## 就看是否需要遍历整个二叉树

如果需要搜索整棵二叉树，**那么递归函数就不要返回值**，如果要搜**索其中一条符合条件的路径，递归函数就需要返回值**，因为遇到符合条件的路径就要返回

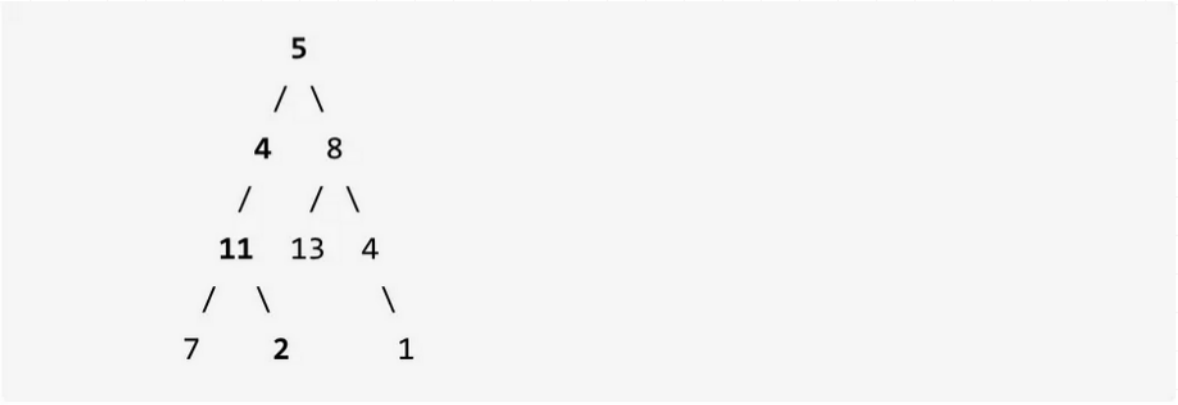
> **对于根节点到叶子节点的路径之和等于目标值，我们可以不用累加形式（代码麻烦，此时需要一个累加器和一个目标值）我们可以采用累减的形式，当减到0时候就为满足条件的时候**

### 112. 路径总和

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

说明：叶子节点是指没有子节点的节点。

示例：给定如下二叉树，以及目标和 sum = 22，




返回 true，因为存在目标和为 22 的根节点到叶子节点的路径 5->4->11->2。

> 第一步：确定递归函数参数和返回值
> bool traversal(TreeNode* cur,int count);//注意函数的返回类型
> 第二步：确定递归终止条件
> 如果为叶子节点且count==0说明找到目标路径则返回true
> 如果为叶子节点但count!=0说明没有找到目标路径则返回false
> 第三步单层逻辑
> if(cur->left)traversal(cur->left,count-cur->left->val)
> if(cur->right)traversal(cur->right,count-cur->right->val)**//这一条代码中存在着回溯的算法含义在里面**
> return 两者traversal的返回bool或

## 精简代码如下：

```
class Solution {
public:
 bool tranversal(TreeNode* root,int count){
 //递归终止条件
 if(!root->left&&!root->right) return count==0?true:false;
```

```
 bool flag=false;
 if(root->left) flag|=tranversal(root->left,count-root->left->val);
 if(root->right) flag|=tranversal(root->right,count-root->right->val);
 return flag;

 }
    bool hasPathSum(TreeNode* root, int targetSum) {
 if(root==NULL) return false;
 return tranversal(root,targetSum-root->val);
    }
};
```

## 迭代算法就是用栈模拟递归过程(非常美妙!!)

> 每次栈模拟的过程中都要存放一个TreeNode节点指针和相应的数据
> 如果是附带一个相应数据则可以用pair<>数据结构

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
 stack<pair<TreeNode*,int>> sta;
 if(!root)return false;
 sta.push(make_pair(root,targetSum-root->val));
 while(!sta.empty()){
 //出栈的过程我们可以理解为进入递归函数了
 pair<TreeNode*,int> pairNode=sta.top();
 sta.pop();
 TreeNode* node=pairNode->first;
 int count=pairNode->second;
 if(!node->left&&!node->right&&count==0) return true;
 if(node->left)sta.push(make_pair(node->left,count-node->left->val));
 if(node->right)sta.push(make_pair(node->right,count-node->right->val));
 }
 return false;
    }
};
```

给你二叉树的根节点 `root` 和一个整数目标和 `targetSum` ，找出所有 **从根节点到叶子节点** 路径总和等于给定目标和的路径。

**叶子节点** 是指没有子节点的节点。

> 此时递归函数不需要返回值