

101. 对称二叉树

101. 对称二叉树

难度 简单

👍 720

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 [1, 2, 2, 3, 4, 4, 3] 是对称的。



但是下面这个 [1, 2, 2, null, 3, null, 3] 则不是镜像对称的:



进阶:

你可以运用递归和迭代两种方法解决这个问题吗?

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/*递归写法
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if(!root) return true;
        return dfs(root->left, root->right);
    }
    bool dfs(TreeNode *p, TreeNode *q)
    {
        if(!p || !q) return !q && !p;
```

```

        return p->val == q->val && dfs(p->left,q->right) && dfs(p->right,q->
left);
    }
};
/*****
/*迭代写法*/
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if(!root) return true;
        stack<TreeNode*> left,right;
        auto l = root->left;
        auto r = root->right;
        while(l || r || left.size() || right.size())
        {
            while(l && r)
            {
                left.push(l);
                right.push(r);
                l = l->left;
                r = r->right;
            }

            if(l || r) return false;
            l = left.top(),left.pop();
            r = right.top(),right.pop();

            if(l->val != r->val) return false;
            l = l->right,r = r->left;
        }
        return true;
    }
};

```

102. 二叉树的层序遍历

102. 二叉树的层序遍历

难度 中等

👍 445

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例:

二叉树: [3, 9, 20, null, null, 15, 7] ,



返回其层次遍历结果:

```
[
  [3],
  [9,20],
  [15,7]
]
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;

        if(!root) return res;

        queue<TreeNode*> q;
        q.push(root);

        while(q.size())
        {
            int len = q.size();
            vector<int> level;

            for(int i = 0; i < len; i++)
            {
                auto t = q.front();
                q.pop();
                level.push_back(t->val);
                if(t->left) q.push(t->left);
                if(t->right) q.push(t->right);
            }
        }
    }
};
```




```

        res.push_back(level);
    }
    return res;
}
};

```

103. 二叉树的锯齿形层次遍历

103. 二叉树的锯齿形层次遍历

难度 **中等**  176  收藏  分享  切换为英文  关注  反馈

给定一个二叉树，返回其节点值的锯齿形层次遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

例如：

给定二叉树 [3, 9, 20, null, null, 15, 7]，

```

    3
   / \
  9  20
 /  \
15   7

```

返回锯齿形层次遍历如下：

```

[
  [3],
  [20,9],
  [15,7]
]

```

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> get_val(vector<TreeNode*> level)
    {
        vector<int> res;
        for (auto &u : level)
            res.push_back(u->val);
        return res;
    }

    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        vector<TreeNode*> level;
    }

```

```

    level.push_back(root);
    res.push_back(get_val(level));
    bool zigzag = true;
    while (true)
    {
        vector<TreeNode*> newLevel;
        for (auto &u : level)
        {
            if (u->left) newLevel.push_back(u->left);
            if (u->right) newLevel.push_back(u->right);
        }
        if (newLevel.size())
        {
            vector<int>temp = get_val(newLevel);
            if (zigzag)
                reverse(temp.begin(), temp.end());
            res.push_back(temp);
            level = newLevel;
        }
        else break;
        zigzag = !zigzag;
    }
    return res;
};

```

104. 二叉树的最大深度

104. 二叉树的最大深度

难度 **简单** 504 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例:

给定二叉树 [3, 9, 20, null, null, 15, 7] ,

```

    3
   / \
  9  20
   / \
  15  7

```

返回它的最大深度 3。

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

```



```

    * };
    */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        return root ? max(maxDepth(root->left),maxDepth(root->right)) + 1 : 0;
    }
};

```

105. 从前序与中序遍历序列构造二叉树

105. 从前序与中序遍历序列构造二叉树

难度 中等  414  收藏  分享  切换为英文  关注  反馈

根据一棵树的前序遍历与中序遍历构造二叉树。

注意:

你可以假设树中没有重复的元素。

例如，给出

```

前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]

```

返回如下的二叉树:

```

    3
   / \
  9  20
   / \
  15  7

```

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    unordered_map<int,int> pos;

    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        int n = preorder.size();
        for (int i = 0; i < n; i++)
            pos[inorder[i]] = i;
        return dfs(preorder, inorder, 0, n - 1, 0, n - 1);
    }

    TreeNode* dfs(vector<int>&pre, vector<int>&in, int pl, int pr, int il, int ir)

```

```

    {
        if (p1 > pr) return NULL;
        int k = pos[pre[p1]] - il;
        TreeNode* root = new TreeNode(pre[p1]);
        root->left = dfs(pre, in, p1 + 1, p1 + k, il, il + k - 1);
        root->right = dfs(pre, in, p1 + k + 1, pr, il + k + 1, ir);
        return root;
    }
};

```

106. 从中序与后序遍历序列构造二叉树

106. 从中序与后序遍历序列构造二叉树

难度 中等

186

收藏

分享

切换为英文

关注

反馈

根据一棵树的中序遍历与后序遍历构造二叉树。

注意:

你可以假设树中没有重复的元素。

例如, 给出

```

中序遍历 inorder = [9,3,15,20,7]
后序遍历 postorder = [9,15,7,20,3]

```

返回如下的二叉树:

```

    3
   / \
  9  20
   / \
  15  7

```

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    unordered_map<int,int> pos;

    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        int n = inorder.size();
        for (int i = 0; i < n; i++)
            pos[inorder[i]] = i;
        return dfs(postorder, inorder, 0, n - 1, 0, n - 1);
    }
}

```

```

// pl, pr 表示当前子树后序遍历在数组中的位置
// il, ir 表示当前子树中序遍历在数组中的位置
TreeNode* dfs(vector<int>&post, vector<int>&in, int pl, int pr, int il, int
ir)
{
    if (pl > pr) return NULL;
    int k = pos[post[pr]] - il;
    TreeNode* root = new TreeNode(post[pr]);
    root->left = dfs(post, in, pl, pl + k - 1, il, il + k - 1);
    root->right = dfs(post, in, pl + k, pr - 1, il + k + 1, ir);
    return root;
}
};

```

107. 二叉树的层次遍历 II

107. 二叉树的层次遍历 II

难度 简单  223  收藏  分享  切换为英文  关注  反馈

给定一个二叉树，返回其节点值自底向上的层次遍历。（即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历）

例如：

给定二叉树 [3, 9, 20, null, null, 15, 7]，

```

    3
   / \
  9  20
   / \
  15  7

```

返回其自底向上的层次遍历为：

```

[
  [15,7],
  [9,20],
  [3]
]

```

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> get_val(vector<TreeNode*> level)
    {
        vector<int> res;
        for (auto &u : level)
            res.push_back(u->val);
    }
};

```



```

        return res;
    }

    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;
        vector<TreeNode*> level;
        level.push_back(root);
        res.push_back(get_val(level));
        while (true)
        {
            vector<TreeNode*> newLevel;
            for (auto &u : level)
            {
                if (u->left) newLevel.push_back(u->left);
                if (u->right) newLevel.push_back(u->right);
            }
            if (newLevel.size())
            {
                res.push_back(get_val(newLevel));
                level = newLevel;
            }
            else break;
        }
        reverse(res.begin(), res.end());
        return res;
    }
};

```

108. 将有序数组转换为二叉搜索树

108. 将有序数组转换为二叉搜索树

难度 **简单** 388 收藏 分享 切换为英文 关注 反馈

将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点的左右两个子树的高度差的绝对值不超过 1。

示例:

给定有序数组: [-10,-3,0,5,9],

一个可能的答案是: [0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：

```

      0
     / \
    -3  9
   /   /
  -10  5

```

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;

```

```



*     TreeNode *right;
*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return build(0, nums.size() - 1, nums);
    }

    TreeNode *build(int l, int r, vector<int>&nums)
    {
        if (l > r) return 0;
        int mid = (l + r) / 2;
        TreeNode *root = new TreeNode(nums[mid]);
        root->left = build(l, mid - 1, nums);
        root->right = build(mid + 1, r, nums);
        return root;
    }
};

```

109. 有序链表转换二叉搜索树

109. 有序链表转换二叉搜索树

难度 中等  186  收藏  分享  切换为英文  关注  反馈

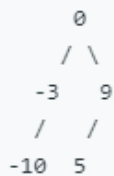
给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点的左右两个子树的高度差的绝对值不超过 1。

示例:

给定的有序链表： [-10, -3, 0, 5, 9],

一个可能的答案是：[0, -3, 9, -10, null, 5]，它可以表示下面这个高度平衡二叉搜索树：



```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;

```

```

*   TreeNode *right;
*   TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {
        if (!head) return 0;
        int l = 0;
        for (auto i = head; i; i = i->next) l ++ ;
        l /= 2;

        ListNode*p = head;
        for (int i = 0; i < l; i ++ ) p = p->next;
        TreeNode *root = new TreeNode(p->val);

        root->right = sortedListToBST(p->next);
        if (l)
        {
            p = head;
            for (int i = 0; i < l - 1; i ++ ) p = p->next;
            p->next = 0;
            root->left = sortedListToBST(head);
        }
        return root;
    }
};

```

110. 平衡二叉树

110. 平衡二叉树

难度 简单

👍 289

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

🗉 反馈

给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

一个二叉树每个节点的左右两个子树的高度差的绝对值不超过1。

示例 1:

给定二叉树 [3, 9, 20, null, null, 15, 7]

```
    3
   / \
  9  20
   / \
  15  7
```

返回 true 。

示例 2:

给定二叉树 [1, 2, 2, 3, 3, null, null, 4, 4]

```
    1
   / \
  2   2
 / \   \
3  3   3
/ \   \
4  4   4
```

返回 false 。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        int h;
        return dfs(root, h);
    }

    bool dfs(TreeNode* root, int &height)
    {
        if (!root)
        {
            height = 0;
            return true;
        }
    }
}
```

```

    }

    int left, right;
    if (!dfs(root->left, left)) return false;
    if (!dfs(root->right, right)) return false;

    height = max(left, right) + 1;

    return abs(left - right) <= 1;
}
};

```

111. 二叉树的最小深度

111. 二叉树的最小深度

难度 **简单** 242 收藏 分享 切换为英文 关注 反馈

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明: 叶子节点是指没有子节点的节点。

示例:

给定二叉树 [3, 9, 20, null, null, 15, 7] ,

```

    3
   / \
  9  20
   / \
  15  7

```

返回它的最小深度 2.

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (!root) return 0;
        int res = INT_MAX;
        if (root->left) res = min(res, minDepth(root->left) + 1);
        if (root->right) res = min(res, minDepth(root->right) + 1);
        if (res == INT_MAX) res = 1;
        return res;
    }
};

```

112. 路径总和

112. 路径总和

难度 简单 277 收藏 分享 切换为英文 关注 反馈

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`。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (!root) return false;
        if (!root->left && !root->right) return root->val == sum;
        if (root->left && hasPathSum(root->left, sum - root->val)) return true;
        if (root->right && hasPathSum(root->right, sum - root->val)) return
true;
        return false;
    }
};
```

113. 路径总和 II

113. 路径总和 II

难度 中等

👍 204

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

给定一个二叉树和一个目标和，找到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。

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:

```
[
  [5,4,11,2],
  [5,8,4,5]
]
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;

    vector<vector<int>> pathSum(TreeNode* root, int sum)
    {
        if (!root) return ans;
        dfs(root, sum);
        return ans;
    }

    void dfs(TreeNode* root, int sum)
    {
        path.push_back(root->val);
        sum -= root->val;
        if (root->left || root->right)
        {
            if (root->left) dfs(root->left, sum);
            if (root->right) dfs(root->right, sum);
        }
        else
            ans.push_back(path);
        path.pop_back();
    }
};
```



```

        {
            if (!sum)
            {
                ans.push_back(path);
            }
        }
        path.pop_back();
    }
};

```

114. 二叉树展开为链表

114. 二叉树展开为链表

难度 中等  315  收藏  分享  切换为英文  关注  反馈

给定一个二叉树，原地将它展开为链表。

例如，给定二叉树



将其展开为：



```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void flatten(TreeNode* root) {
        TreeNode *now = root;
        while (now)
        {

```



```

        if (now->left)
        {
            TreeNode *p = now->left;
            while (p->right) p = p->right;
            p->right = now->right;
            now->right = now->left;
            now->left = 0;
        }
        now = now->right;
    }
}
};

```

115. 不同的子序列

115. 不同的子序列

难度 **困难**  166  收藏  分享  切换为英文  关注  反馈

给定一个字符串 **S** 和一个字符串 **T**，计算在 **S** 的子序列中 **T** 出现的个数。

一个字符串的一个子序列是指，通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所组成的新字符串。（例如，“ACE” 是 “ABCDE” 的一个子序列，而 “AEC” 不是）

示例 1:

输入: S = "rabbbit", T = "rabbit"
 输出: 3
 解释:

如下图所示，有 3 种可以从 S 中得到 "rabbit" 的方案。
 (上箭头符号 ^ 表示选取的字母)

```

rabbbit
^^^ ^^

rabbbit
^^ ^^^^

rabbbit
^^^ ^^^

```

示例 2:

输入: S = "babgbag", T = "bag"
 输出: 5
 解释:

如下图所示，有 5 种可以从 S 中得到 "bag" 的方案。
 (上箭头符号 ^ 表示选取的字母)

```

babgbag
^^ ^

babgbag
^^   ^

```

```

class Solution {

```

```

public:
    int numDistinct(string s, string t) {
        int n = s.size(), m = t.size();
        vector<vector<long long>> f(n + 1, vector<long long>(m + 1));

        for (int i = 0; i <= n; i++) f[i][0] = 1;

        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
            {
                f[i][j] = f[i - 1][j];
                if (s[i - 1] == t[j - 1])
                    f[i][j] += f[i - 1][j - 1];
            }

        return f[n][m];
    }
};

```

116. 填充每个节点的下一个右侧节点指针

116. 填充每个节点的下一个右侧节点指针

难度 中等

152

收藏

分享

切换为英文

关注

反馈

给定一个完美二叉树，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

```
struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}
```

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

示例：

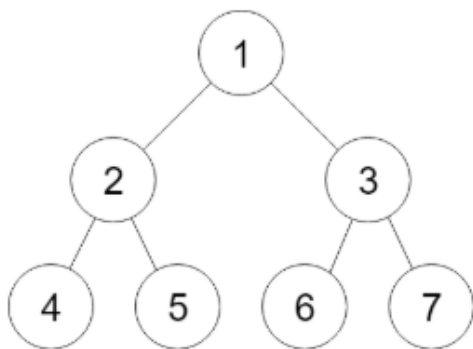


Figure A

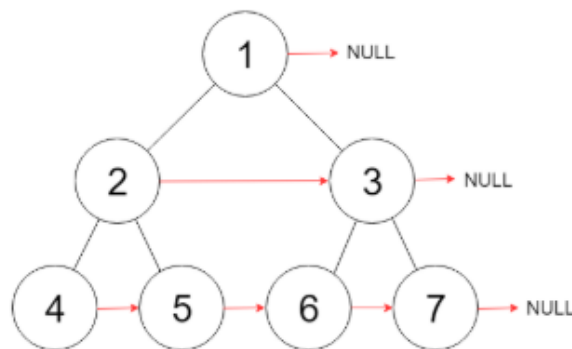


Figure B

输入：{"\$id": "1", "left": {"\$id": "2", "left": {"\$id": "3", "left": null, "next": null, "right": null, "val": 4}, "next": null, "right": {"\$id": "4", "left": null, "next": null, "right": null, "val": 5}, "val": 2}, "next": null, "right": {"\$id": "5", "left": {"\$id": "6", "left": null, "next": null, "right": null, "val": 6}, "next": null, "right": {"\$id": "7", "left": null, "next": null, "right": null, "val": 7}, "val": 3}, "val": 1}}

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *     int val;
 *     TreeLinkNode *left, *right, *next;
 *     TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root)
    {
        if (!root) return;
        TreeLinkNode *last = root;
        while (last->left) // 直到遍历到叶节点
        {
            for (TreeLinkNode *p = last; p; p = p->next)
```


```

    {
        p->left->next = p->right;
        if (p->next) p->right->next = p->next->left;
        else p->right->next = 0;
    }
    last = last->left;
}
}
};

```

117. 填充每个节点的下一个右侧节点指针 II

117. 填充每个节点的下一个右侧节点指针 II

难度 中等  131  收藏  分享  切换为英文  关注  反馈

给定一个二叉树

```

struct Node {
    int val;
    Node *left;
    Node *right;
    Node *next;
}

```

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

进阶：

- 你只能使用常量级额外空间。
- 使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

示例：

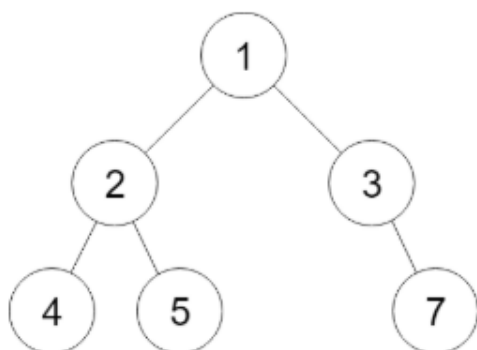


Figure A

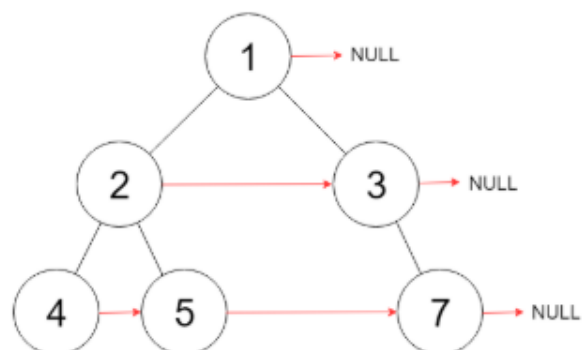


Figure B

```

/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *     int val;
 *     TreeLinkNode *left, *right, *next;

```

```

*   TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
* };
*/
class Solution {
public:
    void connect(TreeLinkNode *root) {
        while (root)
        {
            TreeLinkNode *dummy = new TreeLinkNode(0);
            TreeLinkNode *tail = dummy;
            while (root)
            {
                if (root->left)
                {
                    tail->next = root->left;
                    tail = tail->next;
                }
                if (root->right)
                {
                    tail->next = root->right;
                    tail = tail->next;
                }
                root = root->next;
            }
            tail->next = 0;
            root = dummy->next;
        }
    };
};

```

118. 杨辉三角

118. 杨辉三角

难度 简单

282

收藏

分享

切换为英文

关注

反馈

给定一个非负整数 *numRows*，生成杨辉三角的前 *numRows* 行。



在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

```
输入: 5
输出:
[
  [1],
  [1,1],
  [1,2,1],
  [1,3,3,1],
  [1,4,6,4,1]
]
```

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> res;
        if (!numRows) return res;
        res.push_back(vector<int>(1, 1));
        for (int i = 1; i < numRows; i++)
        {
            vector<int> &last = res.back();
            vector<int> temp;
            temp.push_back(1);
            for (int i = 0; i + 1 < last.size(); i++)
                temp.push_back(last[i] + last[i + 1]);
            temp.push_back(1);
            res.push_back(temp);
        }
        return res;
    }
};
```

119. 杨辉三角 II

119. 杨辉三角 II

难度 简单

139

收藏

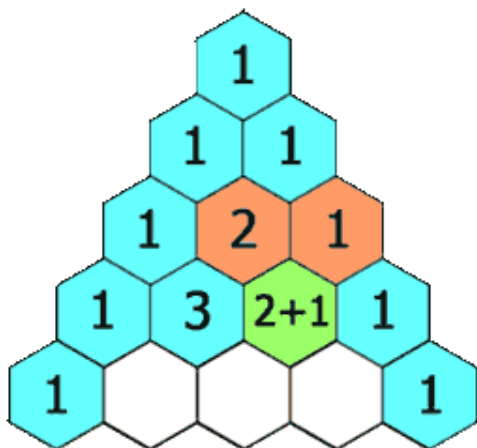
分享

切换为英文

关注

反馈

给定一个非负索引 k ，其中 $k \leq 33$ ，返回杨辉三角的第 k 行。



在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

输入: 3

输出: [1,3,3,1]

进阶:

你可以优化你的算法到 $O(k)$ 空间复杂度吗?

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        vector<int> last(rowIndex + 1), now(rowIndex + 1);
        last[0] = 1;
        for (int i = 0; i < rowIndex; i++)
        {
            now[0] = last[0];
            for (int j = 0; j + 1 <= i; j++)
                now[j + 1] = last[j] + last[j + 1];
            now[i + 1] = last[i];
            last = now;
        }
        return last;
    }
};
```

120. 三角形最小路径和

120. 三角形最小路径和

难度 中等

👍 364

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。

例如，给定三角形：

```
[
  [2],
  [3,4],
  [6,5,7],
  [4,1,8,3]
]
```

自顶向下的最小路径和为 11（即， $2 + 3 + 5 + 1 = 11$ ）。

说明：

如果你可以只使用 $O(n)$ 的额外空间（ n 为三角形的总行数）来解决这个问题，那么你的算法会很加分。

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        int n = triangle.size();
        vector<vector<long long>> f(n, vector<long long>(n));
        f[0][0] = triangle[0][0];
        for(int i = 1; i < n; i++)
            for(int j = 0; j <= i; j++)
            {
                f[i][j] = INT_MAX;
                if(j > 0) f[i][j] = min(f[i][j], f[i-1][j-1] + triangle[i][j]); // 可以从左边更新
                if(j < i) f[i][j] = min(f[i][j], f[i-1][j] + triangle[i][j]); // 可以从右边更新
            }
        long long res = INT_MAX;
        for(int i = 0; i < n; i++) res = min(res, f[n-1][i]); // 枚举最后一行的状态
        return res;
    }
};
```