Convert Sorted Array to Binary Search Tree

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

Solution 1

Hi everyone, this is my accepted recursive Java solution. I get overflow problems at first because I didn't use mid - 1 and mid + 1 as the bound. Hope this helps :)

```java
public TreeNode sortedArrayToBST(int[] num) {
    if (num.length == 0) {
        return null;
    }
    TreeNode head = helper(num, 0, num.length - 1);
    return head;
}

public TreeNode helper(int[] num, int low, int high) {
    if (low > high) { // Done
        return null;
    }
    int mid = (low + high) / 2;
    TreeNode node = new TreeNode(num[mid]);
    node.left = helper(num, low, mid - 1);
    node.right = helper(num, mid + 1, high);
    return node;
}
```

written by jiaming2 original link here

Solution 2

Recursively call the **sortedArrayToBST()** method providing new vector for each call to construct left and right children:

```
class Solution {
public:
    TreeNode *sortedArrayToBST(vector<int> &num) {
        if(num.size() == 0) return NULL;
        if(num.size() == 1)
        {
            return new TreeNode(num[0]);
        }

        int middle = num.size()/2;
        TreeNode* root = new TreeNode(num[middle]);

        vector<int> leftInts(num.begin(), num.begin()+middle);
        vector<int> rightInts(num.begin()+middle+1, num.end());

        root->left = sortedArrayToBST(leftInts);
        root->right = sortedArrayToBST(rightInts);

        return root;
    }
};
```

written by paul7 original link here

Solution 3

I came up with the recursion solution first and tried to translate it into an iterative solution. It is very similar to doing a tree inorder traversal, I use three stacks - nodeStack stores the node I am going to process next, and **leftIndexStack** and **rightIndexStack** store the range where this node need to read from the**nums**.

```java
public class Solution {

    public TreeNode sortedArrayToBST(int[] nums) {

        int len = nums.length;
        if ( len == 0 ) { return null; }

        // 0 as a placeholder
        TreeNode head = new TreeNode(0);

        Deque<TreeNode> nodeStack       = new LinkedList<TreeNode>() {{ push(head
);  }};
        Deque<Integer>  leftIndexStack  = new LinkedList<Integer>()  {{ push(0);
}};
        Deque<Integer>  rightIndexStack = new LinkedList<Integer>()  {{ push(len-
1); }};

        while ( !nodeStack.isEmpty() ) {
            TreeNode currNode = nodeStack.pop();
            int left  = leftIndexStack.pop();
            int right = rightIndexStack.pop();
            int mid   = left + (right-left)/2; // avoid overflow
            currNode.val = nums[mid];
            if ( left <= mid-1 ) {
                currNode.left = new TreeNode(0);
                nodeStack.push(currNode.left);
                leftIndexStack.push(left);
                rightIndexStack.push(mid-1);
            }
            if ( mid+1 <= right ) {
                currNode.right = new TreeNode(0);
                nodeStack.push(currNode.right);
                leftIndexStack.push(mid+1);
                rightIndexStack.push(right);
            }
        }
        return head;
    }

}
```

written by benjamin19890721 original link here