# 题目

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，

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

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。

类似题目：LeetCode 257

# 分析

## 方法一：递归法

**思路：**

递归法，即深度优先遍历

**代码：**

class Solution {

public:

bool hasPathSum(TreeNode\* root, int sum) {

if (root == NULL) return false;

if (root->left == NULL && root->right == NULL &&

root->val == sum)

//叶子节点且值相等

return true;

if (root->left != NULL &&

hasPathSum(root->left, sum - (root->val)))

//左叶节点不为空，右节点符合要求

return true;

if (root->right != NULL &&

hasPathSum(root->right, sum - (root->val)))

return true;

return false;

}

};

或：

class Solution {

public:

bool dfs(TreeNode\* root, int sum){

if(root == NULL){

return false;

}

if(!root->left &&!root->right){

if(root->val == sum){

return true;

}

}

return (dfs(root->left, sum-root->val)||dfs(root->right, sum -root->val));

}

bool hasPathSum(TreeNode\* root, int sum) {

return dfs(root, sum);

}

};

或：

class Solution {

public:

    bool hasPathSum(TreeNode\* root, int targetSum) {

        if(nullptr==root)   return false;

        return dfs(root,targetSum);

    }

    bool dfs(TreeNode\* root,int sum)

    {

        if(nullptr==root)

            return false;

        sum -= root->val;

        if(nullptr==root->left && nullptr==root->right && 0==sum)

        {

            return true;

        }

        return dfs(root->left,sum) || dfs(root->right,sum);

    }

};

或（不建议使用这种方法，不具备通用性）：

class Solution {

public:

int flag=0;

void dfs(TreeNode\* root,int cnt,int sum){

if(!root) return ;

if(!root->left&&!root->right){

if(sum==cnt+root->val) flag=1;

return ;

}

dfs(root->left,cnt+root->val,sum);

dfs(root->right,cnt+root->val,sum);

}

bool hasPathSum(TreeNode\* root, int sum) {

if(!root) return false;

dfs(root,0,sum);

if(flag) return true;

return false;

}

};

## 方法二：迭代法

**思路：**利用栈实现深度优先遍历

**代码：**

bool hasPathSum(TreeNode\* root, int sum) {

//参数1-二叉树的根；参数2-剩余和

if (!root) return false;

stack<TreeNode\*> stk;

TreeNode \*prev = nullptr;

while (root || !stk.empty()) {

while (root) {

stk.push(root);

sum -= root->val; //入栈后更新剩余和

root = root->left; //递归左子树

}

root = stk.top(); //取得右拐点（根结点）

if (!root->left && !root->right && (sum == 0)) {

//判断路径是否满足条件

return true;

}

if (!root->right || root->right == prev) {

//右结点不存在或者，右结点已访问

stk.pop();

sum += root->val; //出栈后更新剩余和

prev = root; //标记在后序遍历序列已访问

root = nullptr; //置空，用于回溯

}

else {

root = root->right; //访问右结点

}

}//模拟递归栈

return false;

}