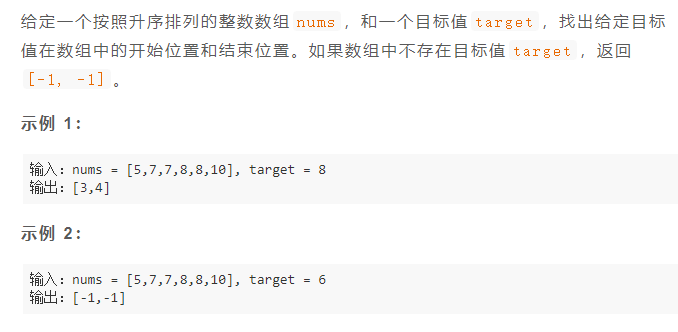
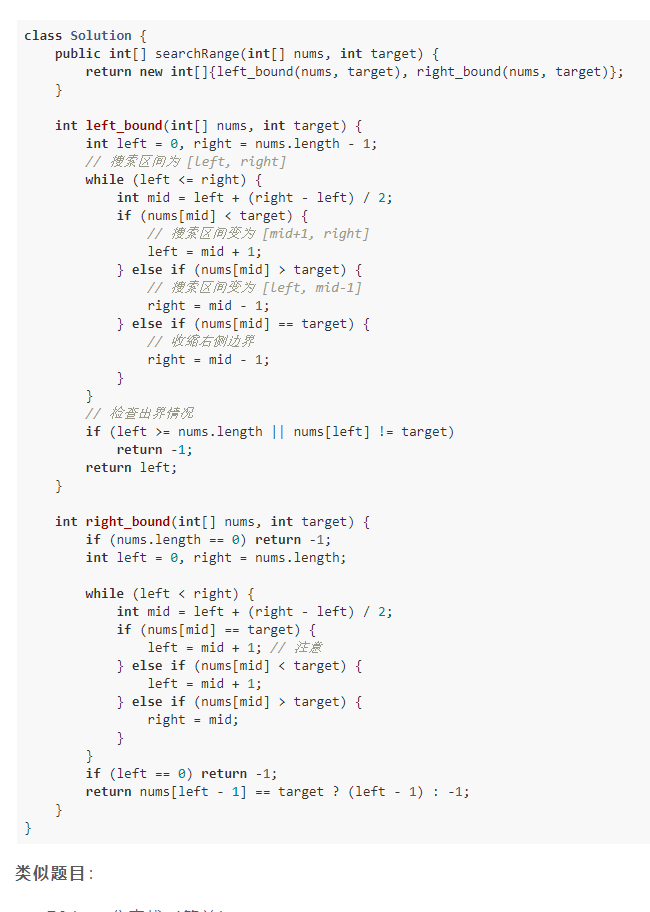
## 二分查找



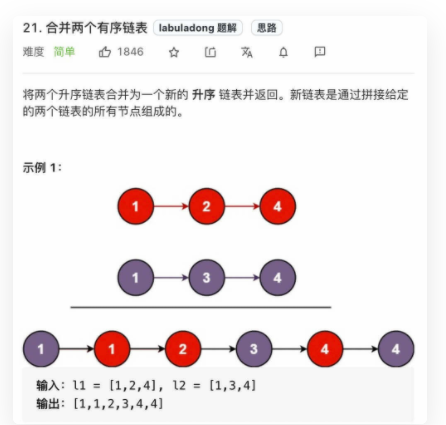
## 在排序数组中查找元素的第一个和最后一个位置

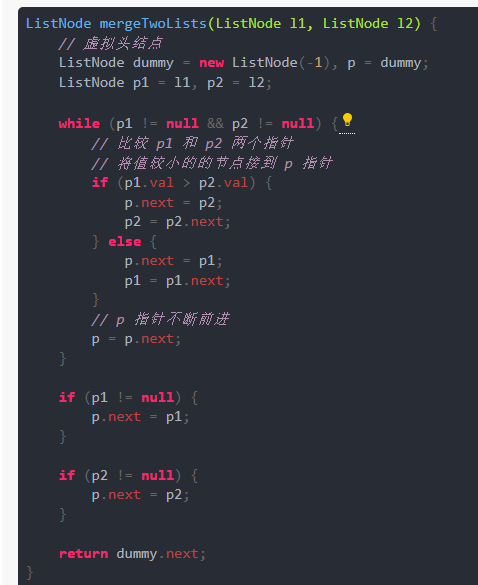




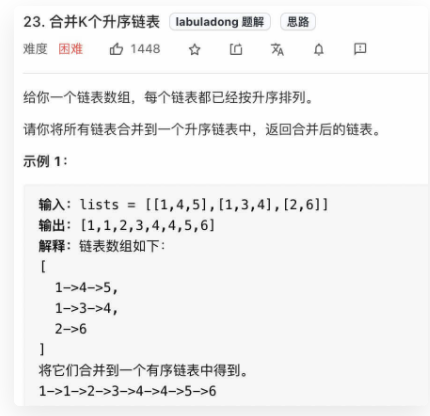
# 链表

## -合并两个有序链表(假头法)





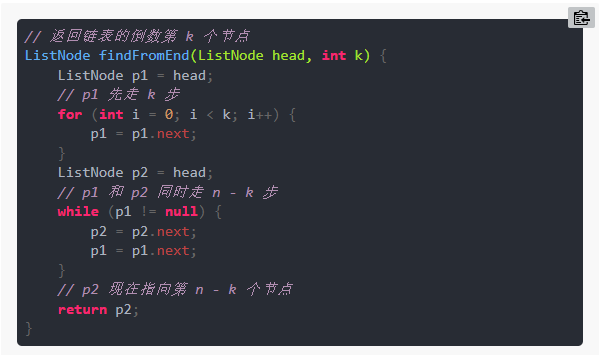
## 合并K个有序链表



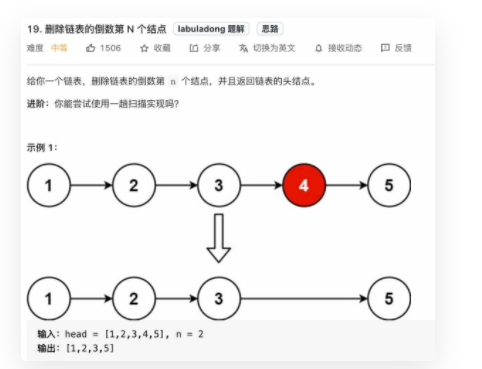


## 双指针

### 单链表倒数第k个节点



### [删除链表的倒数第 N 个结点](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/)



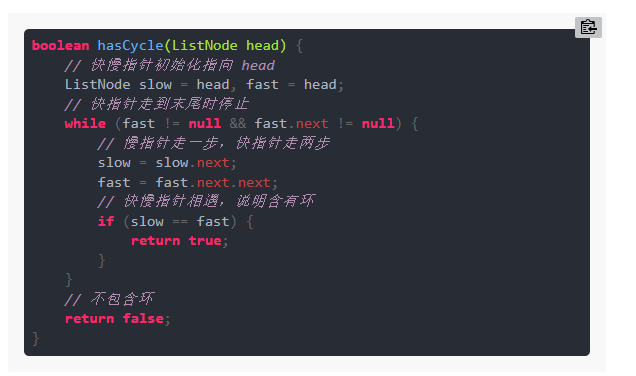


## 快慢指针

### 单链表的中点



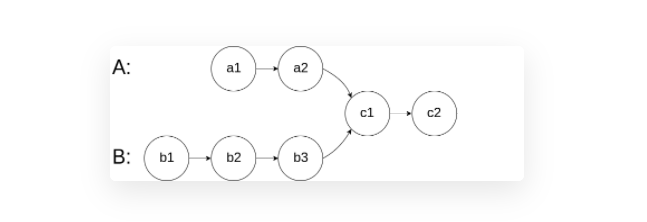
### 判断链表是否有环

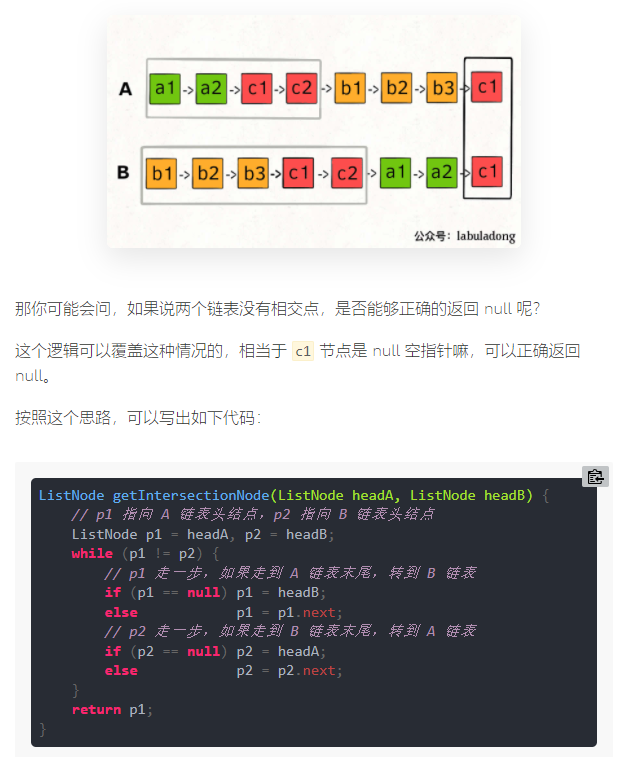


### 判断环起点



### 两个链表是否相交





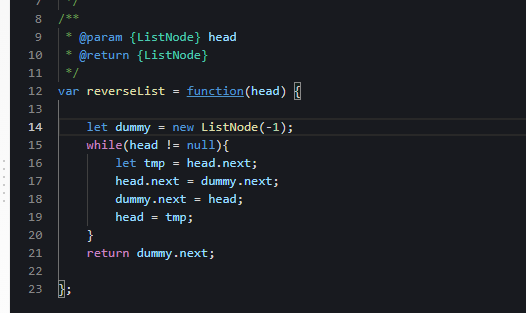




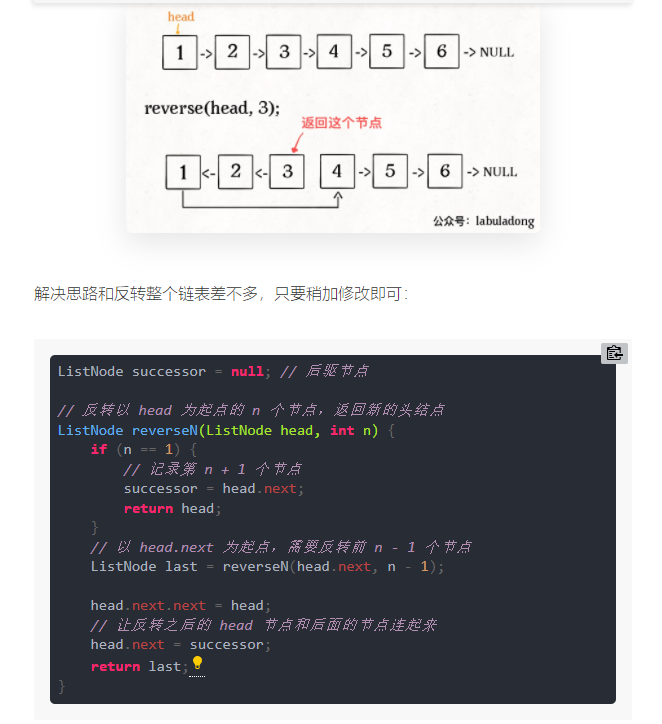
## 反转链表

### 递归反转整个链表

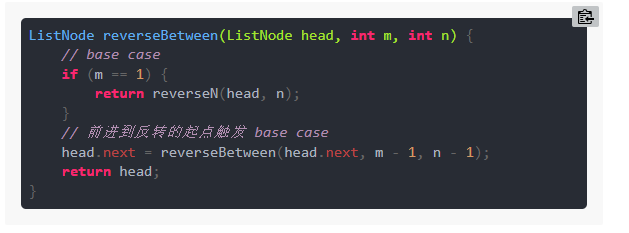




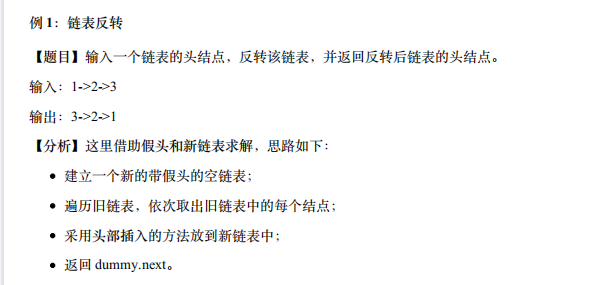
### 反转链表前N个节点

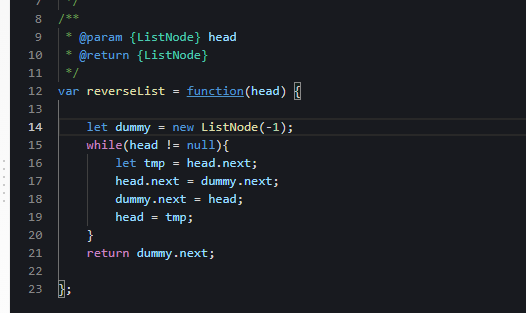


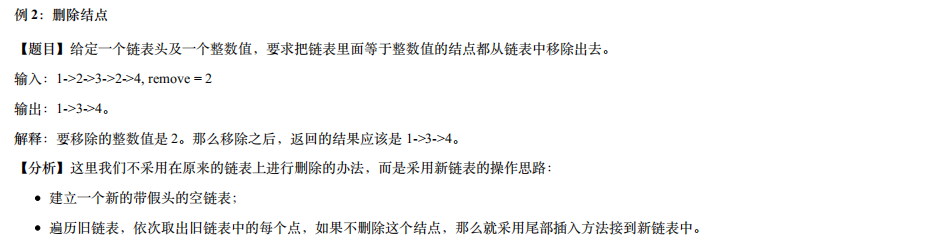
### 反转链表的一部分

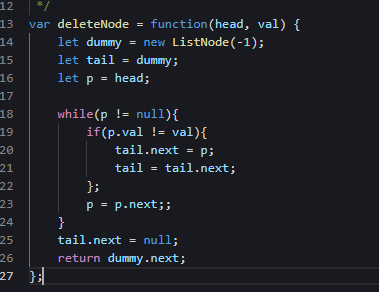


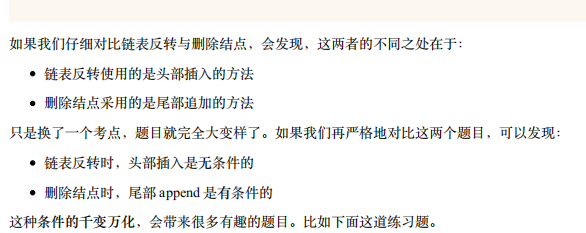
## 假头节点插入





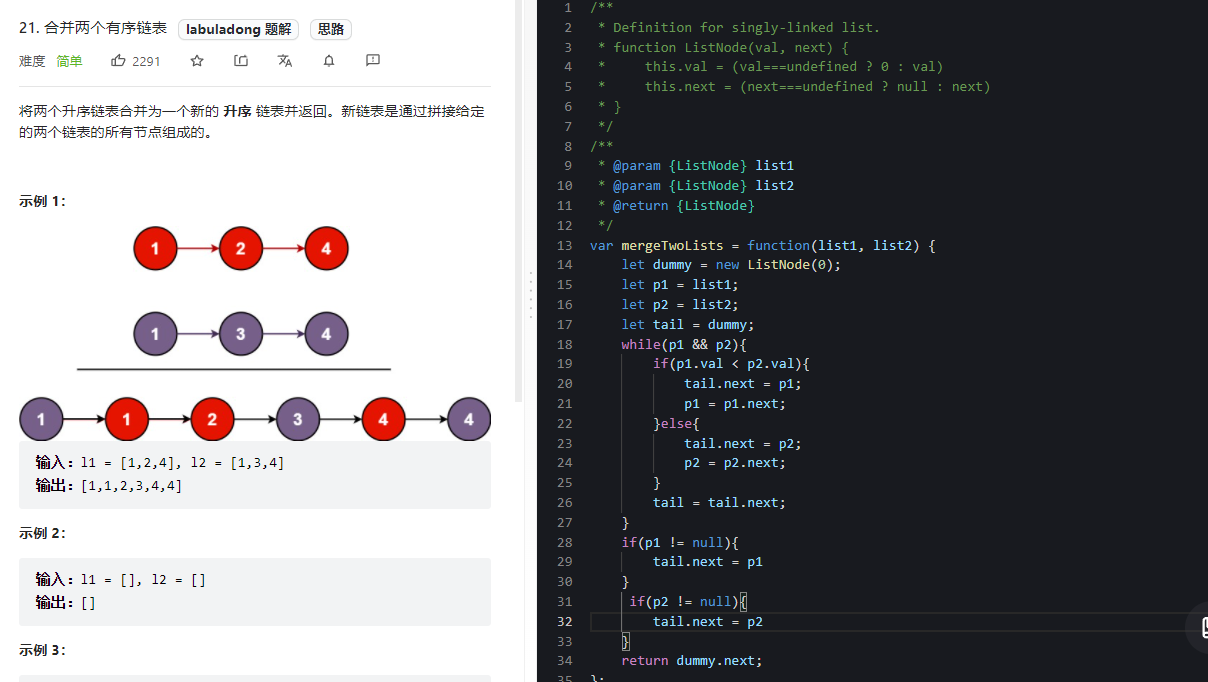




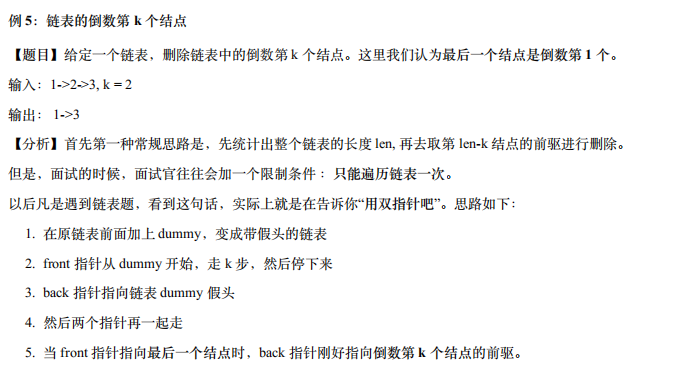


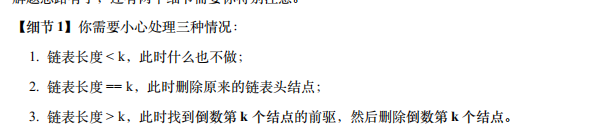






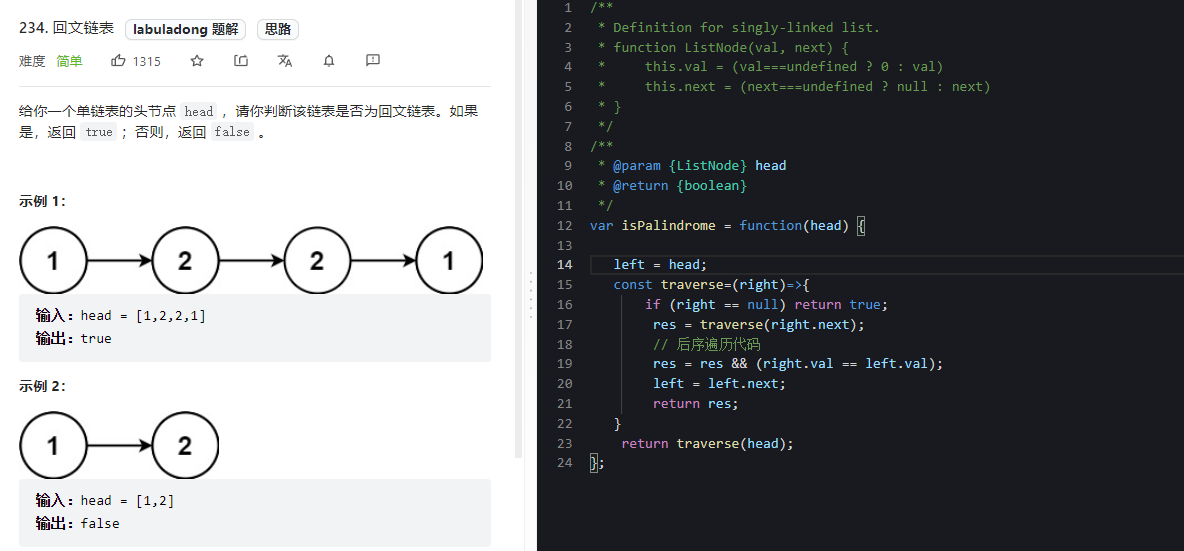
## 双指针





## 回文链表

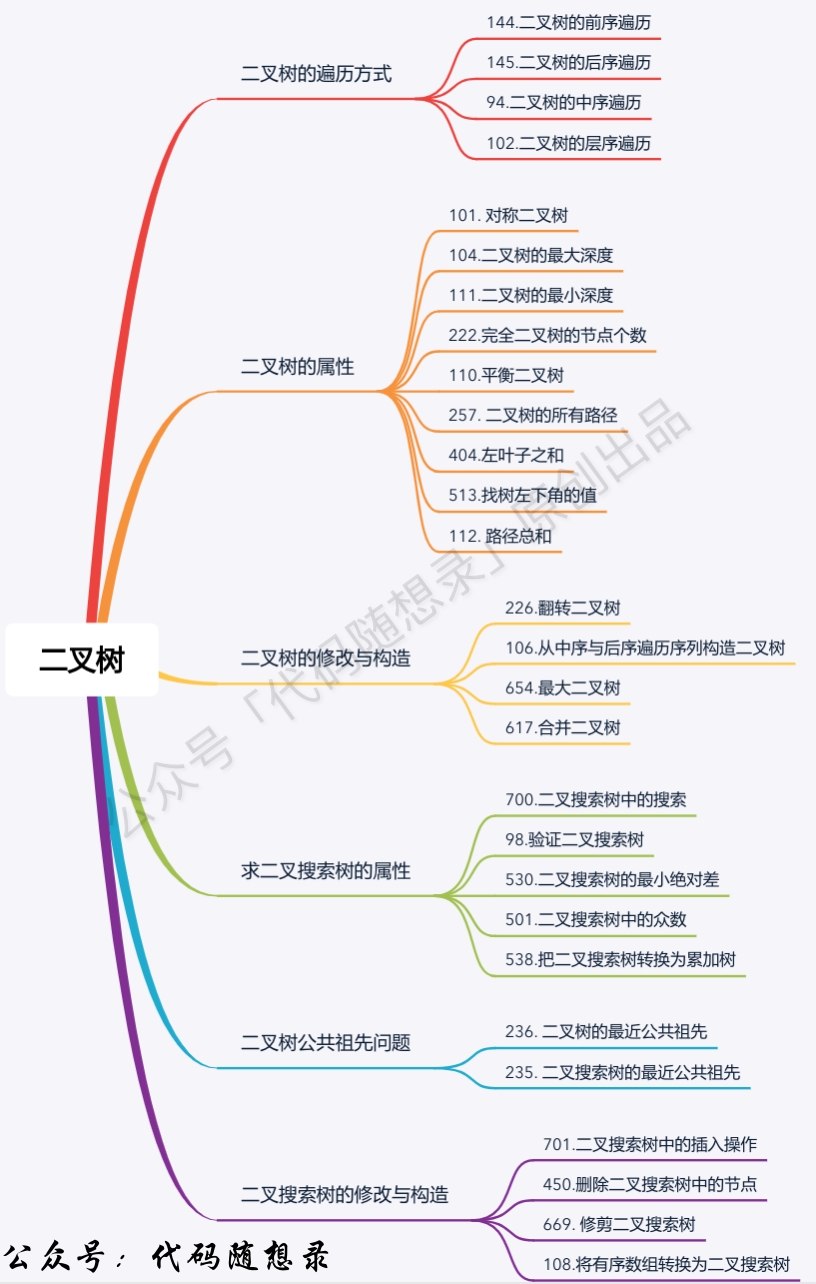
递归

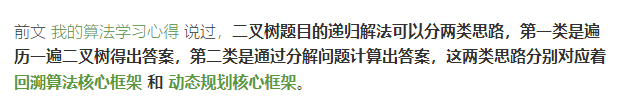


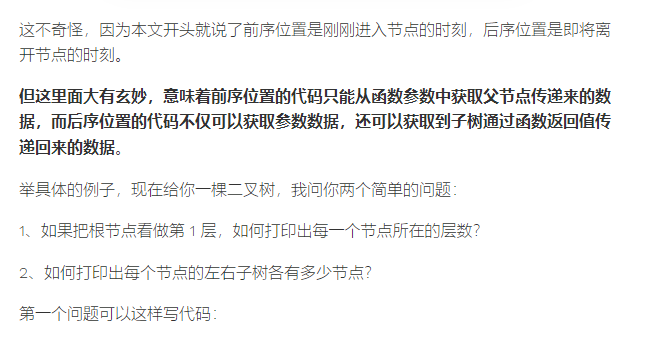
再造一个反向链表



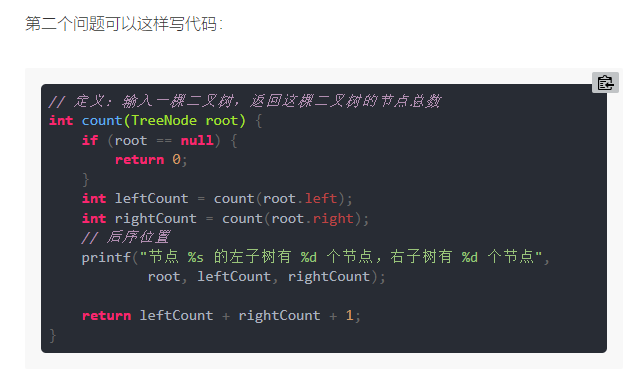
# 二叉树









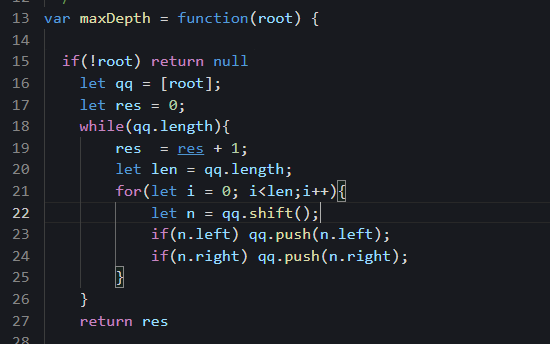


## 最大深度



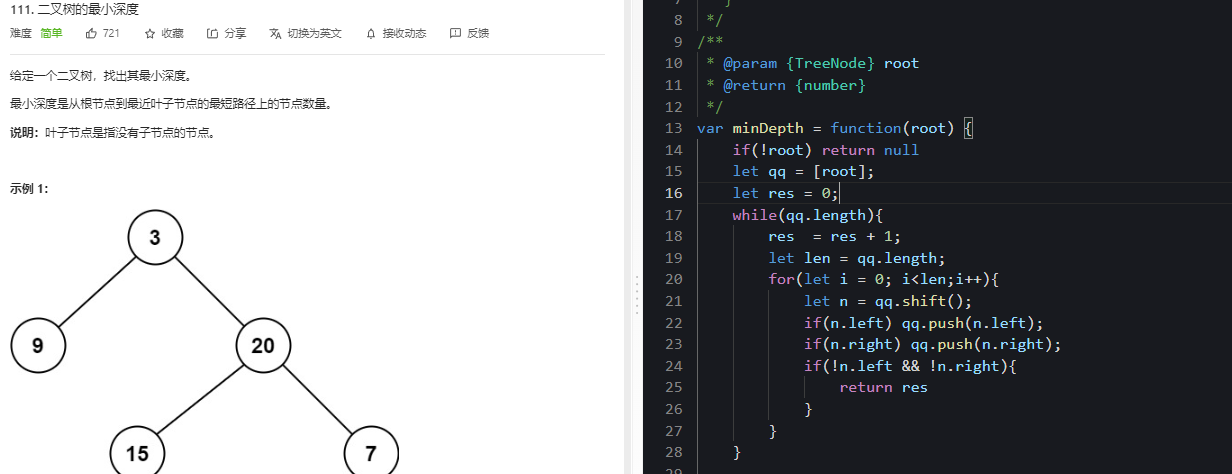


层序遍历



## 最小深度

、、层序遍历



## 二叉树最大直径

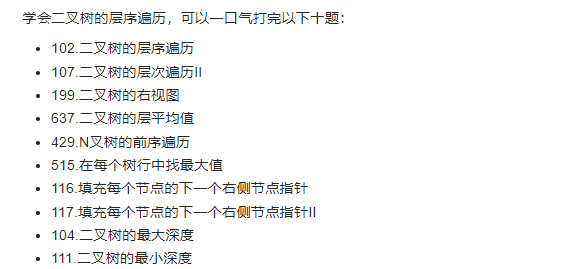
前置遍历

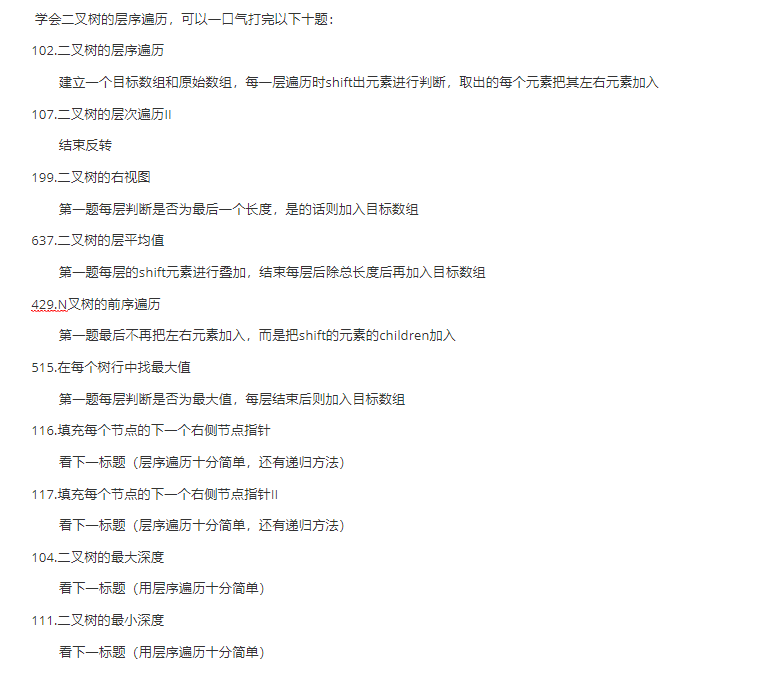


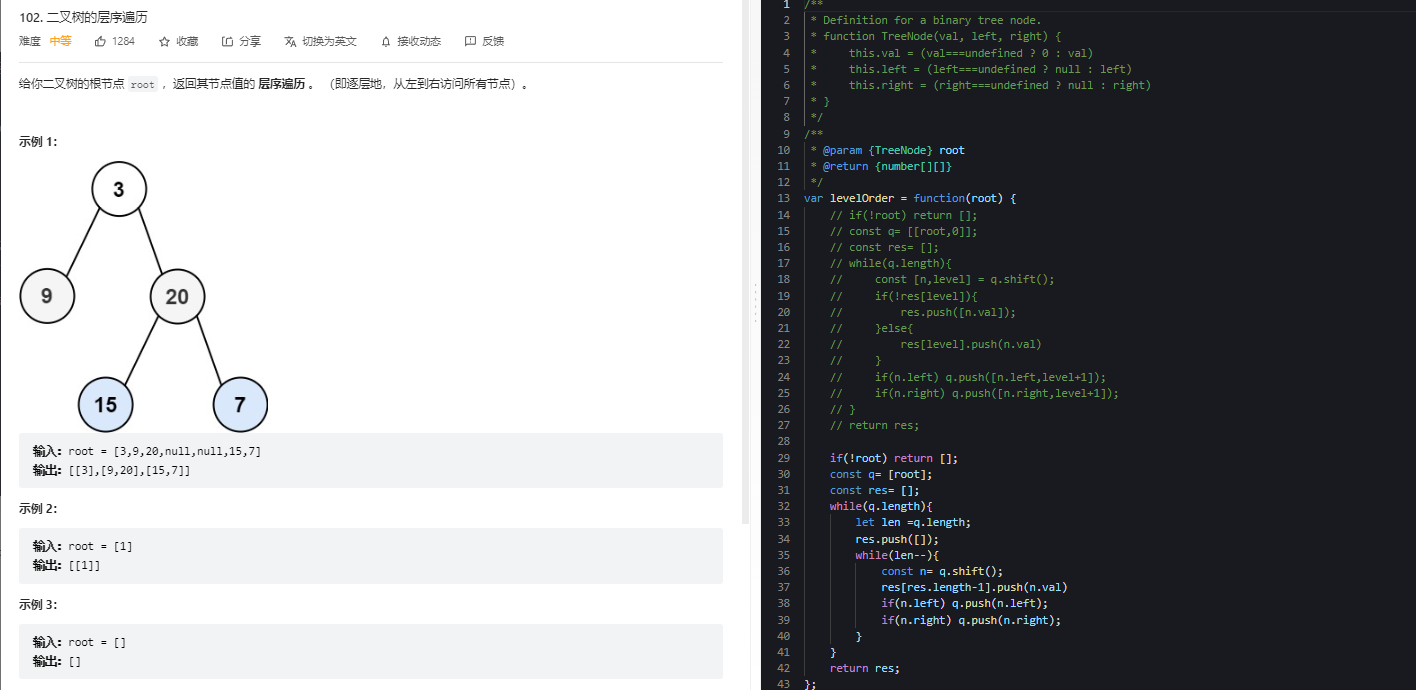
后置遍历

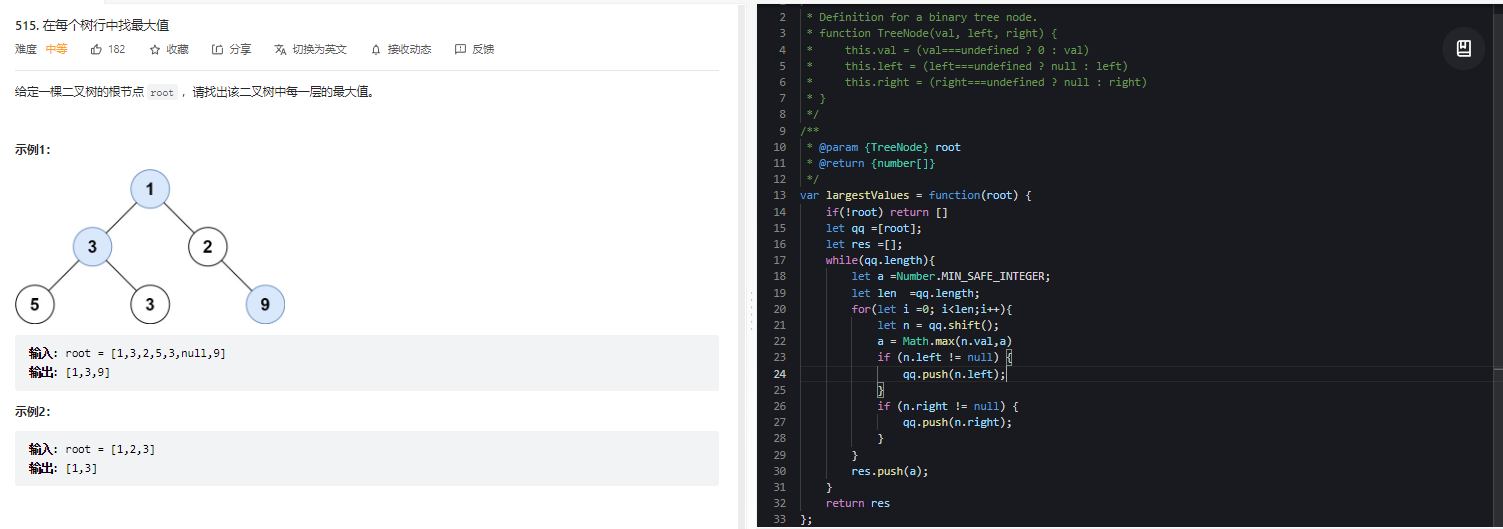


## 层序遍历(10个)

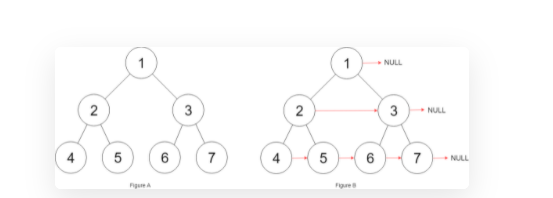


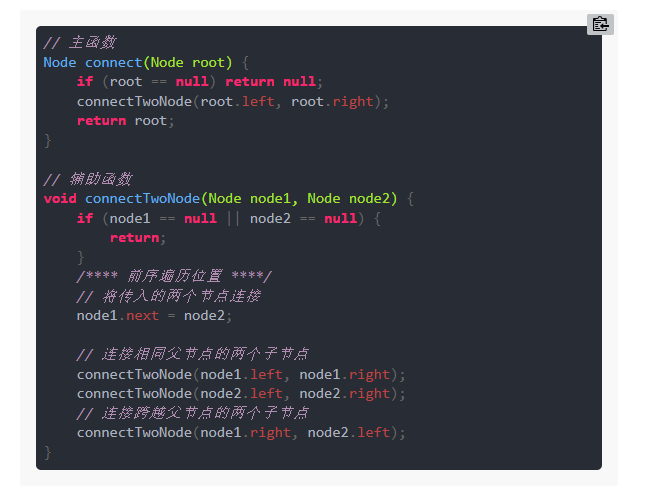






## 填充二叉树节点的右侧指针





层序遍历

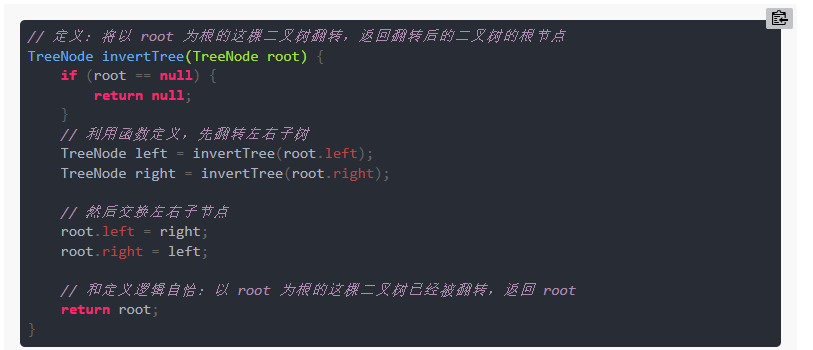


## 翻转二叉树

遍历



递归

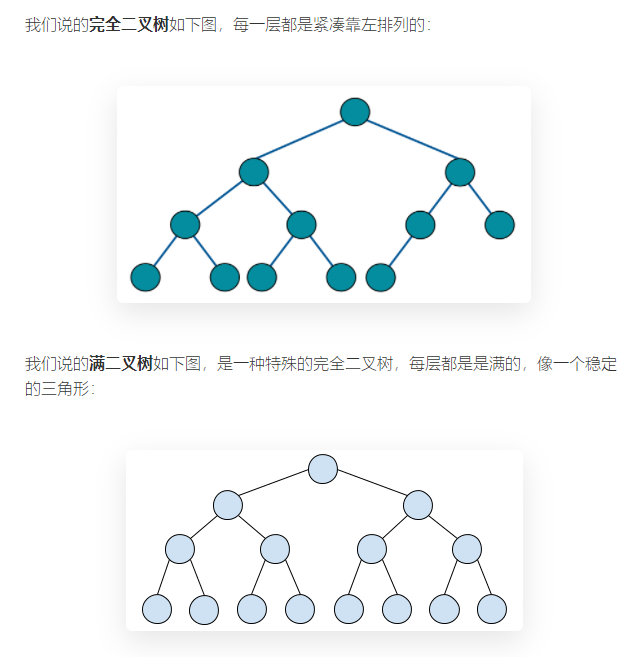


## 二叉树展开为链表

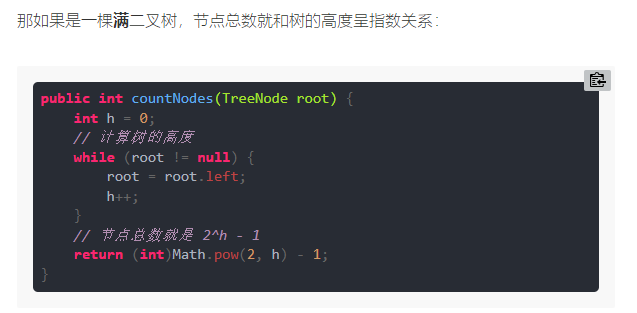


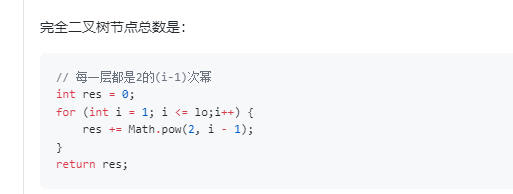
## 序列化与反序列化

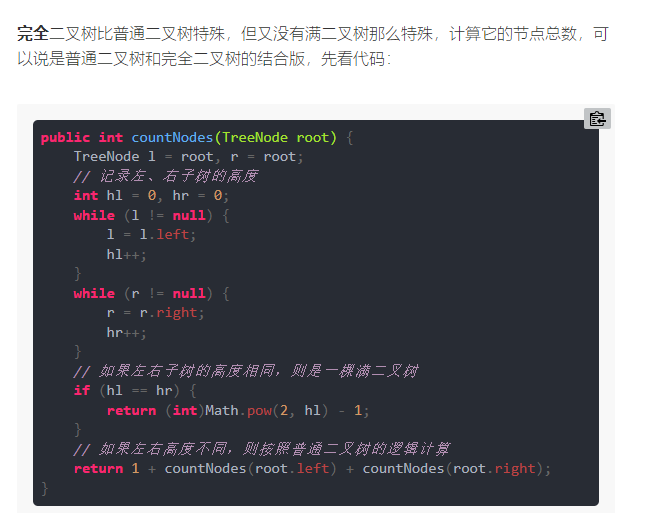
# 完全二叉树











# 二叉搜索树



## 验证二叉搜索树



## 寻找第 K 小的元素



## BST 转化累加树

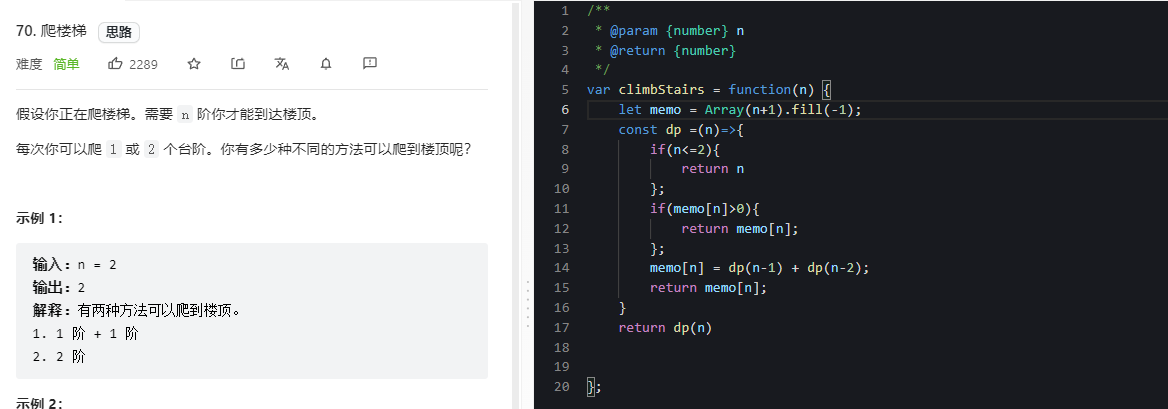


# 动态规划

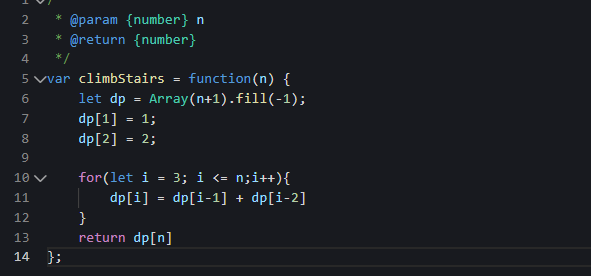
## -----一维

## 爬楼梯

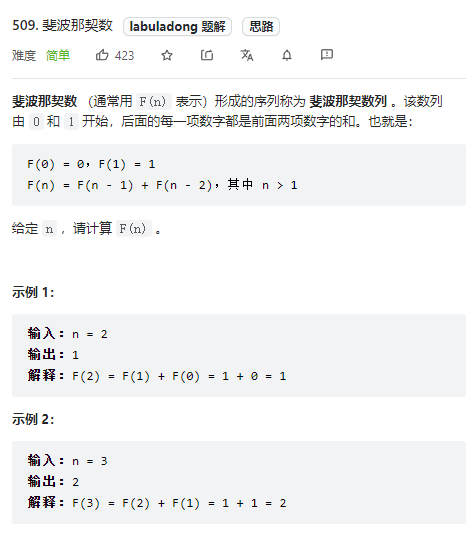
自顶向下



自底向上



## 斐波那契数列



   if(n == 0 || n == 1) return n;

    let dp = Array(n+1).fill(0);

    dp[0] = 0;

    dp[1] = 1;

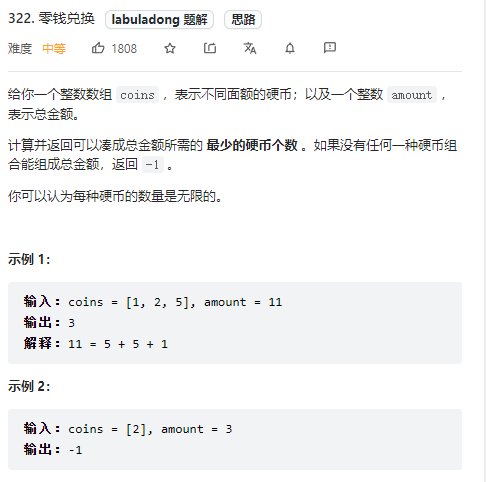
    for(let i = 2; i <= n;i++){

        dp[i] = dp[i-1] + dp[i-2]

    }

    return dp[n]

## 零钱兑换



var coinChange = function(coins, amount) {

    // 数组大小为 amount + 1，初始值也为 amount + 1

    let dp = new Array(amount + 1).fill(amount + 1);

    // base case

    dp[0] = 0

    // 外层 for 循环在遍历所有状态的所有取值

    for (let i = 0; i < dp.length; i++) {

         // 内层 for 循环在求所有选择的最小值

        for (let coin of coins) {

            //子问题无解，跳过

            if (i - coin < 0) {

                continue;

            }

            dp[i] = Math.min(dp[i], dp[i - coin] + 1)

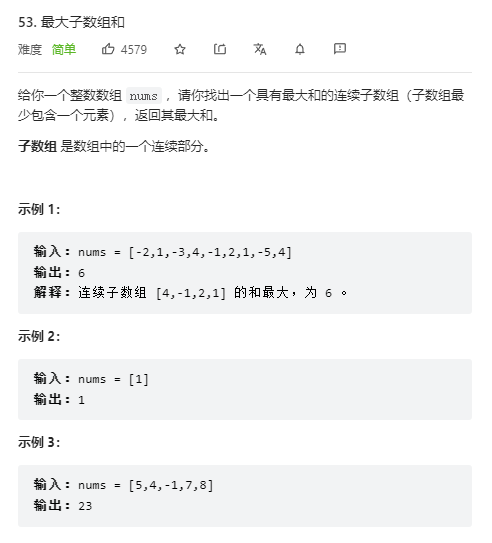
        }

    }

    return (dp[amount] == amount + 1) ? -1 : dp[amount];

};

## 最大子数组和



var maxSubArray = function(nums) {

    let dp =Array(nums.length);

    dp[0] = nums[0];

    for(let i = 1;i<nums.length; i++){

        dp[i] = Math.max(nums[i],dp[i-1] + nums[i])

    }

    let res = Number.MIN\_SAFE\_INTEGER;

    for (let i = 0; i < nums.length; i++) {

        res = Math.max(res, dp[i]);

    }

    return res;

};

## 最长递增子序列



var lengthOfLIS = function(nums) {

    let dp = Array(nums.length).fill(1);

    for(let i = 0; i<nums.length; i++){

        for(let j = 0;j<i; j++){

            if(nums[j] < nums[i]){

                dp[i] = Math.max(dp[i],dp[j] + 1)

            }

        }

    };

    let res = 0;

    for (let i = 0; i < dp.length; i++) {

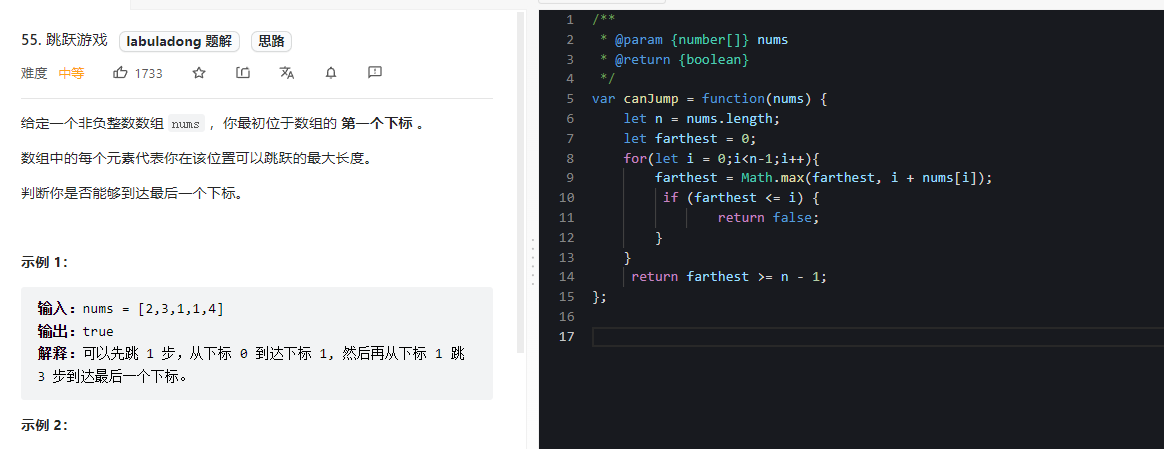
        res = Math.max(res, dp[i]);

    }

    return res;

};

## 跳跃游戏

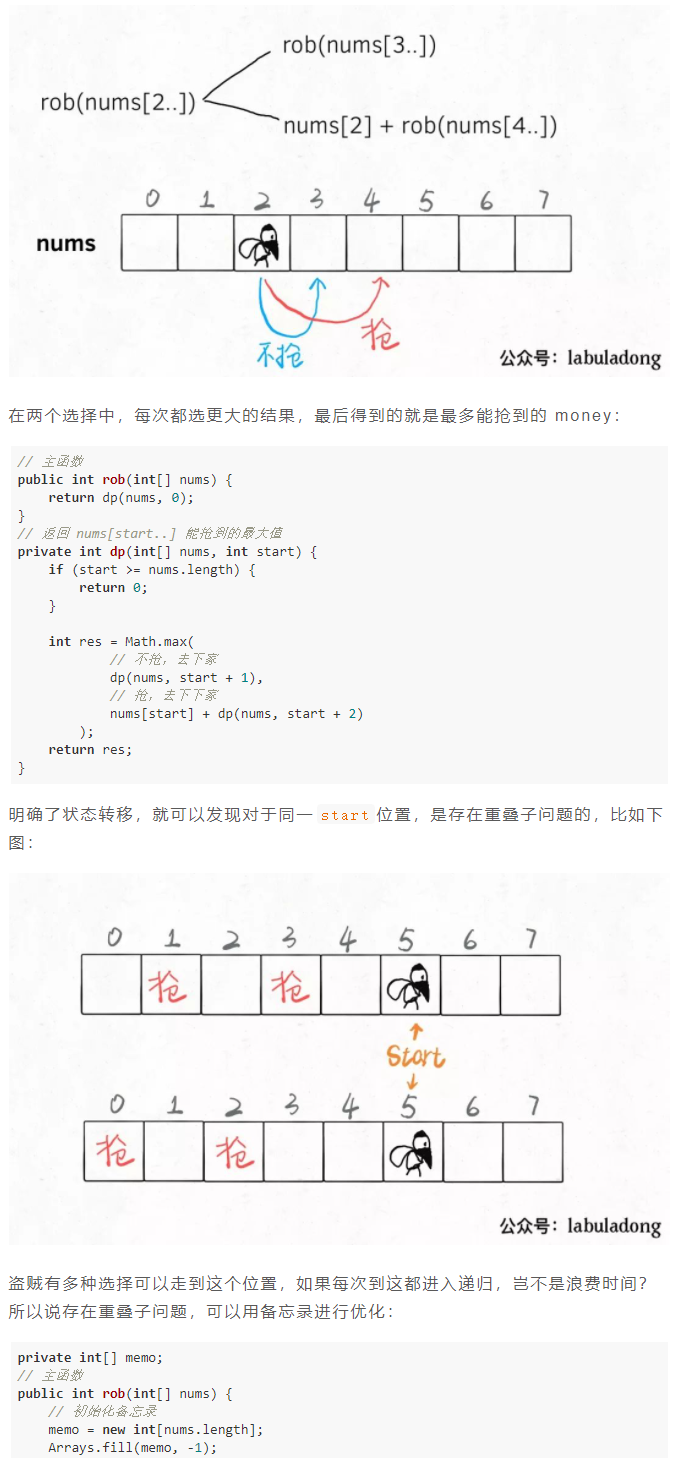


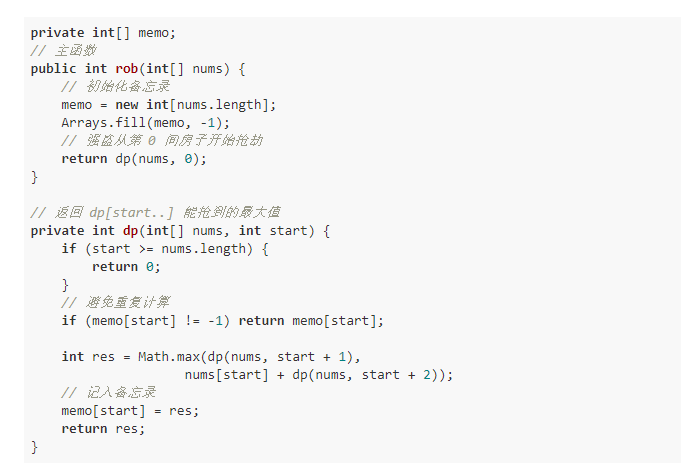
## 跳跃游戏2

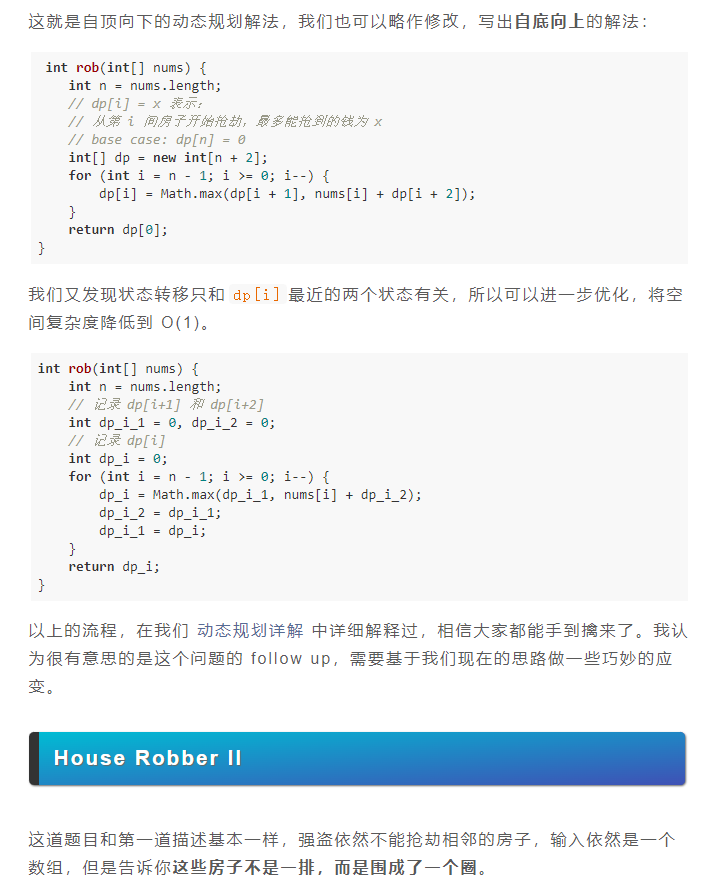


## 打家劫舍

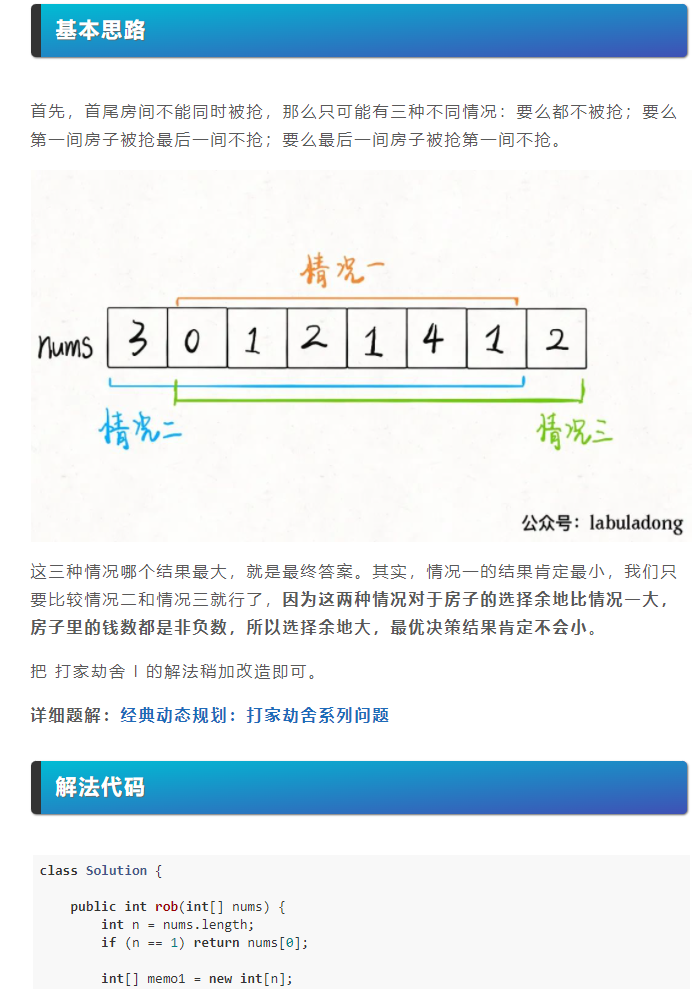


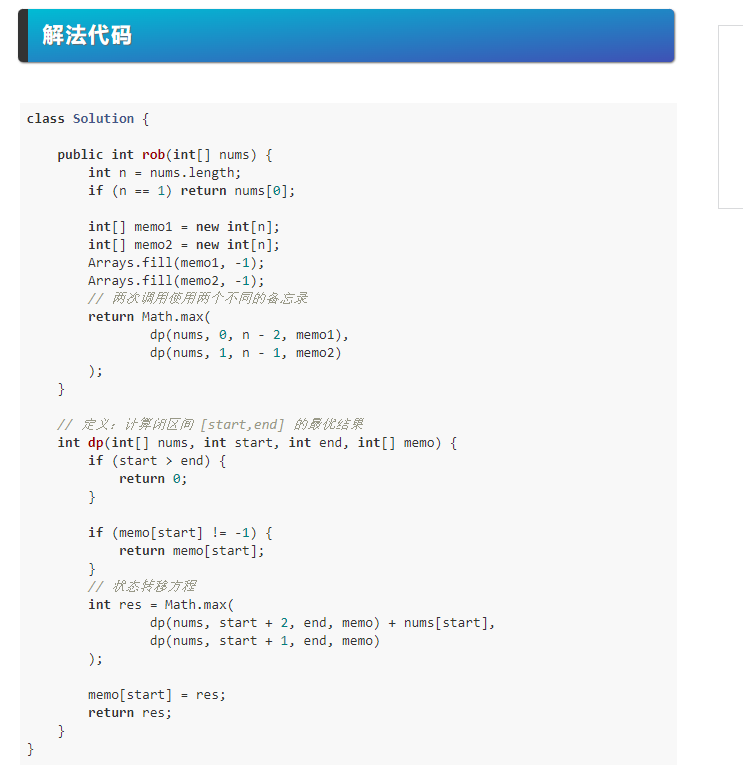






## 打家劫舍2

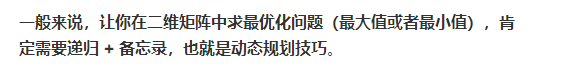




## 打家劫舍3

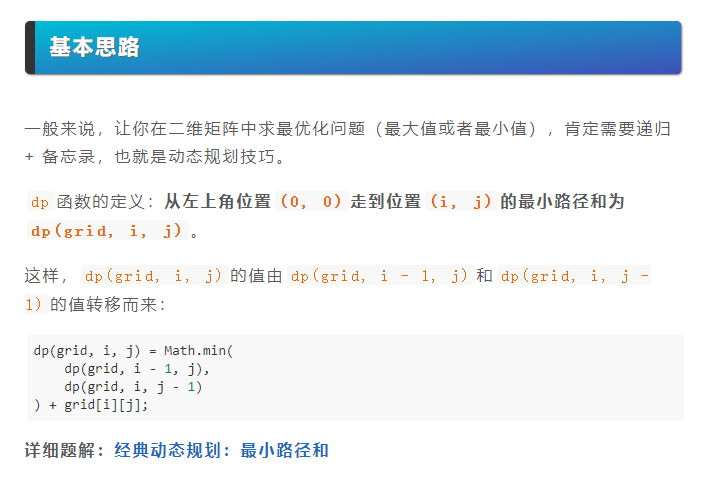


## -----二维



## 最小路径和





var minPathSum = function(grid) {

    const m = grid.length;

    const n = grid[0].length;

    const memo = new Array(m).fill().map(() => new Array(n).fill(-1));

    const dp =(grid,m,n)=>{

        if (m == 0 && n == 0) {

            return grid[0][0];

        }

        if (m < 0 || n < 0) {

            return Number.MAX\_VALUE;

        }

        // 避免重复计算

        if (memo[m][n] != -1) {

            return memo[m][n];

        }

        // 将计算结果记入备忘录

        memo[m][n] = Math.min(

                dp(grid, m - 1, n),

                dp(grid, m, n - 1)

        ) + grid[m][n];

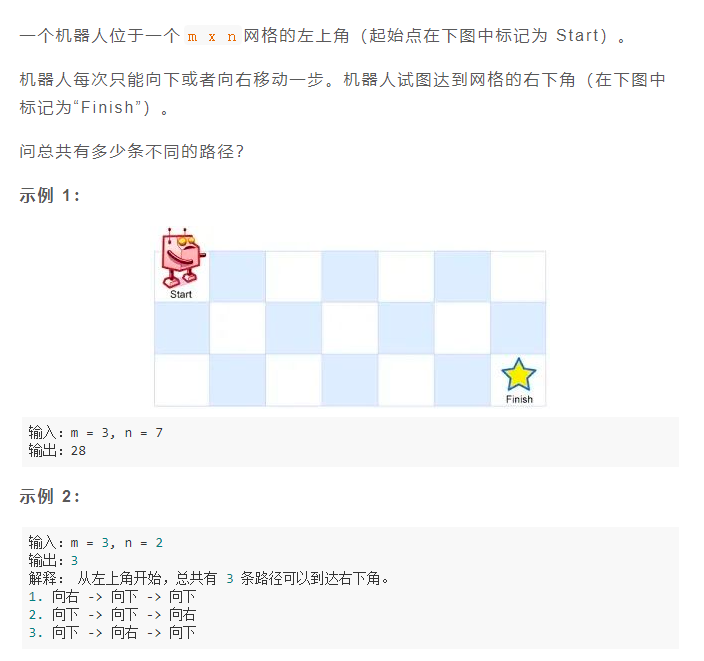
        return memo[m][n];

    }

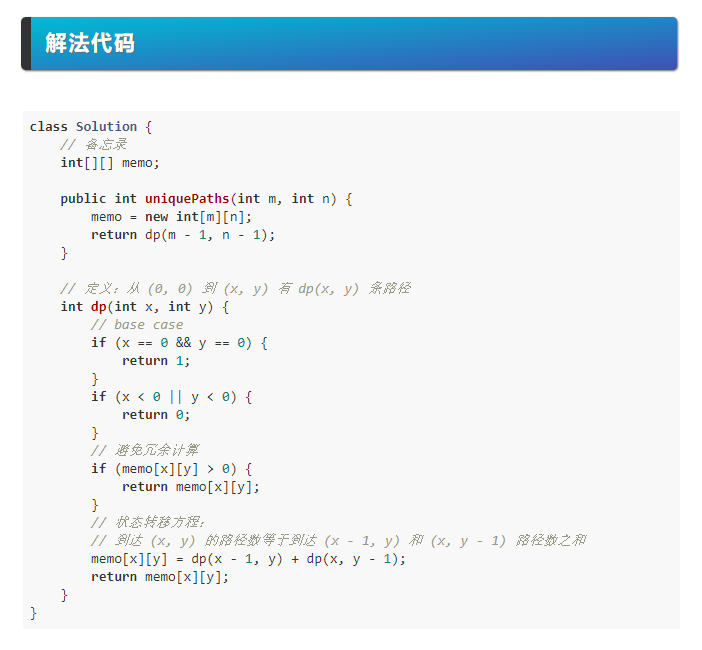
    return dp(grid, m - 1, n - 1);

};

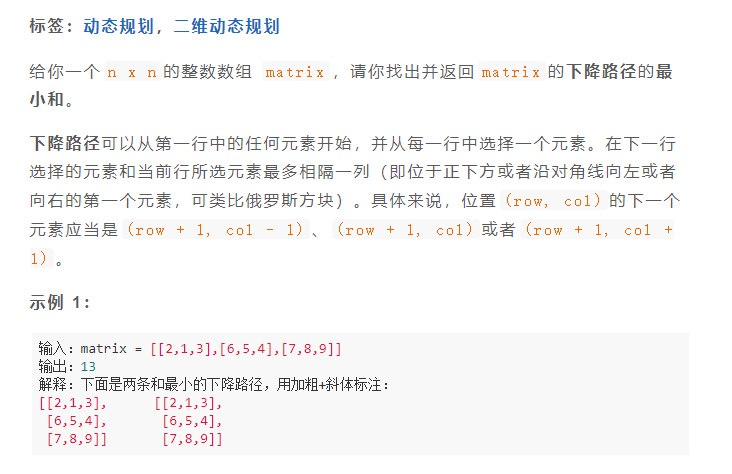
## 不同路径

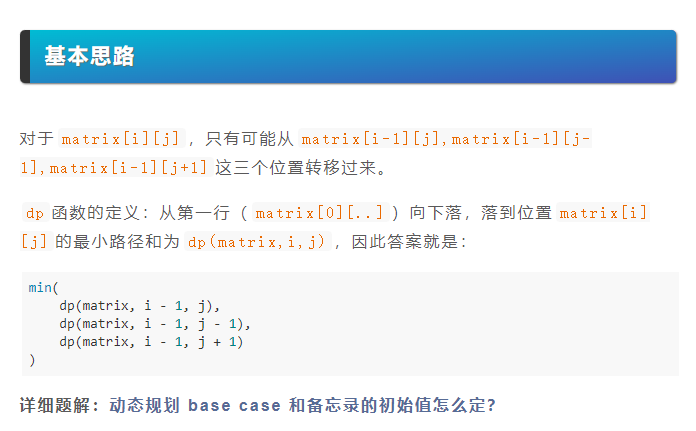


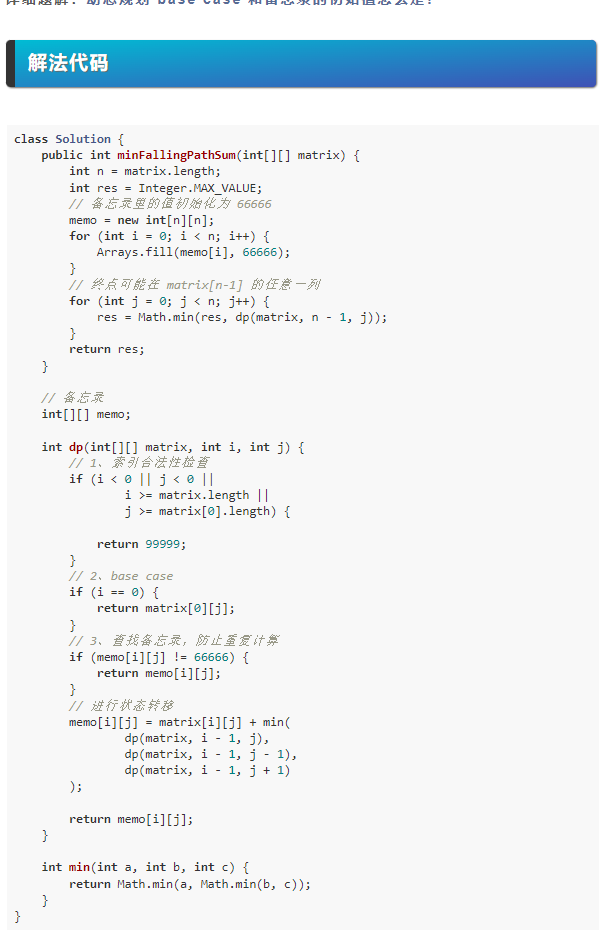




## 下降路径的最小和







## 股票买卖问题

