# 题目

给定一个整数数组，你需要验证它是否是一个二叉搜索树正确的先序遍历序列。

你可以假定该序列中的数都是不相同的。

参考以下这颗二叉搜索树：

5

/ \

2 6

/ \

1 3

示例 1：

输入: [5,2,6,1,3]

输出: false

示例 2：

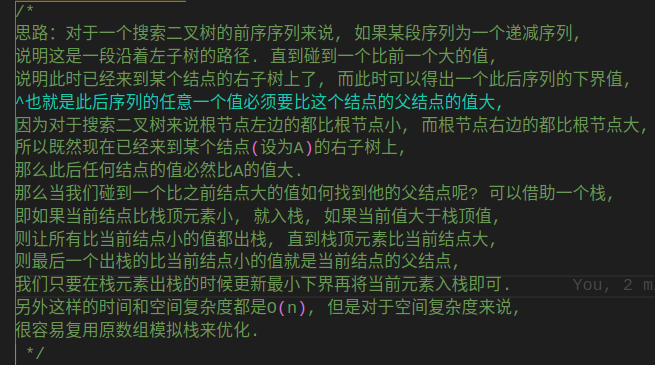
输入: [5,2,1,3,6]

输出: true

进阶挑战：

您能否使用恒定的空间复杂度来完成此题？

# 分析



bool verifyPreorder(vector<int>& pre) {

int k = -1, Min = INT32\_MIN;

for (auto val : pre) {

if (val < Min) return false;

while (k >= 0 && pre[k] < val) Min = pre[k--];

pre[++k] = val; //<复用原数组模拟栈,以优化内存

}

return true;

}

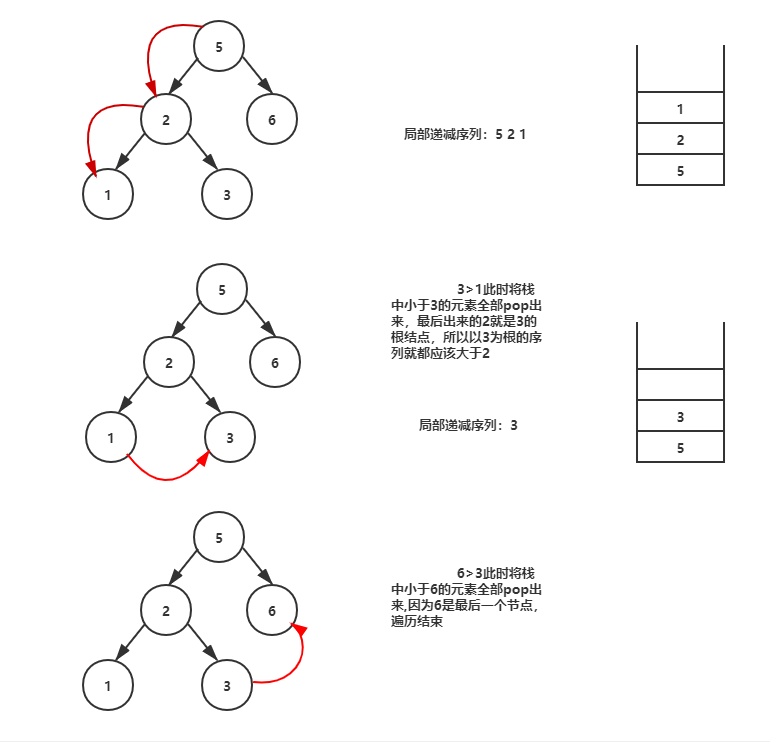
解题思路

1.二叉排序树的定义，对于当前节点root，左边都比他小，右边的都比他大；

2.先序遍历，根->左子->右子

由这两个特性决定了一个正确的二叉排序树的先序序列应该满足局部递减，总体递增

遍历过程如下：



class Solution {

public:

stack<int> s;

int last\_root=0;

bool verifyPreorder(vector<int>& preorder) {

if(preorder.size()==0) return true;

s.push(preorder[0]);

for(int i=1;i<preorder.size();i++){

//cout<<last\_root<<endl;

if(preorder[i]<preorder[i-1]){

if(preorder[i]<last\_root) return false;

s.push(preorder[i]);

}

else{

while(!s.empty()&&s.top()<preorder[i]){

last\_root=s.top();

s.pop();

}

//此时last\_root->right=preorder[i],所以以preorder[i]为根继续的局部递减序列一定都比last\_root大

s.push(preorder[i]);

}

}

//对于最后面一直递减的情况处理尾巴

// while(!s.empty()&&s.top()<preorder[i]){

// }

return true;

}

};

先序遍历搜索树，只有左子树时元素是单调递减的，但存在某一结点的右子树就不满足单调。如

# 5

# / \

# 2 6

# / \

# 1 3

先序遍历[5,2,1,3,6]。只有左子树[5,2,1]，加入2的右子树变成[5,2,1,3]。因此为了保持栈的单调，元素大于栈顶元素，弹出栈中小于当前结点的值，并把当前节点入栈，变成[5,3]，这时栈中又只剩下左子树，满足单调。最后一个弹出栈2的是当前节点3的根节点。我们使用root记录根节点的值，因为程序中当前节点总在根节点右边，所以当前节点小于根节点就是非法。

class Solution {

public:

bool verifyPreorder(vector<int>& preorder) {

if(preorder.empty()) return true;

stack<int> st;

int root = INT\_MIN;

for(int num : preorder){

if(num <root)

return false;

while(!st.empty() && num > st.top()){

root = st.top();

st.pop();

}

st.push(num);

}

return true;

}

};

时间复杂度O(n)O(n)，空间复杂度O(logn)O(logn)。如果使用递归时间复杂度O(n^2)O(n

2

)，时间超时。递归伪代码：

bool verifyPreordCore(vector<int>& preorder, int start,int end){

if(start > end) return true;

int root = preorder[start], littleRange = start + 1, largeRange = start + 1;

while(littleRange <= end && preorder[littleRange] < root)

littleRange ++;

largeRange = littleRange;

while(largeRange <= end){

if(preorder[largeRange] < root)

return false;

largeRange++;

}

return verifyPreordCore(preorder, start+1, littleRange-1) &&

verifyPreordCore(preorder, littleRange, end);

}

思路和心得：

1.这个题真怪

2.局部递减，整体递增

3.单调递减栈

4.时刻更新左侧最上的点

（一条左上至右下的斜线，该斜线最左上的点， 就是维持单调递减栈时的弹出点）

class Solution

{

public:

bool verifyPreorder(vector<int>& preorder)

{ //整体递增，局部递减 用单调递减栈

stack<int> dec\_stack;

int pre\_max = -0x3f3f3f3f;

for (int x: preorder)

{

if (pre\_max > x) //新来的，都在右侧，应该大于之前的最大值（整体递增）

return false;

while (dec\_stack.size() && dec\_stack.top() <= x) //维护单调递减栈(局部递减)

{

pre\_max = dec\_stack.top();

dec\_stack.pop();

}

dec\_stack.push(x); //入栈

}

return true;

}

};

解题思路

解法1：递归。调用一个helper函数verify,保存递归到当前的最大值maxVal和最小值minVal。起点start和终点end。如果start > end说明为空树，返回true。否则若rootVal大于最大值maxVal或小于最小值minVal直接返回false。否则查找第一个大于rootVal的值为其右子树。返回递归判断左右子树是否为二叉搜索树。

解法2：迭代

代码

class Solution1 {

public:

bool verifyPreorder(vector<int>& preorder) {

return verify(preorder, 0, preorder.size() - 1, numeric\_limits<int>::min(), numeric\_limits<int>::max());

}

private:

bool verify(vector<int>& preorder, int start, int end, int minVal, int maxVal) {

if (start > end) {

return true;

}

int rootVal = preorder[start];

if (rootVal <= minVal || rootVal >= maxVal) {

return false;

}

int i = start + 1;

for (; i <= end; i++) {

if (preorder[i] >= rootVal) {

break;

}

}

return verify(preorder, start + 1, i - 1, minVal, rootVal) &&

verify(preorder, i, end, rootVal, maxVal);

}

};

class Solution {

public:

bool verifyPreorder(vector<int>& preorder) {

int low = numeric\_limits<int>::min();

int i = -1;

for (const auto& num : preorder) {

if (num < low) {

return false;

}

while (i >= 0 && num > preorder[i]) {

low = preorder[i--];

}

preorder[++i] = num;

}

return true;

}

};