后序遍历

代码框架：

```
void traverse(TreeNode *root) {
  traverse(root->left);
  traverse(root->right);
  // 处理当前节点
}
```

后序遍历使用：要做的事情是通过左右子树计算出来的结果，就要用到后序遍历。

> 给你输入一棵二叉树，这棵二叉树的子树中可能包含二叉搜索树对吧，请你找到节点之和最大的那棵二叉搜索树，返回它的节点值之和。

分析：

1. 求和需要根据左右自己树的结果来判断；
2. 对指定节点需要判断是否为BST,进一步求解和。

常规思路：

1. 采用先序遍历，求解当前root是否构成BST当构成BST时，求解其和。解出最大值。(超时)

考虑递归算法可以返回vector的情况； 定义：`std::vector<int> res = {是否BST，最小节点值，最大节点值，当前子树和}`

这样其子树的状态通过res返回，如果root为BST,更新当前节点的res；否则res[0] = 0即可。 最后取BST计算中的最大值。

```
class Solution {
public:
  int maxSumBST(TreeNode *root) {
    traverse(root);
    return maxSum;
  }

private:
  // isBST(0,1是), min_val(最小节点值), max_val(最大节点值), sum(子树和)
  std::vector<int> traverse(TreeNode *root) {
    std::vector<int> res(4, 0);

    if (root == nullptr) {
      res[0] = 1;
      res[1] = INT_MAX;
      res[2] = INT_MIN;
      res[3] = 0;
```

```
      return res;
    }

    // 左右子树
    std::vector<int> left  = traverse(root->left);
    std::vector<int> right = traverse(root->right);

    // BST 更新, BST 根节点值大于左子树最大值, 右子树最小值
    if (left[0] && right[0] && root->val > left[2] && root->val <
right[1]) {
      int sum = left[3] + right[3] + root->val;
      res[0]  = 1;
      res[1]  = std::min(root->val, left[1]);
      res[2]  = std::max(root->val, right[2]);
      res[3]  = sum;

      maxSum = (maxSum > sum ? maxSum : sum);
    } else {
      res[0] = 0;
      res[1] = INT_MAX;
      res[2] = INT_MIN;
      res[3] = 0;
    }

    return res;
  }

  int maxSum = 0;
};
```