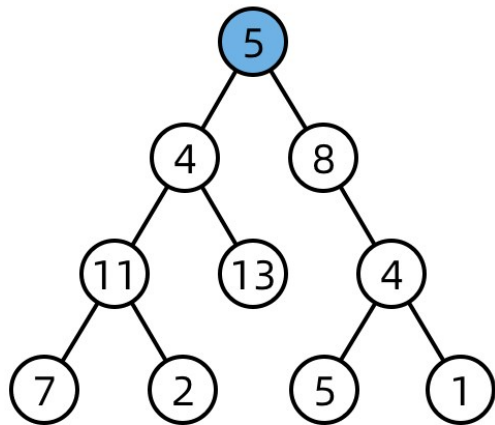




一篇文章解决所有二叉树路径问题(问题分析+分类模板+题目剖析)

星晴pro L5
发布于 2021-06-06 ©447树C++



对于刚刚接触树的问题的新手而言, 路径问题是一个比较棘手的问题。题解中关于二叉树路径问题的总结还偏少, 今天我用一篇文章总结一下二叉树的路径问题。学透这篇文章, 二叉树路径题可以秒杀

问题分类

二叉树路径的问题大致可以分为两类:

1、自顶向下:

顾名思义, 就是从某一个节点(不一定是根节点), 从上向下寻找路径, 到某一个节点(不一定是叶节点)结束

具体题目如下:

257. 二叉树的所有路径

面试题 04.12. 求和路径

112. 路径总和

113. 路径总和 II

437. 路径总和 III

988. 从叶结点开始的最小字符串

而继续细分的话还可以分成一般路径与给定和的路径

2、非自顶向下:

就是从任意节点到任意节点的路径, 不需要自顶向下

124. 二叉树中的最大路径和

687. 最长同值路径

543. 二叉树的直径

解题模板

这类题通常用深度优先搜索(DFS)和广度优先搜索(BFS)解决, BFS较DFS繁琐, 这里为了简洁只展现DFS代码

下面是我对两类题目的分析与模板

一、自顶而下:

dfs

一般路径:

```
vector<vector<int>>>res;
void dfs(TreeNode*root,vector<int>path)
{
    if(!root) return; //根节点为空直接返回
    path.push_back(root->val); //作出选择
    if(!root->left && !root->right) //如果到叶节点
    {
        res.push_back(path);
        return;
    }
    dfs(root->left,path); //继续递归
    dfs(root->right,path);
}

# **给定和的路径:**
void dfs(TreeNode*root, int sum, vector<int> path)
{
    if (!root)
        return;
    sum -= root->val;
    path.push_back(root->val);
    if (!root->left && !root->right && sum == 0)
    {
        res.push_back(path);
        return;
    }
    dfs(root->left, sum, path);
    dfs(root->right, sum, path);
}
```

这类题型DFS注意点:

1、如果是找路径和等于给定target的路径的, 那么可以不用新增一个临时变量cursum来判断当前路径和, 只需要用给定和target减去节点值, 最终结束条件判断target==0即可

2、是否要回溯: 二叉树的问题大部分是不需要回溯的, 原因如下:

二叉树的递归部分: dfs(root->left),dfs(root->right)已经把可能的路径穷尽了, 因此到任意叶节点的路径只可能有一条, 绝对不可能出现另外的路径也到这个满足条件的叶节点的;

而对比二维数组(例如迷宫问题)的DFS,for循环向四个方向查找每次只能朝向一个方向, 并没有穷尽路径, 因此某一个满足条件的点可能是有多条路径到该点的

并且visited数组标记已经走过的路径是会受到另外路径是否访问的影响, 这时候必须回溯

3、找到路径后是否要return;

取决于题目是否要求找到叶节点满足条件的路径,如果必须到叶节点,那么就要return;

但如果是到任意节点都可以,那么必不能return,因为这条路径下面还可能更深的路径满足条件,还要在此基础上继续递归

4、是否要双重递归(即调用根节点的dfs函数后,继续调用根左右节点的pathsum函数):看题目要不要从根节点开始的,还是从任意节点开始

二、非自顶而下:

这类题目一般解题思路如下:

设计一个辅助函数maxpath,调用自身求出以一个节点为根节点的左侧最长路径left和右侧最长路径right,那么经过该节点的最长路径就是left+right

接着只需要从根节点开始dfs,不断比较更新全局变量即可

```
int res=0;

int maxPath(TreeNode *root) //以root为路径起始点的最长路径
{
    if (!root)
        return 0;
    int left=maxPath(root->left);
    int right=maxPath(root->right);
    res = max(res, left + right + root->val); //更新全局变量
    return max(left, right); //返回左右路径较长者
}
```

这类题型DFS注意点:

1、left,right代表的含义要根据题目所求设置,比如最长路径、最大路径和等等

2、全局变量res的初值设置是0还是INT_MIN要看题目节点是否存在负值,如果存在就用INT_MIN,否则就是0

3、注意两点之间路径为1,因此一个点是不能构成路径的

题目分析

下面是对具体题目的分析和代码呈现

一、自顶向下

257. 二叉树的所有路径

直接套用模板1即可,注意把“->”放在递归调用中

```
vector<string> res;

vector<string> binaryTreePaths(TreeNode<T> *root)
{
    dfs(root, "");
    return res;
}
```

```
void dfs(TreeNode*root, string path)
{
    if (!root)
        return;
    path += to_string(root->val);
    if (!root->left && !root->right)
    {
        res.push_back(path);
        return;
    }
    dfs(root->left, path+"->");
    dfs(root->right, path+"->");
}
```

113. 路径总和 II

直接套用模板2

```
vector<vector<int>> res;

vector<vector<int>> pathSum(TreeNode *root, int targetSum)
{
    vector<int> path;
    dfs(root, targetSum, path);
    return res;
}
```

```
void dfs(TreeNode*root, int sum, vector<int> path)
{
    if (!root)
        return;
    sum -= root->val;
    path.push_back(root->val);
    if (!root->left && !root->right && sum == 0)
    {
        res.push_back(path);
        return;
    }
    dfs(root->left, sum, path);
    dfs(root->right, sum, path);
}
```

437. 路径总和 III

双重递归:先调用dfs函数从root开始查找路径,再调用pathsum函数到root左右子树开始查找

套用模板2

```
int count = 0;

int pathSum(TreeNode *root, int targetSum)
{
    if (!root)
        return 0;
    dfs1(root, targetSum); //以root为起始点查找路径
    pathSum(root->left, targetSum); //左子树递归
    pathSum(root->right, targetSum); //右子树递归
    return count;
}
```

```
void dfs(TreeNode *root, int sum)
{
    if (!root)
        return;
```

```

sum -= root->val;
if (sum == 0) //注意不要return,因为不要求到叶节点结束,所以一条路径下面还可能有一条
    count++; //如果找到了一个路径全局变量就+1
dfs1(root->left, sum);
dfs1(root->right, sum);
}

```

988. 从叶结点开始的最小字符串
 换汤不换药, 套用模板1

```

vector<string> path;
string smallestFromLeaf(TreeNode *root)
{
    dfs(root, "");
    sort(path.begin(), path.end()); //升序排序
    return path[0];
}

```

```

void dfs(TreeNode *root, string s)
{
    if (!root)
        return;
    s += 'a' + root->val;
    if (!root->left && !root->right)
    {
        reverse(s.begin(), s.end()); //题目要求从根节点到叶节点, 因此反转
        path.push_back(s);
        return;
    }
    dfs(root->left, s);
    dfs(root->right, s);
}

```

二、非自顶向下

124. 二叉树中的最大路径和
 /left,right分别为根节点左右子树最大路径和,注意:如果最大路径和<0,意味着该路径和对总路径和做负贡献,因此不要计入到总路径中,将它设置为0

```

int res = INT_MIN; //注意节点值可能为负数,因此要设置为最小值
int maxPathSum(TreeNode *root)
{
    maxPath(root);
    return res;
}

int maxPath(TreeNode *root) //以root为路径起始点的最长路径
{
    if (!root)
        return 0;
    int left = max(maxPath(root->left), 0);
    int right = max(maxPath(root->right), 0);
    res = max(res, left + right + root->val); //比较当前最大路径和与左右子树最长路径加上根节点值的较大值,更新全局变量
    return max(left + root->val, right + root->val); //返回左右子树较长的路径加上根节点值
}

```

687. 最长同值路径

```

int longestUnivaluePath(TreeNode *root)
{
    if (!root)
        return 0;
    longestPath(root);
    return res;
}

int longestPath(TreeNode *root)
{
    if (!root)
        return 0;
    int left = longestPath(root->left), right = longestPath(root->right);
    // 如果存在左子节点和根节点同值,更新左最长路径;否则左最长路径为0
    if (root->left && root->val == root->left->val)
        left++;
    else
        left = 0;
    if (root->right && root->val == root->right->val)
        right++;
    else
        right = 0;
    res = max(res, left + right);
    return max(left, right);
}

```

543. 二叉树的直径

```

int res1 = 0;
int diameterOfBinaryTree(TreeNode *root)
{
    maxPath(root);
    return res1;
}

int maxPath(TreeNode *root)
{
    // 这里递归结束条件要特别注意:不能是!root(而且不需要判断root为空,因为只有非空才会进入递归),因为单个节点路径长也是0
    if (!root->left && !root->right)
        return 0;
    int left = root->left ? maxPath(root->left) + 1 : 0; //判断左子节点是否为空,从而更新左边最长路径
    int right = root->right ? maxPath(root->right) + 1 : 0;
    res1 = max(res, left + right); //更新全局变量
    return max(left, right); //返回左右路径较大者
}

```

以上就是二叉树路径问题的总结, 有收获的话点赞收藏加关注吧

@eh-xing-qing

下一篇: 双递归

© 著作权归作者所有

2

条评论>

最热

请先 登录 后发表评论



offrande L1

(编辑过)2021-06-09

感谢大佬整理, 最近也在整理学习。这就回去写一个python版的伪码来补上。

-----交作业, 注意由于Python语言特性, path.append(roo.val) 会引发错误。!!! 错误的写法 -----

```
...
一般路径
...
res = []
def dfs(root, path):
    if not root:
        return # 根节点为空直接返回
    path.append(root.val)
    if not root.left and not root.right: # 到叶节点
        res.append(path)
        return
    dfs(root.left, path)
    dfs(root.right, path)
    return
```

给定和路径

```
def dfs(root, target, path):
    if not root:
        return
    target -= root.val
    path.append(root.val)
    if not root.left and not root.right and target == 0:
        res.append(path)
        return
    dfs(root.left, target, path)
    dfs(root.right, target, path)
    return
```

非自顶向下

```
res = 0
def maxPath(root):
    if not root:
        return 0
    left = maxPath(root.left)
    right = maxPath(root.right)
    res = max(res, left + right + root.val) # 更新全局变量
    return max(left, right)
```

----- !!! 正确写法 -----

```
...
一般路径
...
res = []
def dfs(root, path):
    if not root:
        return # 根节点为空直接返回
    if not root.left and not root.right: # 到叶节点
        res.append(path + [root.val])
        return
    dfs(root.left, path + [root.val])
    dfs(root.right, path + [root.val])
    return
```

给定和路径

```
def dfs(root, target, path):
    if not root:
        return
    target -= root.val # 不同点
    if not root.left and not root.right and target == 0: # 不同点
        res.append(path + [root.val])
        return
    dfs(root.left, target, path + [root.val])
    dfs(root.right, target, path + [root.val])
    return
```

非自顶向下

...

```
res = 0
def maxPath(root):
    if not root:
        return 0
    left = maxPath(root.left)
    right = maxPath(root.right)
    res = max(res, left + right + root.val) # 更新全局变量
    return max(left, right)
```