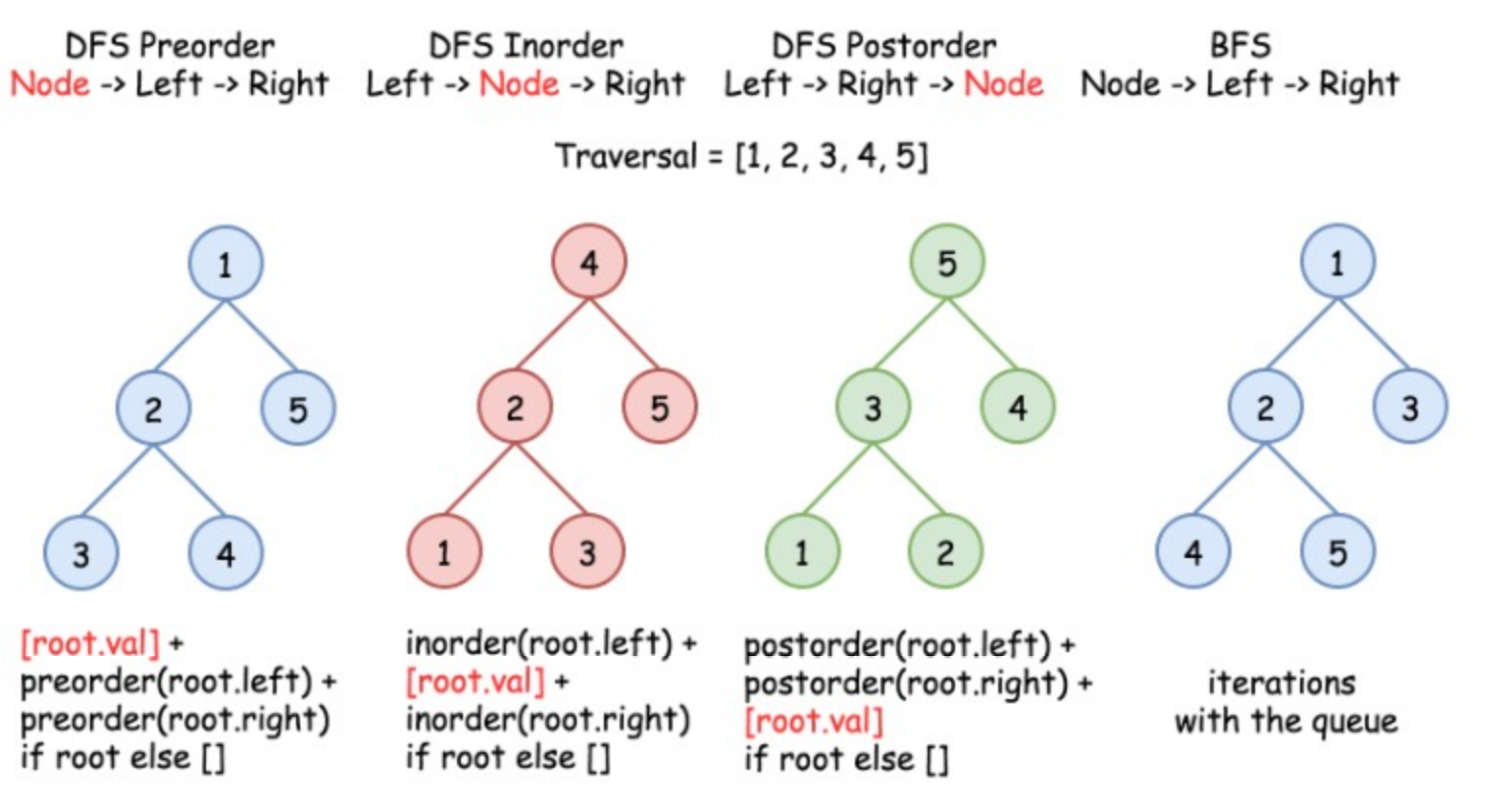
  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

  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.

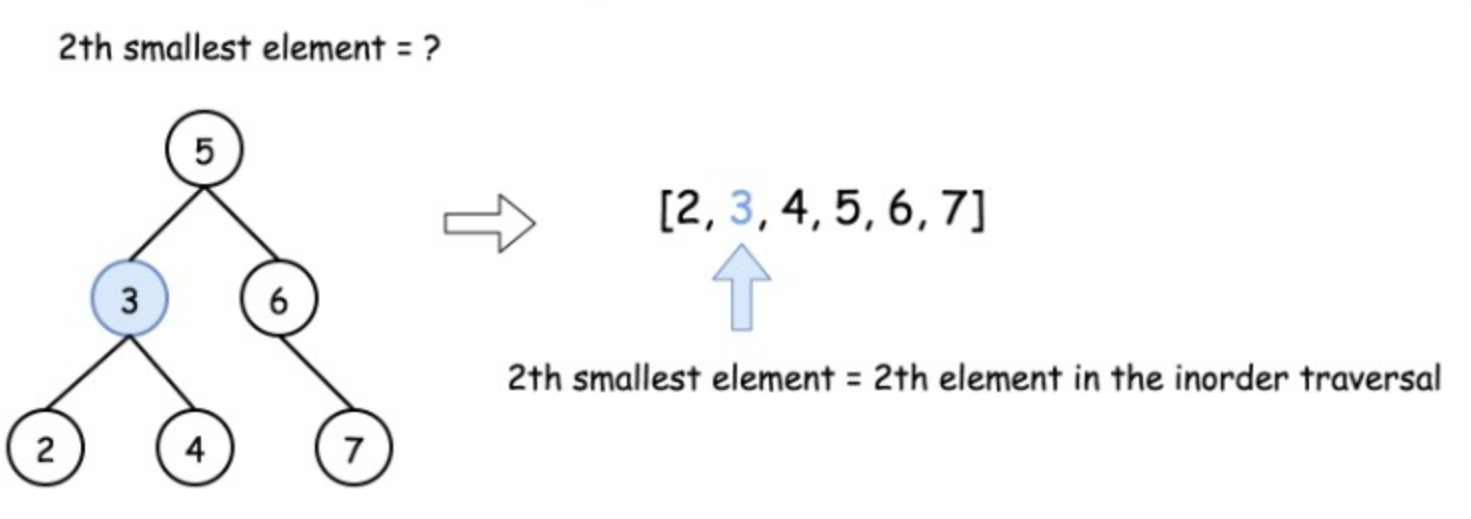
On the following figure the nodes are numerated in the order you visit them, please follow `1-2-3-4-5` to compare different strategies.



> To solve the problem, one could use the property of BST : inorder traversal of BST is an array sorted in the ascending order.

Approach 1: Recursive Inorder Traversal

It's a very straightforward approach with $\mathcal{O}(N)$ time complexity. The idea is to build an inorder traversal of BST which is an array sorted in the ascending order. Now the answer is the `k - 1`th element of this array.



Java | Python | Copy

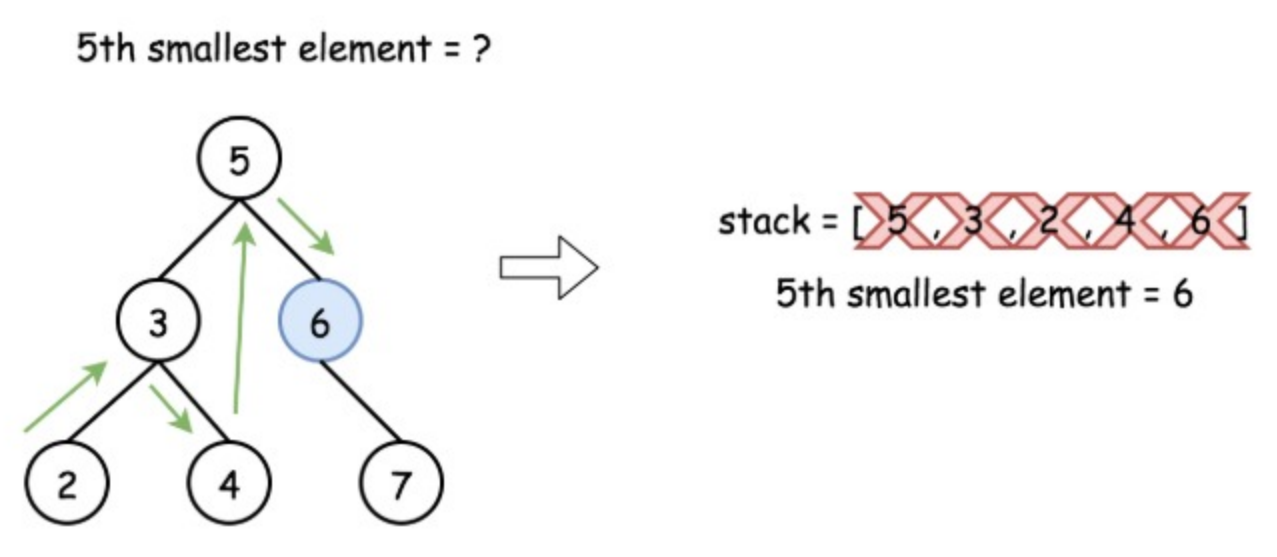
```python
class Solution:
    def kthSmallest(self, root, k):
        """
        :type root: TreeNode
        :type k: int
        :rtype: int
        """
        def inorder(r):
            return inorder(r.left) + [r.val] + inorder(r.right) if r else []

        return inorder(root)[k - 1]
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to build a traversal.
- Space complexity : $\mathcal{O}(N)$ to keep an inorder traversal.

Approach 2: Iterative Inorder Traversal

The above recursion could be converted into iteration, with the help of stack. This way one could speed up the solution because there is no need to build the entire inorder traversal, and one could stop after the kth element.



Java | Python3 | Copy

```python
class Solution:
    def kthSmallest(self, root, k):
        """
        :type root: TreeNode
        :type k: int
        :rtype: int
        """
        stack = []

        while True:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()
            k -= 1
            if not k:
                return root.val
            root = root.right
```

Complexity Analysis

- Time complexity: $\mathcal{O}(H + k)$, where H is a tree height. This complexity is defined by the stack, which contains at least $H + k$ elements, since before starting to pop out one has to go down to a leaf. This results in $\mathcal{O}(\log N + k)$ for the balanced tree and $\mathcal{O}(N + k)$ for completely unbalanced tree with all the nodes in the left subtree.
- Space complexity: $\mathcal{O}(H)$ to keep the stack, where H is a tree height. That makes $\mathcal{O}(N)$ in the worst case of the skewed tree, and $\mathcal{O}(\log N)$ in the average case of the balanced tree.

Follow up

> What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

Insert and delete in a BST were discussed last week, the time complexity of these operations is $\mathcal{O}(H)$, where H is a height of binary tree, and $H = \log N$ for the balanced tree.

Hence without any optimisation insert/delete + search of kth element has $\mathcal{O}(2H + k)$ complexity. How to optimise that?

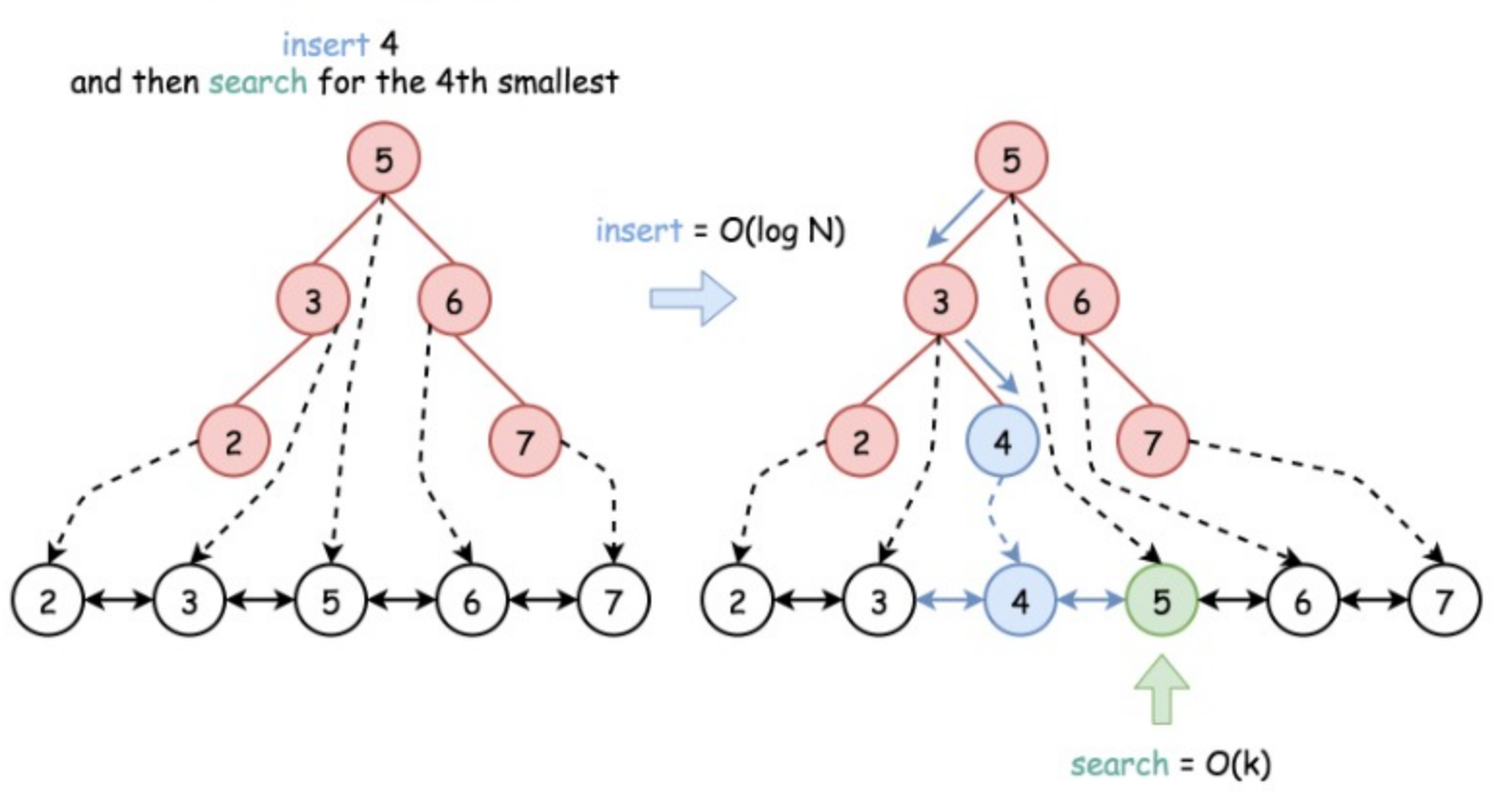
That's a design question, basically we're asked to implement a structure which contains a BST inside and optimises the following operations :

- Insert
- Delete
- Find kth smallest

Seems like a database description, isn't it? Let's use here the same logic as for LRU cache design, and combine an indexing structure (we could keep BST here) with a double linked list.

Such a structure would provide:

- $\mathcal{O}(H)$ time for the insert and delete.
- $\mathcal{O}(k)$ for the search of kth smallest.



The overall time complexity for insert/delete + search of kth smallest is $\mathcal{O}(H + k)$ instead of $\mathcal{O}(2H + k)$.

Complexity Analysis

- Time complexity for insert/delete + search of kth smallest: $\mathcal{O}(H + k)$, where H is a tree height. $\mathcal{O}(\log N + k)$ in the average case, $\mathcal{O}(N + k)$ in the worst case.
- Space complexity : $\mathcal{O}(N)$ to keep the linked list.

Rate this article: ★★★★★

Comments: 51 Sort By

Type comment here... (Markdown is supported)

Preview Post

ashishjain87 ★218 July 22, 2019 1:42 AM
The iterative implementation is beautiful.
114 Share Reply
SHOW 5 REPLIES

happylezhao ★29 May 22, 2019 10:39 PM
Can probably do search in O(H) time, but need O(N) additional storage.
For each node in the tree, we precompute how many are to the left and how many are to the right.
Then at each node, we will know whether the k-th smallest is at the left or the right branch. Search, insert, delete are all O(H).
29 Share Reply
SHOW 8 REPLIES

Sarmon ★623 July 20, 2019 1:23 AM
For people who want to dive more deeply in such design, see B+trees
10 Share Reply

arzgania ★60 June 30, 2019 8:49 AM Report
Can someone explain why the space complexity is O(H + k)? I understand that there could be H maximum elements in the stack (where H = log N, N being number of items in the tree) but why is it H + k?
18 Share Reply
SHOW 5 REPLIES

alexyermolovich ★9 April 9, 2020 11:26 PM
For second approach you can use MaxHeap of size k.
So kth smallest will be on top of the heap.
If new inserted node is greater then top() don't do anything to MaxHeap.
Just insert in to BST. logN (assuming balanced)
Otherwise remove top() logK operation, insert the new node logK
Read More
9 Share Reply
SHOW 5 REPLIES

sri13 ★16 April 10, 2020 7:14 PM
For the design question, I was thinking of an augmented data structure like an Order statistic tree. If we store the number of nodes in the subtree rooted at a given node, shouldn't we be able to answer order statistic queries in lg(n) time? Similarly, i think maintaining these cnts can be done as part of insert and deletes
6 Share Reply
SHOW 1 REPLY

ksangeeth ★8 September 4, 2019 3:52 PM
Augmenting the BST node to include the number of values less than the current node value. That is if inserting or deleting the node, just update this attribute as well. Then the search for the kth smallest element can be done in O(H) time.
6 Share Reply
SHOW 2 REPLIES

codemantra ★46 June 25, 2019 8:19 PM
Not sure if there is a problem with this approach, works for all cases. TC : O(N), SC : O(N)

```
class Solution {
    int k;
    TreeNode result;
```

Read More
4 Share Reply
SHOW 2 REPLIES

mingrui ★111 July 5, 2019 2:32 PM
O(h) time, O(1) auxiliary space solution for the follow up: I think we can just use successor() and predecessor() (O(h) complexity) to maintain the next kth node after each insert() and delete() methods.
For each operation, we compare the edited node and kth node to determine to go forwards or backwards.
2 Share Reply
SHOW 1 REPLY

kremebrulee ★52 July 5, 2019 6:58 AM
In the follow up, are the tree nodes the same as the linked list nodes?
Do the nodes just have more fields?
ex:
class Node {
Node leftChild;
Read More
2 Share Reply
SHOW 2 REPLIES

‹ 1 2 3 4 5 6 ›