Verify Preorder Sequence in Binary Search Tree

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Follow up:
Could you do it using only constant space complexity?

Solution 1

Kinda simulate the traversal, keeping a stack of nodes (just their values) of which we're still in the left subtree. If the next number is smaller than the last stack value, then we're still in the left subtree of all stack nodes, so just push the new one onto the stack. But before that, pop all smaller ancestor values, as we must now be in their right subtrees (or even further, in the right subtree of an ancestor). Also, use the popped values as a lower bound, since being in their right subtree means we must never come across a smaller number anymore.

```
public boolean verifyPreorder(int[] preorder) {
    int low = Integer.MIN_VALUE;
    Stack<Integer> path = new Stack();
    for (int p : preorder) {
        if (p < low)
            return false;
        while (!path.empty() && p > path.peek())
            low = path.pop();
        path.push(p);
    }
    return true;
}
```

Solution 2 ... O(1) extra space

Same as above, but abusing the given array for the stack.

```
public boolean verifyPreorder(int[] preorder) {
    int low = Integer.MIN_VALUE, i = -1;
    for (int p : preorder) {
        if (p < low)
            return false;
        while (i >= 0 && p > preorder[i])
            low = preorder[i--];
        preorder[++i] = p;
    }
    return true;
}
```

Solution 3 ... Python

Same as solution 1, just in Python.

```python
def verifyPreorder(self, preorder):
    stack = []
    low = float('-inf')
    for p in preorder:
        if p < low:
            return False
        while stack and p > stack[-1]:
            low = stack.pop()
        stack.append(p)
    return True
```

written by StefanPochmann original link here

Solution 2

The idea is traversing the preorder list and using a stack to store all predecessors. *currp is a predecessor of current node and current node is in the right subtree of curr*p.

For example, for the following bst with preorder 6,3,1,2,5,4,7:

```
              6
            /   \
          3      7
        /   \
      1     5
      \    /
      2   4
```

We push to stack before we see 2. So at 2 the stack is 6,3,1. For 2, we pop stack until we see 3 which is greater than 2 and curr*p is 1. 2 is in left subtree of 3 and is right child of 1. Stack is 6,3,2 now. Then we see 5, and we pop stack until 6 and curr*p is 3. Stack now is 6,5. Then we see 4 and push to stack. At 7, we pop stack until empty and curr_p is 6.

```cpp
bool verifyPreorder(vector<int>& preorder){
    // using stack
    int sz = preorder.size();
    if(sz < 2) return true;
    stack<int> s;
    s.push(preorder[0]);
    int curr_p = INT_MIN;
    for(int i=1; i<sz; i++){
        if(s.empty() || preorder[i]<s.top()){ // if current node is less than stack top, then go to left subtree
            if(preorder[i]<curr_p) return false;
            s.push(preorder[i]);
        }
        else{
            while(!s.empty() && s.top()<preorder[i]){ //find curr_p such that current node is right child of curr_p
                curr_p = s.top();
                s.pop();
            }
            s.push(preorder[i]);
        }
    }
    return true;
}
```

written by yu23 original link here

Solution 3

A BST's left child is always < root and right child is always > root.

```
public boolean verifyPreorder(int[] preorder) {
    if(preorder == null || preorder.length == 0) return true;
    return verify(preorder, 0, preorder.length - 1);
}

private boolean verify(int[] a, int start, int end) {
    if(start >= end) return true;
    int pivot = a[start];
    int bigger = -1;
    for(int i = start + 1; i <= end; i++) {
        if(bigger == -1 && a[i] > pivot) bigger = i;
        if(bigger != -1 && a[i] < pivot) return false;
    }
    if(bigger == -1) {
        return verify(a, start + 1, end);
    } else {
        return verify(a, start + 1, bigger - 1) && verify(a, bigger, end);
    }
}
```

written by qianzhige original link here