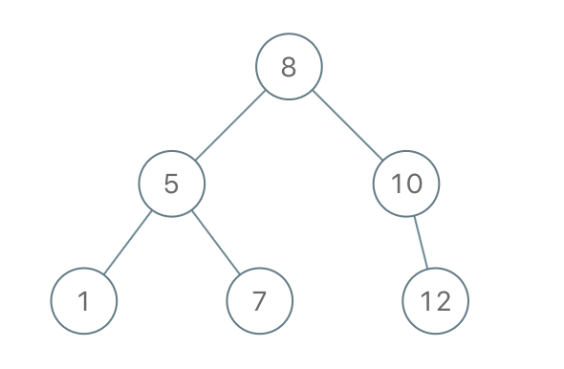
# 描述

返回与给定先序遍历 preorder 相匹配的二叉搜索树（binary search tree）的根结点。

(回想一下，二叉搜索树是二叉树的一种，其每个节点都满足以下规则，对于 node.left 的任何后代，值总 < node.val，而 node.right 的任何后代，值总 > node.val。此外，先序遍历首先显示节点的值，然后遍历 node.left，接着遍历 node.right。）

**示例：**



输入：[8,5,1,7,10,12]

输出：[8,5,10,1,7,null,12]

**提示：**

1 <= preorder.length <= 100

先序 preorder 中的值是不同的。

# 分析

要解决这个问题，我们需要根据给定的先序遍历序列构建对应的二叉搜索树（BST）。二叉搜索树的特性是：对于任意节点，其左子树的所有节点值都小于该节点值，右子树的所有节点值都大于该节点值。先序遍历的顺序是“根->左->右”，我们可以利用这两个特性来构建BST。

解题思路

1、先序遍历特性：先序遍历的第一个元素是根节点，接下来的元素中，小于根节点值的部分属于左子树，大于根节点值的部分属于右子树。

2、递归构建：对于每个子树，我们可以采用同样的方法：以当前子序列的第一个元素为根节点，划分出左子树和右子树的子序列，然后递归构建左子树和右子树。

具体步骤如下：

- 若当前序列为空，返回空节点。

- 取序列的第一个元素作为当前根节点的值。

- 划分左子树序列（所有小于根节点值的元素）和右子树序列（所有大于根节点值的元素）。

- 递归构建左子树和右子树，并将它们分别作为当前根节点的左孩子和右孩子。

代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

\* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

\* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

\* };

\*/

class Solution {

public:

TreeNode\* bstFromPreorder(vector<int>& preorder) {

return build(preorder, 0, preorder.size() - 1);

}

private:

// 递归构建BST，参数为当前子序列的起始和结束索引

TreeNode\* build(vector<int>& preorder, int start, int end) {

// 若起始索引大于结束索引，说明当前子序列为空，返回空节点

if (start > end) {

return nullptr;

}

// 以当前子序列的第一个元素为根节点

int rootVal = preorder[start];

TreeNode\* root = new TreeNode(rootVal);

// 寻找左子树和右子树的分界点：第一个大于根节点值的元素索引

int split = start + 1;

while (split <= end && preorder[split] < rootVal) {

split++;

}

// 递归构建左子树（[start+1, split-1]为左子树序列）

root->left = build(preorder, start + 1, split - 1);

// 递归构建右子树（[split, end]为右子树序列）

root->right = build(preorder, split, end);

return root;

}

};

解释：

- 递归函数设计：build函数接收先序遍历序列以及当前子序列的起始和结束索引，返回构建好的子树的根节点。

- 根节点选择：每个子序列的第一个元素（preorder[start]）作为当前子树的根节点。

- 划分左右子树：通过遍历找到第一个大于根节点值的元素索引split，该索引左边的元素（start+1到split-1）构成左子树的先序序列，右边的元素（split到end）构成右子树的先序序列。

- 递归构建：分别对左子树序列和右子树序列递归调用build函数，得到的结果分别作为当前根节点的左孩子和右孩子。

这种方法充分利用了先序遍历和二叉搜索树的特性，通过递归实现了树的构建，时间复杂度为O(n^2)（最坏情况下，每次划分左子树需要遍历整个子序列），空间复杂度为O(n)（递归栈深度，最坏情况下为线性）。对于题目中n <= 100的约束，该方法完全适用。