剑指 Offer 03. 数组中重复的数字

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

示例 1：

输入：
[2, 3, 1, 0, 2, 5, 3]
输出：2 或 3

```python
class Solution:
    def findRepeatNumber(self, nums: List[int]) -> int:
        dic = {i:0 for i in nums}
        for i in nums:
            dic[i] += 1
            if dic[i]>1:return i
```

剑指 Offer 04. 二维数组中的查找

在一个 n * m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

示例：

现有矩阵 matrix 如下：

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

给定 target = 5，返回 true。

给定 target = 20，返回 false。

限制：

0 <= n <= 1000

```
0 <= m <= 1000
```

```python
class Solution:
    def findNumberIn2DArray(self, matrix: List[List[int]], target: int) -> bool:
        self.tag = False
        for listM in matrix:
            self.find(listM,target)
        return self.tag
    def find(self,nums,target):
        left,right = 0,len(nums)-1
        while(left<=right):
            mid = left+ (right-left)//2
            if(nums[mid]==target):
                self.tag = True
                return
            if(nums[mid]>target):
                right = mid -1
            if(nums[mid]<target):
                left = mid + 1
```

剑指 Offer 05. 替换空格

请实现一个函数，把字符串 s 中的每个空格替换成"%20"。

示例 1：

输入：s = "We are happy."
输出："We%20are%20happy."

限制：

0 <= s 的长度 <= 10000

```python
class Solution:
    def replaceSpace(self, s: str) -> str:
        newS = ""
        for i in range(len(s)):
            if s[i]==" ":newS = newS+"%20"
            else:   newS = newS+s[i]
        return newS
```

剑指 Offer 06. 从尾到头打印链表

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。

示例 1：

输入：head = [1,3,2]
输出：[2,3,1]

限制：

0 <= 链表长度 <= 10000

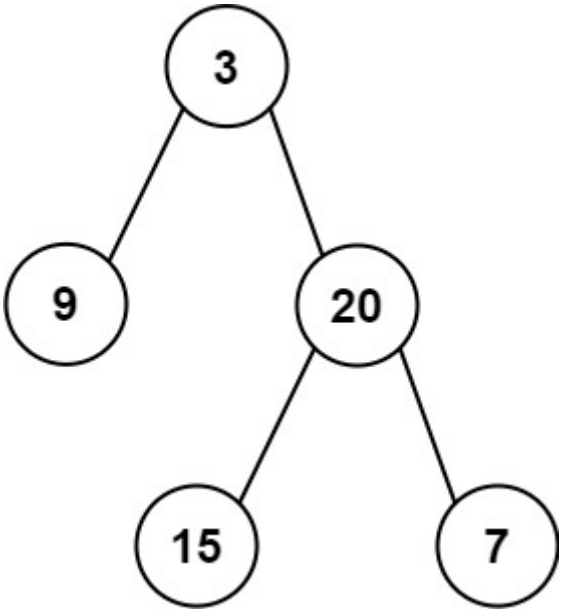
```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reversePrint(self, head: ListNode) -> List[int]:
        tmp = []
        res = []
        while(head!=None):
            tmp.append(head.val)
            head = head.next
        for i in range(len(tmp)):
            x= tmp.pop()
            res.append(x)
        return res
```

剑指 Offer 07. 重建二叉树

输入某二叉树的前序遍历和中序遍历的结果，请构建该二叉树并返回其根节点。

假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

示例 1:




```
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
示例 2:

Input: preorder = [-1], inorder = [-1]
Output: [-1]


限制:

0 <= 节点个数 <= 5000
```

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if not preorder:
            return

        root = TreeNode(preorder[0])
        i = inorder.index(preorder[0])
        root.left = self.buildTree(preorder[1:i+1], inorder[:i])
        root.right = self.buildTree(preorder[i+1:], inorder[i+1:])
        return root
```

注意:

```
not preorder 与 preorder == None 的区别
```

剑指 Offer 09. 用两个栈实现队列

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1)

示例 1：

输入：
["CQueue","appendTail","deleteHead","deleteHead"]
[[],[3],[],[]]
输出：[null,null,3,-1]
示例 2：

输入：
["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]
[[],[],[5],[2],[],[]]
输出：[null,-1,null,null,5,2]
提示：

1 <= values <= 10000
最多会对 appendTail、deleteHead 进行 10000 次调用

```python
class CQueue:

    def __init__(self):
        self.stack1 = []
        self.stack2 = []
    def appendTail(self, value: int) -> None:
        self.stack1.append(value)


    def deleteHead(self) -> int:
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        if not self.stack2:
            return -1
        else:
            return self.stack2.pop()
```

注：

当self.stack2为空时，才能往里面添加self.stack1的元素

剑指 Offer 10- I. 斐波那契数列

写一个函数，输入 n ，求斐波那契（Fibonacci）数列的第 n 项（即 F(N)）。斐波那契数列的定义如下：

F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入：n = 2
输出：1
示例 2：

输入：n = 5
输出：5

提示：

0 <= n <= 100

```python
class Solution:
    def fib(self, n: int) -> int:
        if (n==0 or n==1): return n
        dp = [0] * (n+1)
        dp[0],dp[1]=0,1
        for i in range(2,n+1):
            dp[i] = dp[i-1]+dp[i-2]
        return dp[n] % 1000000007
```

剑指 Offer 10- II. 青蛙跳台阶问题

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入：n = 2
输出：2
示例 2：

输入：n = 7
输出：21
示例 3：

输入：n = 0
输出：1
提示：

0 <= n <= 100

```python
class Solution:
    def numWays(self, n: int) -> int:
        if n == 0 or n==1:return 1

        dp = [1] * (n+1)

        for i in range(2,n+1):
            dp[i] = dp[i-1] +dp[i-2]
        return dp[n] % 1000000007
```

剑指 Offer 11. 旋转数组的最小数字

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。

给你一个可能存在 重复 元素值的数组 numbers ，它原来是一个升序排列的数组，并按上述情形进行了一次旋转。请返回旋转数组的最小元素。例如，数组 [3,4,5,1,2] 为 [1,2,3,4,5] 的一次旋转，该数组的最小值为 1。

注意，数组 [a[0], a[1], a[2], ..., a[n-1]] 旋转一次 的结果为数组 [a[n-1], a[0], a[1], a[2], ..., a[n-2]] 。

示例 1：

输入：numbers = [3,4,5,1,2]
输出：1
示例 2：

输入：numbers = [2,2,2,0,1]
输出：0

提示：

n == numbers.length
1 <= n <= 5000
-5000 <= numbers[i] <= 5000
numbers 原来是一个升序排序的数组，并进行了 1 至 n 次旋转

```python
class Solution:
    def minArray(self, numbers: List[int]) -> int:
        left , right = 0, len(numbers)-1
        while(left<right):
            mid = left + (right-left)//2
            if(numbers[mid]< numbers[right]):
                right = mid
            elif(numbers[mid]>numbers[right]):
                left = mid +1
            else:
                right = right-1
        return numbers[left]
```

注:

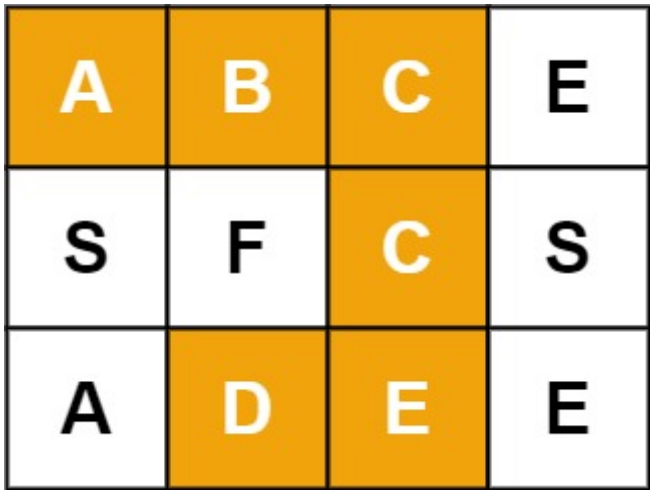
```
(numbers[mid]== numbers[right])时    right = right -1
```

剑指 Offer 12. 矩阵中的路径

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

例如，在下面的 3×4 的矩阵中包含单词 "ABCCED"（单词中的字母已标出）。



示例 1：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
输出：true
示例 2：

输入：board = [["a","b"],["c","d"]], word = "abcd"
输出：false

提示：

m == board.length
n = board[i].length
1 <= m, n <= 6
1 <= word.length <= 15
board 和 word 仅由大小写英文字母组成

```python
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:

        for i in range(len(board)):
            for j in range(len(board[0])):
                if self.backtrak(board,word,i,j,0):
                    return True
        return False

    def backtrak(self,board,word,i,j,k):
        if i<0 or i>=len(board) or j<0 or j>=len(board[0]) or board[i][j] !=
word[k]:
            return False
        if k == len(word)-1:
            return True
        board[i][j] = ''
        res = self.backtrak(board,word,i+1,j,k+1) or self.backtrak(board,word,i-
1,j,k+1) or self.backtrak(board,word,i,j+1,k+1) or self.backtrak(board,word,i,j-
1,k+1)

        board[i][j] = word[k]
        return res
```

注:

```
返回True的条件 k == len(word)-1
以及: board[i][j] = ''   board[i][j] = word[k]
```

剑指 Offer 14- I. 剪绳子

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]*k[1]*...*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

示例 1：

输入: 2
输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1
示例 2:

输入: 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36
提示：

2 <= n <= 58

```python
class Solution:
    def cuttingRope(self, n: int) -> int:
        dp = [-1] * (n+1)
        if(n==0 or n ==1 or n==2):
            return 1
        dp[1]=1
        dp[2]=1
        for i in range(3,n+1):
            for j in range(1,i-1):
                dp[i] = max(dp[i],max(j*(i-j),j*dp[i-j]))
        return dp[n]
```

确定状态转移方程
当 i ≥ 2 时，假设对长度为 i 绳子剪出的第一段绳子长度是 j（1≤j<i），则有以下两种方案：
将 i 剪成 j 和 i-j 长度的绳子，且 i-j 不再继续剪，此时的乘积是 j×(i−j) ；
将 i 剪成 j 和 i-j 长度的绳子，且 i-j 继续剪成多段长度的绳子，此时的乘积是 j×dp[i−j] 。
因此，当 j 固定时，有 dp[i]=max(j×(i−j),j×dp[i−j])。由于 j 的取值范围是 1 到 i ，需要遍历所有的 j 得到dp[i]的

剑指 Offer 14- II. 剪绳子 II

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m - 1] 。请问 k[0]*k[1]*...*k[m - 1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入: 2
输出: 1
解释: 2 = 1 + 1, 1 × 1 = 1
示例 2:

输入: 10
输出: 36
解释: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36

```python
class Solution:
    def cuttingRope(self, n: int) -> int:
        dp = [-1] * (n+1)
        if(n==0 or n ==1 or n==2):
            return 1
        dp[1]=1
        dp[2]=1
        for i in range(3,n+1):
            for j in range(1,i-1):
                dp[i] = max(dp[i],max(j*(i-j),j*dp[i-j]))
        return dp[n] %1000000007
```

剑指 Offer 15. 二进制中1的个数

编写一个函数，输入是一个无符号整数（以二进制串的形式），返回其二进制表达式中数字位数为 '1' 的个数（也被称为 汉明重量).）。

提示：

请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
在 Java 中，编译器使用 二进制补码 记法来表示有符号整数。因此，在上面的 示例 3 中，输入表示有符号整数 -3。

示例 1：

输入：n = 11 (控制台输入 00000000000000000000000000001011)
输出：3
解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
示例 2：

输入：n = 128 (控制台输入 00000000000000000000000010000000)
输出：1
解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
示例 3：

输入：n = 4294967293 (控制台输入 11111111111111111111111111111101，部分语言中 n = -3）
输出：31
解释：输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。

提示：

输入必须是长度为 32 的 二进制串 。

```
class Solution:
    def hammingWeight(self, n: int) -> int:
        res = 0
        while(n!=0):
            n= n&(n-1)
            res+=1
        return res
```

注意：

```
a = 10  # 10的二进制为1010
b = 18  # 18的二进制为10010
"""
按位与运算符:
```

```
& : 如果两个相应位都为1,则该位的结果为1,否则为0
01010
10010
计算的结果为:
00010
把二进制00010转10进制结果为2,所以下面的打印结果应该为2
"""
print(a & b)  # 打印结果为2

"""
按位或运算符:
| : 只要对应的二个二进位有一个为1时，结果位就为1
01010
10010
计算的结果为:
11010
把二进制11010转10进制结果为26,所以下面的打印结果应该为26
"""
print(a | b)  # 打印结果为26
"""
按位异或运算符:
^ : 当两对应的二进位相异时，结果为1
01010
10010
计算的结果为:
11000
把二进制11000转10进制结果为24,所以下面的打印结果应该为24
"""
print(a ^ b)  # 打印结果为24
"""
按位取反运算符:
~ : 对数据的每个二进制位取反,即把1变为0,把0变为1
01010
计算的结果为:
10101
把二进制10101转10进制结果为-11,所以下面的打印结果应该为-11
"""
print(~a)  # 打印结果为-11
```

剑指 Offer 16. 数值的整数次方

实现 pow(x, n) ，即计算 x 的 n 次幂函数（即，xn）。不得使用库函数，同时不需要考虑大数问题。

示例 1：

输入：x = 2.00000, n = 10
输出：1024.00000
示例 2：

输入：x = 2.10000, n = 3
输出：9.26100
示例 3：

输入：x = 2.00000, n = -2
输出：0.25000
解释：2-2 = 1/22 = 1/4 = 0.25

提示：

-100.0 < x < 100.0
-231 <= n <= 231-1
-104 <= xn <= 104

```python
class Solution:
    def myPow(self, x: float, n: int) -> float:
        res =1
        if n<0:
            x=1/x
            n = -n

        while n:
            if n & 1: res *= x
            x *= x
            n >>= 1
        return res
```

注意：

n右移，快幂思想，当n & 1 等于一，也就是为True时，才执行 res *= x

剑指 Offer 17. 打印从1到最大的n位数

输入数字 n，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999。

示例 1:

输入: n = 1
输出: [1,2,3,4,5,6,7,8,9]

说明：

用返回一个整数列表来代替打印
n 为正整数

```python
class Solution:
    def printNumbers(self, n: int) -> List[int]:
        total = 1
        while n>0:
            total *= 10
            n -=1
        return [i for i in range(1,total)]
```

注意：

```
n每减少1，total就要*10
```

剑指 Offer 18. 删除链表的节点

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

注意：此题对比原题有改动

示例 1:

输入: head = [4,5,1,9], val = 5
输出: [4,1,9]
解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.
示例 2:

输入: head = [4,5,1,9], val = 1
输出: [4,5,9]
解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

说明:

题目保证链表中节点的值互不相同
若使用 C 或 C++ 语言，你不需要 free 或 delete 被删除的节点

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def deleteNode(self, head: ListNode, val: int) -> ListNode:
        dummy = ListNode(-1)
        dummy.next = head
        p = dummy
        while(p!=None and p.next!=None):
            if p.next.val==val:
                p.next = p.next.next
                return dummy.next
            p = p.next
```

(困难)剑指 Offer 19. 正则表达式匹配

请实现一个函数用来匹配包含'. '和'*'的正则表达式。模式中的字符'.'表示任意一个字符，而'*'表示它前面的字符可以出现任意次（含0次）。在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但与"aa.a"和"ab*a"均不匹配。

示例 1:

输入:
s = "aa"
p = "a"
输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
示例 2:

输入:
s = "aa"
p = "a*"
输出: true
解释: 因为 '*' 代表可以匹配零个或多个前面的那一个元素, 在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。
示例 3:

输入:
s = "ab"
p = ".*"
输出: true
解释: ".*" 表示可匹配零个或多个（'*'）任意字符（'.'）。
示例 4:

输入:
s = "aab"
p = "c*a*b"
输出: true
解释: 因为 '*' 表示零个或多个，这里 'c' 为 0 个, 'a' 被重复一次。因此可以匹配字符串 "aab"。
示例 5:

输入:
s = "mississippi"
p = "mis*is*p*."
输出: false
s 可能为空，且只包含从 a-z 的小写字母。
p 可能为空，且只包含从 a-z 的小写字母以及字符 . 和 *，无连续的 '*'。

```python
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        m, n = len(s), len(p)
        dp = [[False] * (n+1) for _ in range(m+1)]

        # 初始化
        dp[0][0] = True
        for j in range(1, n+1):
            if p[j-1] == '*':
                dp[0][j] = dp[0][j-2]
```

```
        # 状态更新
        for i in range(1, m+1):
            for j in range(1, n+1):
                if s[i-1] == p[j-1] or p[j-1] == '.':
                    dp[i][j] = dp[i-1][j-1]
                elif p[j-1] == '*':     # 【题目保证'*'号不会是第一个字符，所以此处有j>=2】
                    if s[i-1] != p[j-2] and p[j-2] != '.':
                        dp[i][j] = dp[i][j-2]
                    else:
                        dp[i][j] = dp[i][j-2] | dp[i-1][j]

        return dp[m][n]
```

剑指 Offer 20. 表示数值的字符串

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。

数值（按顺序）可以分成以下几个部分：

若干空格
一个 小数 或者 整数
（可选）一个 'e' 或 'E' ，后面跟着一个 整数
若干空格
小数（按顺序）可以分成以下几个部分：

（可选）一个符号字符（'+' 或 '-'）
下述格式之一：
至少一位数字，后面跟着一个点 '.'
至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字
一个点 '.' ，后面跟着至少一位数字
整数（按顺序）可以分成以下几个部分：

（可选）一个符号字符（'+' 或 '-'）
至少一位数字
部分数值列举如下：

["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]
部分非数值列举如下：

["12e", "1a3.14", "1.2.3", "+-5", "12e+5.4"]

示例 1：

输入：s = "0"
输出：true
示例 2：

输入：s = "e"
输出：false
示例 3：

输入：s = "."
输出：false
示例 4：

输入：s = "   .1 "
输出：true

提示：

1 <= s.length <= 20
s 仅含英文字母（大写和小写），数字（0-9），加号 '+' ，减号 '-' ，空格 ' ' 或者点 '.' 。

```
class Solution:
    def isNumber(self, s: str) -> bool:
        s = s.strip()
        i = 0
        if not s: return False
        isDoe = False    # .是否出现过
        isE = False      #E是否出现过
        isNum = False    #数字是否出现过

        while i < len(s):
            #数字的出现没有限制，只需要将isNum改为True就行
            if '0'<=s[i]<='9':
                i+=1
                isNum = True

            #小数点的限制为：1. 不可以同时出现两个小数点，则isDoe为True时return False
            #                2.不能出现在E或e的后面
            #                2.前面或者后面必须要有数字
            elif s[i] == '.':
                if isDoe or isE : return False
                if  (i-1>=0 and '0'<=s[i-1]<='9') or (i+1<len(s) and '0'<=s[i+1]
<='9'):
                    i += 1
                    isDoe = True
                else:
                    return False

            # E合理的位置必须大于0，  且不能同时出现两个E，则isE为True时return False，
            # 同时E前面必须出现过数字，isNum为False时return False
            elif i>0 and s[i] in 'Ee':
                if isE or not isNum: return False
                if i == len(s) - 1:
                    return False
                i += 1
                isE = True

            # + - 出现的位置必须在第0位或者E(e)的后面，且不能出现在最后一位
            elif s[i] =='+' or s[i] == '-':
                if i != len(s)-1 and (i == 0 or s[i-1] in 'eE'):
                    i += 1
                else:
                    return False
```

```
            else:
                return False

        return True
```

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数在数组的前半部分，所有偶数在数组的后半部分。

示例：

输入：nums = [1,2,3,4]
输出：[1,3,2,4]
注：[3,1,2,4] 也是正确的答案之一。

提示：

0 <= nums.length <= 50000
0 <= nums[i] <= 10000

```
class Solution:
    def exchange(self, nums: List[int]) -> List[int]:
        slow , fast = 0,0
        while(fast<len(nums)):
            if(nums[fast]%2==1):
                nums[slow],nums[fast]=nums[fast],nums[slow]
                slow +=1
            fast +=1
        return nums
```

剑指 Offer 22. 链表中倒数第k个节点

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

示例：

给定一个链表: 1->2->3->4->5, 和 k = 2.

返回链表 4->5.

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getKthFromEnd(self, head: ListNode, k: int) -> ListNode:
        dumpy = ListNode(-1)
        dumpy.next = head
        return self.resNode(dumpy,k)
    def resNode(self,head,k):
        slowHead= fastHead=head
        for i in range(k):
            fastHead=fastHead.next
        while(fastHead!=None):
            fastHead= fastHead.next
            slowHead = slowHead.next
        return slowHead
```

剑指 Offer 24. 反转链表

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例:

输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL

限制：

0 <= 节点个数 <= 5000

```python
# Definition for singly-linked list.
```

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre = None
        while(head):
            tmp = head.next
            head.next = pre
            pre = head
            head = tmp
        return pre
```

剑指 Offer 25. 合并两个排序的链表

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

示例1：

输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
限制：

0 <= 链表长度 <= 1000

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        dumpy = ListNode(-1)
        p = dumpy
        while(l1 and l2):
            if l1.val>l2.val:
                p.next = l2
                l2 = l2.next
            else:
                p.next = l1
                l1 = l1.next
            p = p.next
        if l1: p.next=l1
        if l2: p.next=l2

        return dumpy.next
```

(理解)剑指 Offer 26. 树的子结构

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

例如:
给定的树 A:

给定的树 B:

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

示例 1:

输入：A = [1,2,3], B = [3,1]
输出：false
示例 2:

输入：A = [3,4,5,1,2], B = [4,1]
输出：true
限制:

0 <= 节点个数 <= 10000

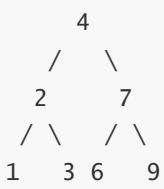
```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSubStructure(self, A: TreeNode, B: TreeNode) -> bool:
        if (A == None or B == None):
            return False
        if (A.val == B.val and self.compareTree(A, B)):
            return True
        return (self.isSubStructure(A.left, B) or self.isSubStructure(A.right,
B))

    def compareTree(self, A: TreeNode, B: TreeNode):
        if (B == None):
            return True
        if (B != None and A == None):
            return False
        if (A.val != B.val):
            return False
```

```
        return self.compareTree(A.left, B.left) and self.compareTree(A.right,
B.right)
```

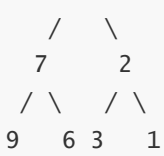
剑指 Offer 27. 二叉树的镜像

请完成一个函数，输入一个二叉树，该函数输出它的镜像。

例如输入：



镜像输出：



示例 1：

输入：root = [4,2,7,1,3,6,9]
输出：[4,7,2,9,6,3,1]

限制：

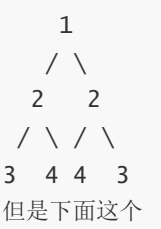
0 <= 节点个数 <= 1000

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def mirrorTree(self, root: TreeNode) -> TreeNode:
        if (root==None):
            return
        if(root.left== None and root.right==None):
            return root
        root.left,root.right = root.right,root.left
        self.mirrorTree(root.left)
        self.mirrorTree(root.right)
        return root
```

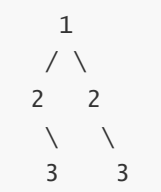
剑指 Offer 28. 对称的二叉树

请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的。

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:



示例 1：

输入：root = [1,2,2,3,4,4,3]
输出：true
示例 2：

输入：root = [1,2,2,null,3,null,3]
输出：false

限制：

0 <= 节点个数 <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if(root == None):
            return True
        return self.check(root.left,root.right)

    def check(self,left,right):
        if(left==None or right == None):
            return left == right
        if(left.val!=right.val):
            return False
```

```
        return self.check(left.left,right.right) and
self.check(left.right,right.left)
```

剑指 Offer 29. 顺时针打印矩阵

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

示例 1：

输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[1,2,3,6,9,8,7,4,5]
示例 2：

输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
输出：[1,2,3,4,8,12,11,10,9,5,6,7]

限制：

0 <= matrix.length <= 100
0 <= matrix[i].length <= 100

```python
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if(len(matrix)==0):return []
        m = len(matrix)
        n = len(matrix[0])
        upBound,downBound = 0,m-1
        leftBound,rightBound = 0,n-1
        res = []
        while(len(res)<m*n):
            # 在顶部从左到右
            if(upBound<=downBound):
                for i in range(leftBound,rightBound+1):
                    res.append(matrix[upBound][i])
                upBound +=1
            # 在右侧从上到下
            if(leftBound<=rightBound):
                for i in range(upBound,downBound+1):
                    res.append(matrix[i][rightBound])
                rightBound -=1
            #在底部从右向左
            if(upBound<=downBound):
                for i in range(rightBound,leftBound-1,-1):
                    res.append(matrix[downBound][i])
                downBound -=1
            #在左侧从下向上
            if(leftBound<=rightBound):
                for i in range(downBound,upBound-1,-1):
                    res.append(matrix[i][leftBound])
```

```
                leftBound +=1
        return res
```

剑指 Offer 30. 包含min函数的栈

定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

示例:

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.min();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.min();   --> 返回 -2.
```

提示：

各函数的调用总次数不超过 20000 次

```
class MinStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stk,self.minStack= [] , []


    def push(self, x: int) -> None:
        self.stk.append(x)
        if(not self.minStack or self.minStack[-1]>=x ):
            self.minStack.append(x)


    def pop(self) -> None:
        if (self.stk.pop()==self.minStack[-1]):
            self.minStack.pop()


    def top(self) -> int:
        return self.stk[-1]


    def min(self) -> int:
        return self.minStack[-1]
```

剑指 Offer 31. 栈的压入、弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。

示例 1：

```
输入：pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
输出：true
解释：我们可以按以下顺序执行：
push(1), push(2), push(3), push(4), pop() -> 4,
push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

示例 2：

```
输入：pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
输出：false
解释：1 不能在 2 之前弹出。
```

提示：

```
0 <= pushed.length == popped.length <= 1000
0 <= pushed[i], popped[i] < 1000
pushed 是 popped 的排列。
```

```python
class Solution:
    def validateStackSequences(self, pushed: List[int], popped: List[int]) ->
bool:
        stack = []
        for num in pushed:
            if num != popped[0]:
                stack.append(num)
            else:
                popped.pop(0)

            while stack and  stack[-1] == popped[0]:
                stack.pop(-1)
                popped.pop(0)

        return not stack
```

剑指 Offer 32 - I. 从上到下打印二叉树

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。

例如：

```
给定二叉树: [3,9,20,null,null,15,7],
```

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

返回：

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

提示：

节点总数 <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[int]:
        # 层序遍历，BFS
        if not root:
            return []
        res = []
        queue = [root]
        while queue:
            node = queue.pop(0)
            res.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        return res
```

剑指 Offer 32 - II. 从上到下打印二叉树 II

从上到下按层打印二叉树，同一层的节点按从左到右的顺序打印，每一层打印到一行。

例如：
给定二叉树: [3,9,20,null,null,15,7],

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

返回其层次遍历结果：

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

提示：

节点总数 <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        # 层序遍历，BFS
        if not root:
            return []
        res = []
        tmp = [root]
        while tmp:
            floor = []
            swap = []
            for t in tmp:
                floor.append(t.val)
                if t.left:
                    swap.append(t.left)
                if t.right:
                    swap.append(t.right)
            res.append(floor)
            tmp = swap
        return res
```

剑指 Offer 32 - III. 从上到下打印二叉树 III

请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

例如：
给定二叉树: [3,9,20,null,null,15,7],

```
    3
   / \
```

```
  9  20
    /  \
   15   7
```
返回其层次遍历结果：

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

提示：

节点总数 <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        tag = True
        res = []
        tmp = [root]
        while tmp:
            floor = []
            swap = []
            for t in tmp:
                floor.append(t.val)
                if t.left:
                    swap.append(t.left)
                if t.right:
                    swap.append(t.right)
            if tag:
                res.append(floor)
                tag = False
            else:
                floor.reverse()
                res.append(floor)
                tag = True
            tmp = swap
        return res
```

剑指 Offer 33. 二叉搜索树的后序遍历序列

剑指 Offer 33. 二叉搜索树的后序遍历序列

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历结果。如果是则返回 `true`，否则返回 `false`。假设输入的数组的任意两个数字都互不相同。

参考以下这颗二叉搜索树：

```
     5
    / \
   2   6
  / \
 1   3
```

示例 1：

```
输入: [1,6,3,2,5]
输出: false
```

示例 2：

```
输入: [1,3,2,6,5]
输出: true
```

提示：

数组长度 <= 1000

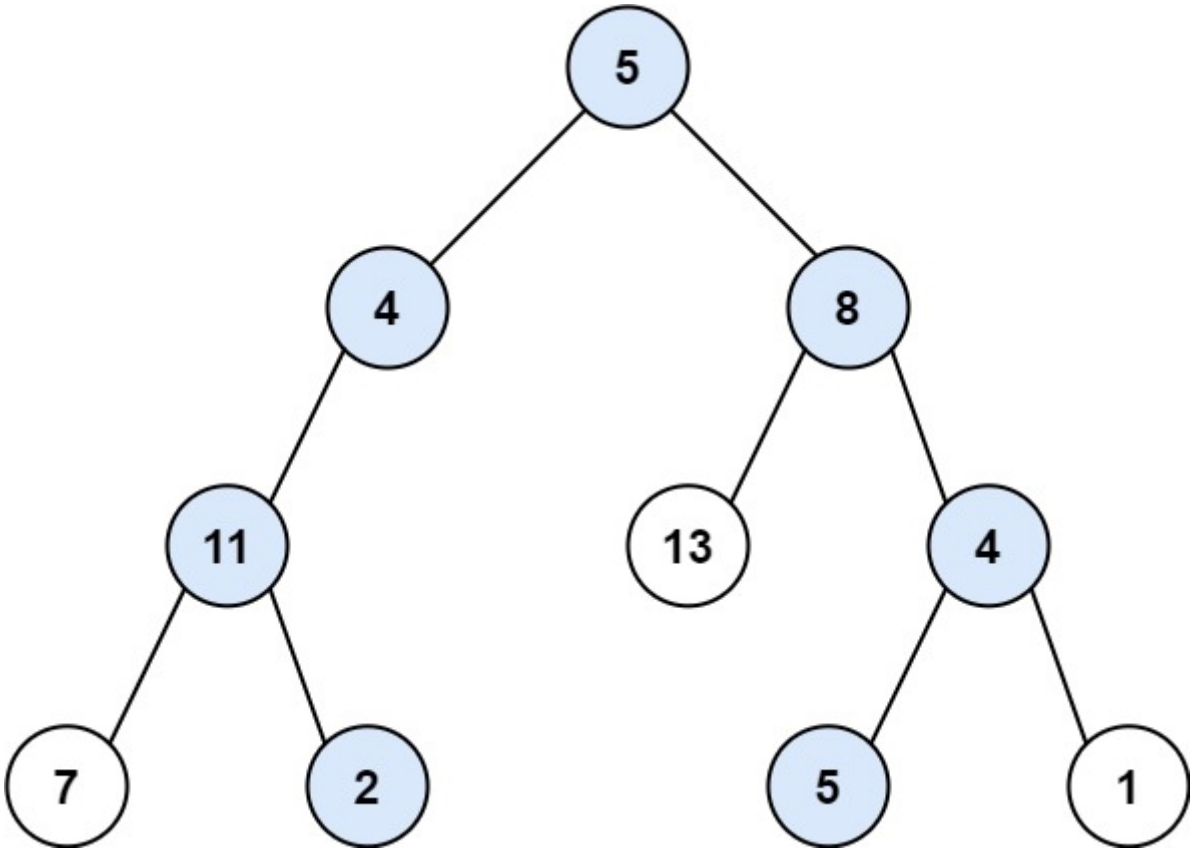
```python
class Solution:
    def verifyPostorder(self, postorder: List[int]) -> bool:
        stack = []
        orderList = sorted(postorder)

        for num in orderList:
            if num != postorder[0]:
                stack.append(num)
            else:
                postorder.pop(0)
            while(stack and stack[-1]== postorder[0]):
                stack.pop(-1)
                postorder.pop(0)
        return not stack
```

(理解)剑指 Offer 34. 二叉树中和为某一值的路径

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。

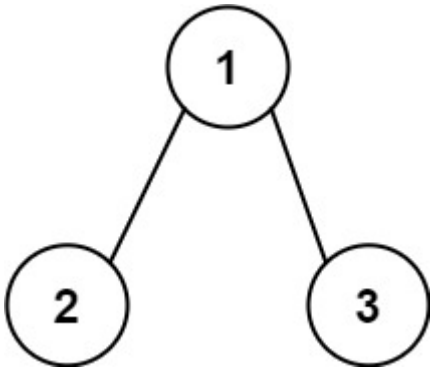
叶子节点 是指没有子节点的节点。

示例 1：




```
输入：root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
输出：[[5,4,11,2],[5,8,4,5]]
```

示例 2：



输入：root = [1,2,3], targetSum = 5
输出：[]
示例 3：

输入：root = [1,2], targetSum = 0
输出：[]

提示：

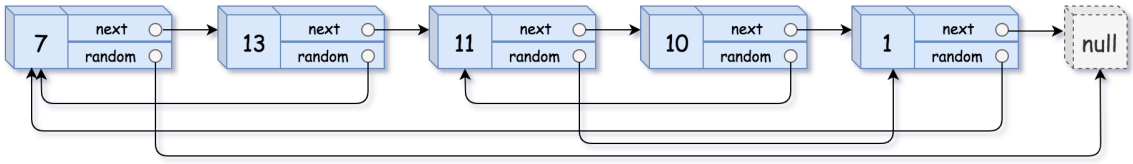
树中节点总数在范围 [0, 5000] 内
-1000 <= Node.val <= 1000
-1000 <= targetSum <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def pathSum(self, root: TreeNode, target: int) -> List[List[int]]:
        self.res = []
        self.tmp = []
        self.traverse(root,target,self.tmp)
        return self.res
    def traverse(self,root,target,path):
        if root == None: return

        if root.left == None and root.right== None:
            if sum(self.tmp)==target-root.val:
                self.tmp.append(root.val)
                self.res.append(self.tmp[:])
                self.tmp.pop()
            return


        self.tmp.append(root.val)
        self.traverse(root.left,target,self.tmp)
        self.tmp.pop()

        self.tmp.append(root.val)
        self.traverse(root.right,target,self.tmp)
        self.tmp.pop()
```

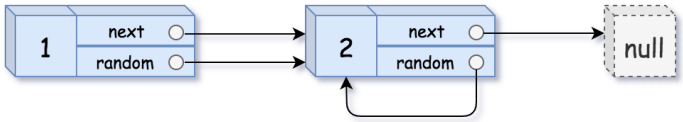
(理解)剑指 Offer 35. 复杂链表的复制

请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null。

示例 1：



输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]
示例 2：



输入：head = [[1,1],[2,1]]
输出：[[1,1],[2,1]]
示例 3：

输入：head = [[3,null],[3,0],[3,null]]
输出：[[3,null],[3,0],[3,null]]
示例 4：

输入：head = []
输出：[]
解释：给定的链表为空（空指针），因此返回 null。

提示：

-10000 <= Node.val <= 10000
Node.random 为空（null）或指向链表中的节点。
节点数目不超过 1000 。

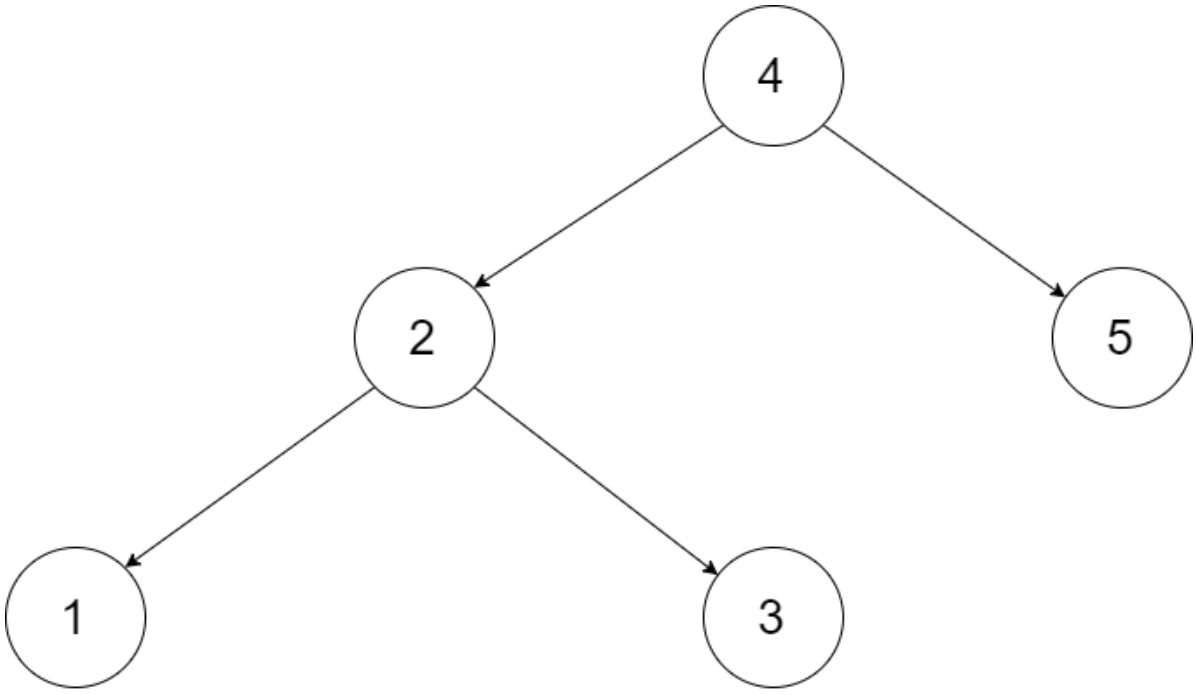
```
"""
# Definition for a Node.
class Node:
    def __init__(self, x: int, next: 'Node' = None, random: 'Node' = None):
        self.val = int(x)
        self.next = next
        self.random = random
"""
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        if head == None: return None

        cur = head
        while(cur):
            tmp =Node(cur.val)
```

```
            tmp.next = cur.next
            cur.next = tmp
            cur = tmp.next

        cur = head
        while(cur):
            if cur.random:
                cur.next.random = cur.random.next
            cur = cur.next.next
        cur = res = head.next
        pre =head
        while(cur.next):
            pre.next = pre.next.next
            cur.next = cur.next.next
            pre = pre.next
            cur = cur.next
        pre.next = None
        return res
```

(理解)剑指 Offer 36. 二叉搜索树与双向链表

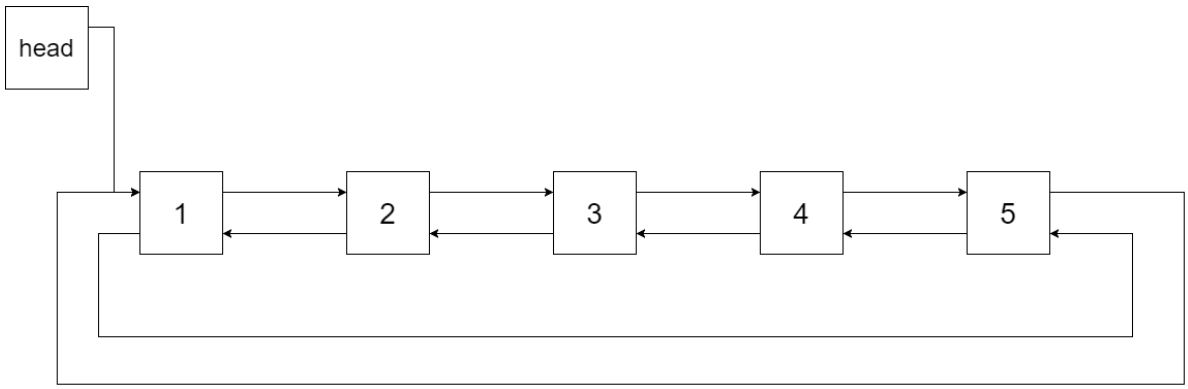
输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向。

为了让您更好地理解问题，以下面的二叉搜索树为例：



我们希望将这个二叉搜索树转化为双向循环链表。链表中的每个节点都有一个前驱和后继指针。对于双向循环链表，第一个节点的前驱是最后一个节点，最后一个节点的后继是第一个节点。

下图展示了上面的二叉搜索树转化成的链表。“head” 表示指向链表中有最小元素的节点。



特别地，我们希望可以就地完成转换操作。当转化完成以后，树中节点的左指针需要指向前驱，树中节点的右指针需要指向后继。还需要返回链表中的第一个节点的指针。

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
"""
class Solution:
    def treeToDoublyList(self, root: 'Node') -> 'Node':
        if not root: return

        self.pre = None
        self.traverse(root)
        self.pre.right,self.head.left = self.head ,self.pre

        return self.head
    def traverse(self,cur):

        if not cur: return

        self.traverse(cur.left)
        if self.pre:
            self.pre.right,cur.left = cur,self.pre
        else:
            self.head = cur
        self.pre = cur
        self.traverse(cur.right)
```

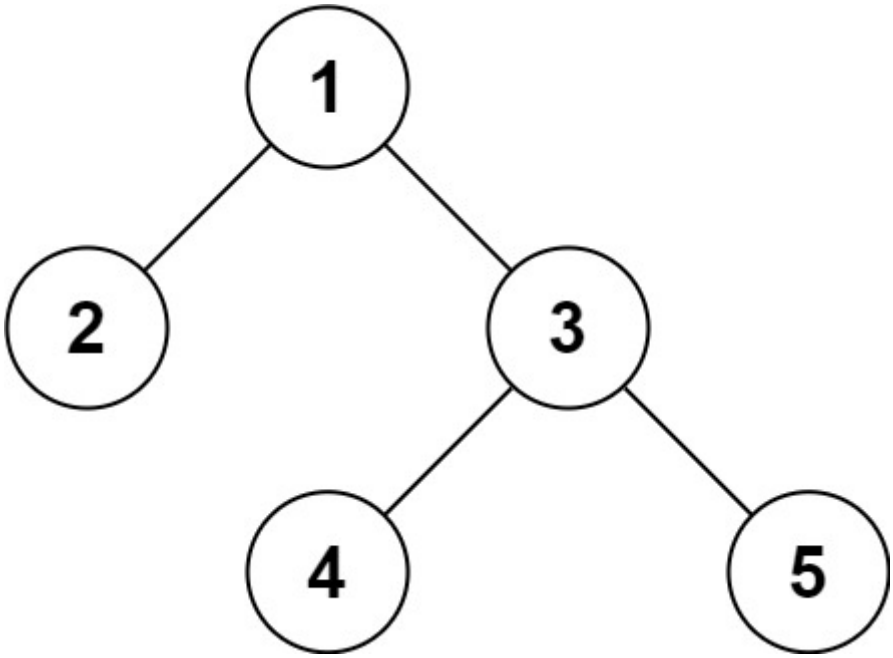
(困难)剑指 Offer 37. 序列化二叉树

请实现两个函数，分别用来序列化和反序列化二叉树。

你需要设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

提示：输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

示例：



输入：root = [1,2,3,null,null,4,5]
输出：[1,2,3,null,null,4,5]

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:

    def serialize(self, root):
        if not root: return "[]"
        queue = [root]
        res = []
        while queue:
            node = queue.pop(0)
            if node:
                res.append(str(node.val))
                queue.append(node.left)
```

```
                queue.append(node.right)
            else: res.append("null")
        return '[' + ','.join(res) + ']'

    def deserialize(self, data):
        if data == "[]": return
        vals, i = data[1:-1].split(','), 1
        root = TreeNode(int(vals[0]))
        queue =  [root]
        while queue:
            node = queue.pop(0)
            if vals[i] != "null":
                node.left = TreeNode(int(vals[i]))
                queue.append(node.left)
            i += 1
            if vals[i] != "null":
                node.right = TreeNode(int(vals[i]))
                queue.append(node.right)
            i += 1
        return root



# Your Codec object will be instantiated and called as such:
# codec = Codec()
# codec.deserialize(codec.serialize(root))
```

剑指 Offer 38. 字符串的排列

输入一个字符串，打印出该字符串中字符的所有排列。

你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。

示例:

输入：s = "abc"
输出：["abc","acb","bac","bca","cab","cba"]

限制：

1 <= s 的长度 <= 8

```
class Solution:
    def permutation(self, s: str) -> List[str]:
        self.res =[]
        track = ''
        used = [False] * (len(s))
        li = list(s)
```

```
        li.sort()
        newS = ''.join(li)

        self.backtrack(newS,track,used)

        return self.res

    def backtrack(self,s,track,used):
        if len(track) == len(s):
            self.res.append(track)
        for i in range(len(s)):
            if used[i]: continue
            if i>0 and s[i] == s[i-1] and not used[i-1]:continue
            track += s[i]
            used[i] = True
            self.backtrack(s,track,used)
            track = track[:-1]
            used[i] = False
```

剑指 Offer 39. 数组中出现次数超过一半的数字

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1:

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

限制：

1 <= 数组长度 <= 50000

```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        mid = len(nums)//2

        dicCount = {i:0 for i in nums}

        for x in nums: dicCount[x] +=1

        for k,v in dicCount.items():
            if v>mid: return k

        return 0
```

剑指 Offer 40. 最小的k个数

输入整数数组 arr ，找出其中最小的 k 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。

示例 1：

输入：arr = [3,2,1], k = 2
输出：[1,2] 或者 [2,1]
示例 2：

输入：arr = [0,1,2,1], k = 1
输出：[0]

限制：

0 <= k <= arr.length <= 10000
0 <= arr[i] <= 10000

```python
class Solution:
    def getLeastNumbers(self, arr: List[int], k: int) -> List[int]:
        arr.sort()
        return arr[:k]
```

剑指 Offer 41. 数据流中的中位数

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

例如，

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

void addNum(int num) - 从数据流中添加一个整数到数据结构中。
double findMedian() - 返回目前所有元素的中位数。
示例 1：

输入：
["MedianFinder","addNum","addNum","findMedian","addNum","findMedian"]
[[],[1],[2],[],[3],[]]
输出：[null,null,null,1.50000,null,2.00000]
示例 2：

输入：

```
["MedianFinder","addNum","findMedian","addNum","findMedian"]
[[],[2],[],[3],[]]
输出：[null,null,2.00000,null,2.50000]
```

```python
class MedianFinder:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.res = []


    def addNum(self, num: int) -> None:
        self.res.append(num)


    def findMedian(self) -> float:
        n =len(self.res)
        self.res.sort()
        if n%2==1:
            return self.res[n//2]
        else:
            return (self.res[n//2-1]+self.res[n//2])/2




# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder()
# obj.addNum(num)
# param_2 = obj.findMedian()
```

剑指 Offer 42. 连续子数组的最大和

输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。

要求时间复杂度为O(n)。

示例1:

```
输入: nums = [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
```

提示：

```
1 <= arr.length <= 10^5
-100 <= arr[i] <= 100
```

```python
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        dp = [0] * (len(nums))
        dp[0] =nums[0]
        maxRes = dp[0]
        for i in range(1,len(nums)):
            dp[i] = max(dp[i-1],0)+nums[i]
            maxRes = max(dp[i],maxRes)
        return maxRes
```

(困难) 剑指 Offer 43. 1 ~ n 整数中 1 出现的次数

输入一个整数 n ，求1～n这n个整数的十进制表示中1出现的次数。

例如，输入12，1～12这些整数中包含1 的数字有1、10、11和12，1一共出现了5次。

示例 1：

输入：n = 12
输出：5
示例 2：

输入：n = 13
输出：6

限制：

1 <= n < 2^31

```python
class Solution:
    def countDigitOne(self, n: int) -> int:
        digit, res = 1, 0
        high, cur, low = n // 10, n % 10, 0
        while high != 0 or cur != 0:
            if cur == 0: res += high * digit
            elif cur == 1: res += high * digit + low + 1
            else: res += (high + 1) * digit
            low += cur * digit
            cur = high % 10
            high //= 10
            digit *= 10
        return res
```

(理解) 剑指 Offer 44. 数字序列中某一位的数字

数字以0123456789101112131415…的格式序列化到一个字符序列中。在这个序列中，第5位（从下标0开始计数）是5，第13位是1，第19位是4，等等。

请写一个函数，求任意第n位对应的数字。

示例 1：

输入：n = 3
输出：3
示例 2：

输入：n = 11
输出：0

限制：

0 <= n < 2^31

```
class Solution:
    def findNthDigit(self, n: int) -> int:
        digit, start, count = 1, 1, 9
        while n > count: # 1.
            n -= count
            start *= 10
            digit += 1
            count = 9 * start * digit
        num = start + (n - 1) // digit # 2.
        return int(str(num)[(n - 1) % digit]) # 3.
```

剑指 Offer 45. 把数组排成最小的数

输入一个非负整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。

示例 1:

输入: [10,2]
输出: "102"
示例 2:

输入: [3,30,34,5,9]
输出: "3033459"

提示:

0 < nums.length <= 100
说明:

输出结果可能非常大，所以你需要返回一个字符串而不是整数
拼接起来的数字可能会有前导 0，最后结果不需要去掉前导 0

```
class Solution:
    def minNumber(self, nums: List[int]) -> str:
        strs = [str(num) for num in nums]
        strs.sort(key=functools.cmp_to_key(lambda x,y:int(x+y)-int(y+x)))
        return ''.join(strs)
```

剑指 Offer 46. 把数字翻译成字符串

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a” ，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

示例 1:

输入: 12258
输出: 5
解释: 12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"

提示：

0 <= num < 231

```
class Solution:
    def translateNum(self, num: int) -> int:
        s = str(num)
        dp = [1] * (len(s)+1)

        for i in range(2,len(s)+1):
            tmp = s[i-2:i]
            if tmp>='10' and tmp<= '25':
                dp[i] = dp[i-1] +dp[i-2]

            else:
                dp[i] = dp[i-1]
        return dp[len(s)]
```

剑指 Offer 47. 礼物的最大价值

在一个 m*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

示例 1:

输入:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 12
解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物

提示:

0 < grid.length <= 200
0 < grid[0].length <= 200

```python
class Solution:
    def maxValue(self, grid: List[List[int]]) -> int:
        m = len(grid)
        n = len(grid[0])

        dp = [[0] * n for _ in range(m)]

        dp [0][0] = grid[0][0]

        for i in range(1,m):
            dp[i][0] = dp[i-1][0] + grid[i][0]

        for i in range(1,n):
            dp[0][i] = dp[0][i-1] + grid[0][i]

        for i in range(1,m):
            for j in range(1,n):
                dp[i][j] = max(dp[i-1][j],dp[i][j-1]) + grid[i][j]

        return dp[m-1][n-1]
```

剑指 Offer 48. 最长不含重复字符的子字符串

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。

示例 1:

输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
示例 2:

输入: "bbbbb"
输出: 1

解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
示例 3:

输入: "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
      请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        window = {i:0 for i in s}
        res = slow = fast =0
        n = len(s)
        while(fast<n):
            c = s[fast]
            fast +=1
            window[c] +=1
            while(window[c]>1):
                d = s[slow]
                slow +=1
                window[d] -=1
            res = max(res,fast-slow)
        return res
```

剑指 Offer 49. 丑数

我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数。

示例:

输入: n = 10
输出: 12
解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数。
说明:

1 是丑数。
n 不超过1690。

```python
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        dp  = [1] * n
        a=b=c=0
        for i in range(1,n):
            dp[i] = min(dp[a]*2,dp[b]*3,dp[c]*5)

            if dp[i] == dp[a]*2: a+=1
            if dp[i] == dp[b]*3: b+=1
            if dp[i] == dp[c]*5: c+=1
        return dp[n-1]
```

剑指 Offer 50. 第一个只出现一次的字符

在字符串 s 中找出第一个只出现一次的字符。如果没有，返回一个单空格。 s 只包含小写字母。

示例 1:

输入：s = "abaccdeff"
输出：'b'
示例 2:

输入：s = ""
输出：' '

限制：

0 <= s 的长度 <= 50000

```python
class Solution:
    def firstUniqChar(self, s: str) -> str:
        if s == "":return " "
        count = {i:0 for i in s}
        for x in s:
            count[x] +=1

        for key,value in count.items():
            if value==1:
                return key
        return " "
```

剑指 Offer 51. 数组中的逆序对

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

示例 1:

输入: [7,5,6,4]

输出: 5

限制:

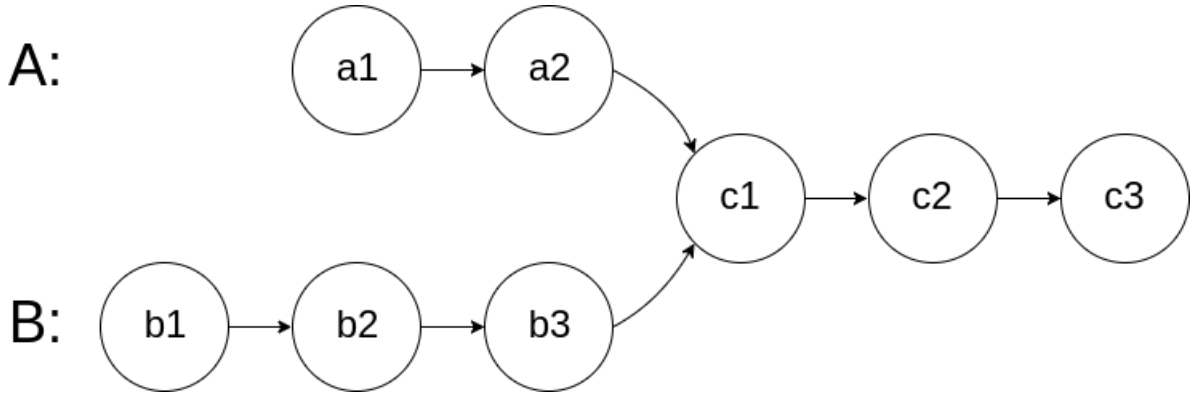
0 <= 数组长度 <= 50000

```python
import bisect
class Solution:
    def reversePairs(self, nums: List[int]) -> int:
        res, tb = 0, []
        for n in reversed(nums) :
            idx = bisect.bisect_left(tb, n)
            res += idx
            tb.insert(idx, n)
        return res
```

剑指 Offer 52. 两个链表的第一个公共节点

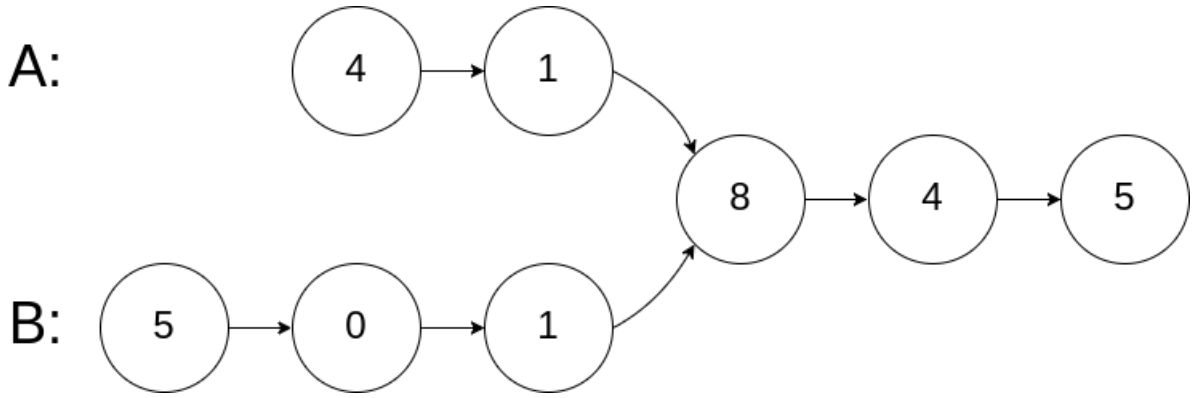
输入两个链表，找出它们的第一个公共节点。

如下面的两个链表:



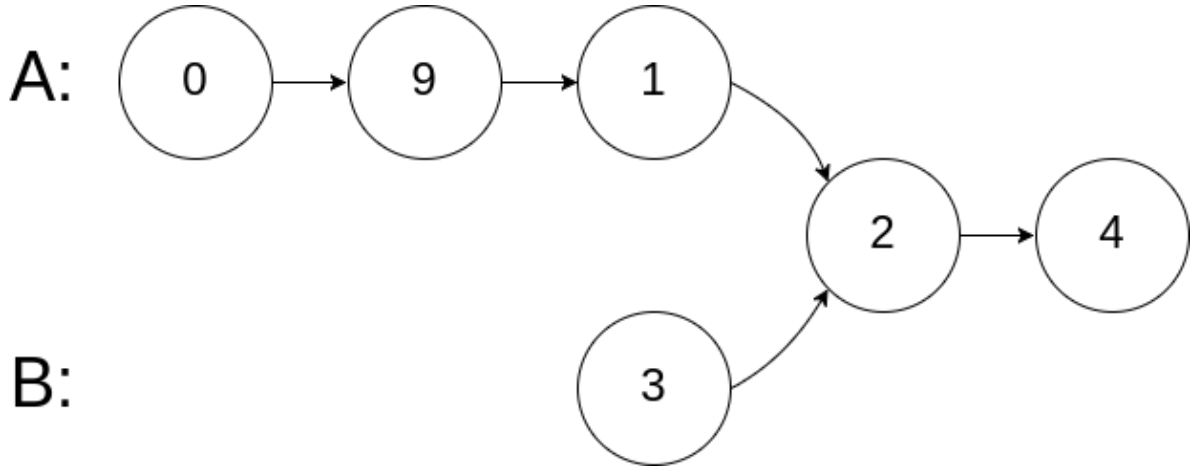
在节点 c1 开始相交。

示例 1:



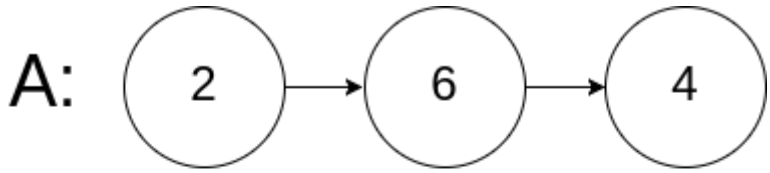
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

示例 2：



输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
输出：Reference of the node with value = 2
输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3：



输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出：null
输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
解释：这两个链表不相交，因此返回 null。

注意：

如果两个链表没有交点，返回 null.
在返回结果后，两个链表仍须保持原有的结构。
可假定整个链表结构中没有循环。
程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。
本题与主站 160 题相同：https://leetcode-cn.com/problems/intersection-of-two-linked-lists/

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        l1,l2 = headA,headB
        while(l1 != l2):
            if not l1:
                l1 = headB
            else:
                l1 = l1.next
            if not l2:
                l2 = headA
            else:
                l2 = l2.next
        return l2
```

剑指 Offer 53 - I. 在排序数组中查找数字 I

统计一个数字在排序数组中出现的次数。

示例 1:

输入: nums = [5,7,7,8,8,10], target = 8
输出: 2
示例 2:

输入: nums = [5,7,7,8,8,10], target = 6
输出: 0

提示:

0 <= nums.length <= 105
-109 <= nums[i] <= 109
nums 是一个非递减数组
-109 <= target <= 109

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:

        return self.rightIndex(nums,target) - self.leftIndex(nums,target)+1

    def leftIndex(self,nums,target):
        left,right = 0,len(nums)-1
        while(left<=right):
            mid = left +(right-left)//2
            if nums[mid]< target:
                left = mid+1
            elif nums[mid] > target:
                right = mid-1
            else:
                right = mid-1
        return left

    def rightIndex(self,nums,target):
        left,right = 0,len(nums)-1
        while(left<=right):
            mid = left +(right-left)//2
            if nums[mid]< target:
                left = mid+1
            elif nums[mid] > target:
                right = mid-1
            else:
                left = mid+1
        return right
```

剑指 Offer 53 - II. 0～n-1中缺失的数字

一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0～n-1之内。在范围0～n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。

示例 1:

输入: [0,1,3]
输出: 2
示例 2:

输入: [0,1,2,3,4,5,6,7,9]
输出: 8

限制:

1 <= 数组长度 <= 10000

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:

        for i in range(len(nums)):
            if nums[i] != i:
                return i
        return len(nums)
```

剑指 Offer 54. 二叉搜索树的第k大节点

给定一棵二叉搜索树，请找出其中第 k 大的节点的值。

示例 1:

输入: root = [3,1,4,null,2], k = 1

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

输出: 4
示例 2:

输入: root = [5,3,6,2,4,null,null,1], k = 3

```
       5
      / \
     3   6
    / \
   2   4
  /
 1
```

输出: 4

限制:

1 ≤ k ≤ 二叉搜索树元素个数

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def kthLargest(self, root: TreeNode, k: int) -> int:
        self.res = []
        self.traverse(root)
        return self.res[-k]
    def traverse(self,root):
        if root == None: return

        self.traverse(root.left)
        self.res.append(root.val)
        self.traverse(root.right)
```

剑指 Offer 55 - I. 二叉树的深度

输入一棵二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度。

例如:

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 。

提示:

节点总数 <= 10000

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
```

```
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        if root == None: return 0

        leftDeep = self.maxDepth(root.left)
        rightDeep = self.maxDepth(root.right)

        return max(leftDeep,rightDeep) +1
```

剑指 Offer 55 - II. 平衡二叉树

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

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
  / \
 4   4
```

返回 false 。

限制：

0 <= 树的结点个数 <= 10000

```
# Definition for a binary tree node.
# class TreeNode:
```

```
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        self.isBalanced = True
        self.deep(root)
        return self.isBalanced

    def deep(self,root):
        if root == None: return 0
        leftDeep = self.deep(root.left)
        rightDeep = self.deep(root.right)

        if abs(leftDeep-rightDeep)>1:
            self.isBalanced = False

        return max(leftDeep,rightDeep)+1
```

剑指 Offer 56 - I. 数组中数字出现的次数

一个整型数组 nums 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

示例 1：

输入：nums = [4,1,4,6]
输出：[1,6] 或 [6,1]
示例 2：

输入：nums = [1,2,10,4,1,4,3,3]
输出：[2,10] 或 [10,2]

限制：

2 <= nums.length <= 10000

```
class Solution:
    def singleNumbers(self, nums: List[int]) -> List[int]:
        dicNums= {i:0 for i in nums}
        res = []
        for x in nums:
            dicNums[x] +=1
        for k,v in dicNums.items():
            if v==1:
                res.append(k)
        return res
```

剑指 Offer 56 - II. 数组中数字出现的次数 II

在一个数组 nums 中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字。

示例 1：

输入：nums = [3,4,3,3]
输出：4
示例 2：

输入：nums = [9,1,7,9,7,9,7]
输出：1

限制：

1 <= nums.length <= 10000
1 <= nums[i] < 2^31

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        dicNums= {i:0 for i in nums}
        for x in nums:
            dicNums[x] +=1
        for k,v in dicNums.items():
            if v==1:
                return k
```

剑指 Offer 57. 和为s的两个数字

输入一个递增排序的数组和一个数字s，在数组中查找两个数，使得它们的和正好是s。如果有多对数字的和等于s，则输出任意一对即可。

示例 1：

输入：nums = [2,7,11,15], target = 9
输出：[2,7] 或者 [7,2]
示例 2：

输入：nums = [10,26,30,31,47,60], target = 40
输出：[10,30] 或者 [30,10]

限制：

1 <= nums.length <= 10^5
1 <= nums[i] <= 10^6

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        left ,right = 0,len(nums)-1
        while(left<right):
            total = nums[left]+nums[right]
            if (total<target):
                left +=1
            elif(total>target):
                right -=1
            else:
                return [nums[left],nums[right]]
```

剑指 Offer 57 - II. 和为s的连续正数序列

输入一个正整数 target ，输出所有和为 target 的连续正整数序列（至少含有两个数）。

序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

示例 1：

输入：target = 9
输出：[[2,3,4],[4,5]]
示例 2：

输入：target = 15
输出：[[1,2,3,4,5],[4,5,6],[7,8]]

限制：

1 <= target <= 10^5

```python
class Solution:
    def findContinuousSequence(self, target: int) -> List[List[int]]:
        window  = []
        res = []
        right = 1
        while(right<target):
            window.append(right)
            right +=1
            while(sum(window)>=target):
                if sum(window)==target:
                    res.append(window[:])
                window.pop(0)
        return res
```

剑指 Offer 58 - I. 翻转单词顺序

输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am a student. "，则输出"student. a am I"。

示例 1：

输入: "the sky is blue"
输出: "blue is sky the"
示例 2：

输入: " hello world! "
输出: "world! hello"
解释: 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
示例 3：

输入: "a good   example"
输出: "example good a"
解释: 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

说明：

无空格字符构成一个单词。
输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

```
class Solution:
    def reverseWords(self, s: str) -> str:
        listS = [x for x in s.split(" ") if x]
        return ' '.join(listS[::-1])
```

剑指 Offer 58 - II. 左旋转字符串

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

示例 1：

输入: s = "abcdefg", k = 2
输出: "cdefgab"
示例 2：

输入: s = "lrloseumgh", k = 6
输出: "umghlrlose"

限制：

1 <= k < s.length <= 10000

```python
class Solution:
    def reverseLeftWords(self, s: str, n: int) -> str:
        listS = list(s)
        listS[:n] = reversed(listS[:n])
        listS[n:] = reversed(listS[n:])
        listS.reverse()
        return ''.join(listS)
```

剑指 Offer 59 - I. 滑动窗口的最大值

给定一个数组 nums 和滑动窗口的大小 k，请找出所有滑动窗口里的最大值。

示例:

输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出: [3,3,5,5,6,7]
解释:

```
  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

```python
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        deque = [];result = [] # deque也可以用collection里的双端队列实现
        for i in range(0, len(nums)):
            while deque and nums[i]>nums[deque[-1]]: # 只存有可能成为最大值的数字的index进deque
                deque.pop()
            deque.append(i)
            while i-deque[0]>k-1: # 如果相距超过窗口k长度则弃掉
                deque.pop(0)
            if i >= k-1:
                result.append(nums[deque[0]]) # 这过程中始终保持deque[0]为最大值的index
        return result
```

(理解)剑指 Offer 60. n个骰子的点数

把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。

你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第 i 小的那个的概率。

示例 1:

输入: 1
输出: [0.16667,0.16667,0.16667,0.16667,0.16667,0.16667]
示例 2:

输入: 2
输出:
[0.02778,0.05556,0.08333,0.11111,0.13889,0.16667,0.13889,0.11111,0.08333,0.05556
,0.02778]

限制:

1 <= n <= 11

```
class Solution:
    def dicesProbability(self, n: int) -> List[float]:
        '''
            dp[i][j]表示i个骰子，和为j时的概率
            为了保证dp[n]是n个骰子，所以要从把dp长度设置为n+1，即下标从0到n。其中dp[0]无意义,d[i][0]也无意义
            【注意】i个骰子，其和是从i开始的，要到6i，所以范围要用6i+1
        '''
        dp = [ [0]*(6*i+1) for i in range(n+1)]
        p = 1/6

        # i=1时，初始化一个骰子的各种数的可能性，从1到6
        for k in range(1,7):
            dp[1][k] = p

        '''
            i个骰子，和为j:
                i：从2开始，到n
                j：从i开始到6i
        '''
        for i in range(2,n+1):
            for j in range(i,6*i+1):
                dp[i][j] = sum(dp[i-1][max(1,j-6):j])*p
                '''
                当最后一个骰子点数为1时，前i-1个骰子的点数和必须为j-1,概率为1/6;
                当最后一个骰子点数为2时，前n-1个骰子的点数和必须为j-2;
                。。。。。
                以此类推，
                当后一个点数为6，前i-1个骰子的点数和必须为k-6

                和*概率 = 当最后一个骰子点数为1时前i-1个骰子的点数和必须为j-1 * 1/6 + 当最后一个骰子点数为2时前n-1个骰子的点数和必须为j-2 * 1/6 = （最后点数为1 +最后点数为2+....+最后点数为6）* 1/6
                '''
        return dp[n][n:]
```

剑指 Offer 61. 扑克牌中的顺子

从若干副扑克牌中随机抽 5 张牌，判断是不是一个顺子，即这5张牌是不是连续的。2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A 不能视为 14。

示例 1:

输入: [1,2,3,4,5]
输出: True

示例 2:

输入: [0,0,1,2,5]
输出: True

```python
class Solution:
    def isStraight(self, nums: List[int]) -> bool:
        repeat = set()
        ma,mi = 0,14
        for num in nums:
            if num == 0:continue
            ma = max(ma,num)
            mi = min(mi,num)
            if num in repeat: return False
            repeat.add(num)
        return ma - mi < 5
```

剑指 Offer 62. 圆圈中最后剩下的数字

0,1,···,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字（删除后从下一个数字开始计数）。求出这个圆圈里剩下的最后一个数字。

例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。

示例 1：

输入: n = 5, m = 3
输出: 3
示例 2：

输入: n = 10, m = 17
输出: 2

限制：

```
1 <= n <= 10^5
1 <= m <= 10^6
```

```python
class Solution:
    def lastRemaining(self, n: int, m: int) -> int:
        f = 0
        for i in range(2, n + 1):
            f = (m + f) % i
        return f
```

剑指 Offer 63. 股票的最大利润

假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？

示例 1:

输入: [7,1,5,3,6,4]
输出: 5
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格。
示例 2:

输入: [7,6,4,3,1]
输出: 0
解释: 在这种情况下, 没有交易完成, 所以最大利润为 0。

限制：

0 <= 数组长度 <= 10^5

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices: return 0
        dp =[[0]*2 for _ in range(len(prices))]
        dp[0][0] = 0
        dp[0][1] = -prices[0]

        for i in range(1,len(prices)):
            dp[i][0] = max(dp[i-1][0],dp[i-1][1]+prices[i])
            dp[i][1] = max(dp[i-1][1],-prices[i])

        return dp[len(prices)-1][0]
```

剑指 Offer 64. 求1+2+...+n

求 1+2+...+n ，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

示例 1:

输入: n = 3
输出: 6
示例 2:

输入: n = 9
输出: 45

限制:

1 <= n <= 10000

```
class Solution:
    def sumNums(self, n: int) -> int:
        return sum(list(range(1, n+1)))
```

剑指 Offer 65. 不用加减乘除做加法

写一个函数，求两个整数之和，要求在函数体内不得使用 “+”、“-”、“*”、“/” 四则运算符号。

示例:

输入: a = 1, b = 1
输出: 2

提示:

a, b 均可能是负数或 0
结果不会溢出 32 位整数

```
class Solution:
    def add(self, a: int, b: int) -> int:
        return sum([a,b])
```

剑指 Offer 66. 构建乘积数组

给定一个数组 A[0,1,…,n-1]，请构建一个数组 B[0,1,…,n-1]，其中 B[i] 的值是数组 A 中除了下标 i 以外的元素的积，即 B[i]=A[0]×A[1]×…×A[i-1]×A[i+1]×…×A[n-1]。不能使用除法。

示例：

输入：[1,2,3,4,5]
输出：[120,60,40,30,24]

提示：

所有元素乘积之和不会溢出 32 位整数
a.length <= 100000

```python
class Solution:
    def constructArr(self, a: List[int]) -> List[int]:
        n = len(a)
        if n==0: return []
        prefix = [1] *len(a)
        prefix[0] = a[0]
        suffix = [1] *len(a)
        suffix[n-1] = a[n-1]

        for i in range(1,n):
            prefix[i] = prefix[i-1]*a[i]
        for i in range(n-2,-1,-1):
            suffix[i] = suffix[i+1]*a[i]
        res = [0]*n
        res[0] = suffix[1]
        res[n-1] = prefix[n-2]

        for i in range(1,n-1):
            res[i] = prefix[i-1]*suffix[i+1]
        return res
```

剑指 Offer 67. 把字符串转换成整数

写一个函数 StrToInt，实现把字符串转换成整数这个功能。不能使用 atoi 或者其他类似的库函数。

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。

当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。

该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0。

说明：

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [−231, 231 − 1]。如果数值超过这个范围，请返回 INT_MAX (231 − 1) 或 INT_MIN (−231) 。

示例 1:

输入: "42"
输出: 42
示例 2:

输入: "   -42"
输出: -42
解释: 第一个非空白字符为 '-', 它是一个负号。
     我们尽可能将负号与后面所有连续出现的数字组合起来，最后得到 -42 。
示例 3:

输入: "4193 with words"
输出: 4193
解释: 转换截止于数字 '3' ，因为它的下一个字符不为数字。
示例 4:

输入: "words and 987"
输出: 0
解释: 第一个非空字符是 'w', 但它不是数字或正、负号。
     因此无法执行有效的转换。
示例 5:

输入: "-91283472332"
输出: -2147483648
解释: 数字 "-91283472332" 超过 32 位有符号整数范围。
     因此返回 INT_MIN (−231) 。

```python
class Solution:
    def strToInt(self, str: str) -> int:
        st = str.lstrip(" ")
        if not st:
            return 0

        pos = 1
        if st[0] in '+-':
            pos = 1 if st[0] == '+' else -1
            st = st[1:]

        ans = ""
        for va in st:
            if va.isnumeric():
                ans += va
            else:
                break

        if not ans:
            return 0
```

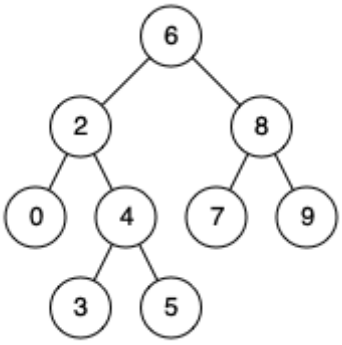
```
        num = int(ans)*pos
        if num >= 2**31-1:
            return 2**31-1
        if num <= -2**31:
            return -2**31
        return num
```

剑指 Offer 68 - I. 二叉搜索树的最近公共祖先

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉搜索树: root = [6,2,8,0,4,7,9,null,null,3,5]



示例 1:

输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出: 6
解释: 节点 2 和节点 8 的最近公共祖先是 6。
示例 2:

输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
输出: 2
解释: 节点 2 和节点 4 的最近公共祖先是 2, 因为根据定义最近公共祖先节点可以为节点本身。

说明:

所有节点的值都是唯一的。
p、q 为不同节点且均存在于给定的二叉搜索树中。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
```

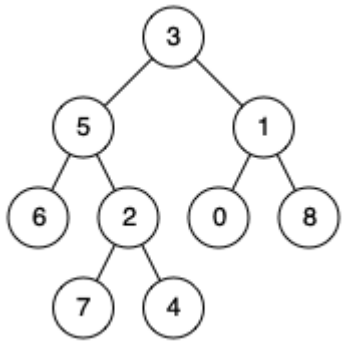
```
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        if root == None: return None

        if p.val>q.val:
            return self.lowestCommonAncestor(root,q,p)
        if p.val<=root.val and q.val>=root.val:
            return root
        if p.val>root.val:
            return self.lowestCommonAncestor(root.right,p,q)
        else:
            return self.lowestCommonAncestor(root.left,p,q)
```

剑指 Offer 68 - II. 二叉树的最近公共祖先

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉树: root = [3,5,1,6,2,0,8,null,null,7,4]



示例 1:

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
示例 2:

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
输出: 5
解释: 节点 5 和节点 4 的最近公共祖先是节点 5。因为根据定义最近公共祖先节点可以为节点本身。

说明:

所有节点的值都是唯一的。
p、q 为不同节点且均存在于给定的二叉树中。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: TreeNode, p: TreeNode, q: TreeNode) ->
TreeNode:
        if root == None: return None
        if root == p or root == q: return root

        leftNode = self.lowestCommonAncestor(root.left,p,q)
        rightNode = self.lowestCommonAncestor(root.right,p,q)

        if leftNode != None and rightNode != None:
            return root
        if leftNode == None and rightNode == None:
            return None
        return leftNode if leftNode!=None else rightNode
```

剑指 Offer II 001. 整数除法

给定两个整数 a 和 b ，求它们的除法的商 a/b ，要求不得使用乘号 '*'、除号 '/' 以及求余符号 '%' 。

注意：

整数除法的结果应当截去（truncate）其小数部分，例如：truncate(8.345) = 8 以及 truncate(-2.7335) = -2
假设我们的环境只能存储 32 位有符号整数，其数值范围是 [−231, 231−1]。本题中，如果除法结果溢出，则返回 231 − 1

示例 1：

输入：a = 15, b = 2
输出：7
解释：15/2 = truncate(7.5) = 7
示例 2：

输入：a = 7, b = -3
输出：-2
解释：7/-3 = truncate(-2.33333..) = -2
示例 3：

输入：a = 0, b = 1
输出：0
示例 4：

输入：a = 1, b = 1
输出：1

提示：

-231 <= a, b <= 231 - 1
b != 0

```python
class Solution:
    def divide(self, a: int, b: int) -> int:
        if a >0 and b>0 or a<0 and b<0: flag = False
        else: flag = True
        a,b = abs(a),abs(b)
        res = 0
        while(a>=b):
            n = 1
            t = b
            while a > t<<1:
                n <<= 1
                t <<= 1
            a -= t
            res +=n
        if flag: res = -res
        return res if -2**31 <=res <= 2**31-1 else 2**31 - 1
```

剑指 Offer II 002. 二进制加法

给定两个 01 字符串 a 和 b ，请计算它们的和，并以二进制字符串的形式输出。

输入为 非空 字符串且只包含数字 1 和 0。

示例 1:

输入: a = "11", b = "10"
输出: "101"
示例 2:

输入: a = "1010", b = "1011"
输出: "10101"

提示：

每个字符串仅由字符 '0' 或 '1' 组成。
1 <= a.length, b.length <= 10^4
字符串如果不是 "0" ，就都不含前导零。

```python
class Solution:
```

```python
    def addBinary(self, a: str, b: str) -> str:
        res , carry = '', 0

        while a or b or carry:
            tmp = int(a[-1] if a else 0) + int(b[-1] if b else 0) + carry
            if tmp>1:
                tmp = tmp -2
                carry = 1
            else :
                carry = 0
            res += str(tmp)

            a,b = a[:-1],b[:-1]
        return res[::-1]
```

剑指 Offer II 003. 前 n 个数字二进制中 1 的个数

给定一个非负整数 n ，请计算 0 到 n 之间的每个数字的二进制表示中 1 的个数，并输出一个数组。

示例 1:

输入: n = 2
输出: [0,1,1]
解释:
0 --> 0
1 --> 1
2 --> 10
示例 2:

输入: n = 5
输出: [0,1,1,2,1,2]
解释:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101

说明 :

0 <= n <= 105

进阶:

给出时间复杂度为 O(n*sizeof(integer)) 的解答非常容易。但你可以在线性时间 O(n) 内用一趟扫描做到吗?
要求算法的空间复杂度为 O(n) 。

你能进一步完善解法吗？要求在C++或任何其他语言中不使用任何内置函数（如 C++ 中的 __builtin_popcount ）来执行此操作。

```
class Solution:
    def countBits(self, n: int) -> List[int]:
        res = []

        for i in range(0,n+1):
            count = 0
            while(i>0):
                i = i&(i-1)
                count +=1
            res.append(count)
        return res
```

剑指 Offer II 004. 只出现一次的数字

给你一个整数数组 nums ，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次 。请你找出并返回那个只出现了一次的元素。

示例 1：

输入：nums = [2,2,3,2]
输出：3
示例 2：

输入：nums = [0,1,0,1,0,1,100]
输出：100

提示：

1 <= nums.length <= 3 * 104
-231 <= nums[i] <= 231 - 1
nums 中，除某个元素仅出现 一次 外，其余每个元素都恰出现 三次

进阶：你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        resNums ={i:0 for i in nums}
        for x in nums:
            resNums[x] +=1
        for k,v in resNums.items():
            if v == 1:
                return k
```

剑指 Offer II 005. 单词长度的最大乘积

给定一个字符串数组 words，请计算当两个字符串 words[i] 和 words[j] 不包含相同字符时，它们长度的乘积的最大值。假设字符串中只包含英语的小写字母。如果没有不包含相同字符的一对字符串，返回 0。

示例 1:

输入: words = ["abcw","baz","foo","bar","fxyz","abcdef"]
输出: 16
解释: 这两个单词为 "abcw", "fxyz"。它们不包含相同字符，且长度的乘积最大。
示例 2:

输入: words = ["a","ab","abc","d","cd","bcd","abcd"]
输出: 4
解释: 这两个单词为 "ab", "cd"。
示例 3:

输入: words = ["a","aa","aaa","aaaa"]
输出: 0
解释: 不存在这样的两个单词。

提示:

2 <= words.length <= 1000
1 <= words[i].length <= 1000
words[i] 仅包含小写字母

```python
class Solution:
    def maxProduct(self, words: List[str]) -> int:
        res = 0
        n = len(words)
        for i in range(n):
            for j in range(i,n):
                wordsi = set(list(words[i]))
                wordsj = set(list(words[j]))
                if not (wordsi & wordsj):
                    res =max(res,len(words[i])*len(words[j]))
        return res
```

剑指 Offer II 006. 排序数组中两个数字之和

给定一个已按照 升序排列 的整数数组 numbers ，请你从数组中找出两个数满足相加之和等于目标数 target 。

函数应该以长度为 2 的整数数组的形式返回这两个数的下标值。numbers 的下标 从 0 开始计数 ，所以答案数组应当满足 0 <= answer[0] < answer[1] < numbers.length 。

假设数组中存在且只存在一对符合条件的数字，同时一个数字不能使用两次。

示例 1：

输入：numbers = [1,2,4,6,10], target = 8
输出：[1,3]
解释：2 与 6 之和等于目标数 8 。因此 index1 = 1, index2 = 3 。
示例 2：

输入：numbers = [2,3,4], target = 6
输出：[0,2]
示例 3：

输入：numbers = [-1,0], target = -1
输出：[0,1]

提示：

2 <= numbers.length <= 3 * 104
-1000 <= numbers[i] <= 1000
numbers 按 递增顺序 排列
-1000 <= target <= 1000
仅存在一个有效答案

```python
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        left ,right = 0,len(numbers)-1
        while(left<right):
            total = numbers[left]+numbers[right]
            if (total<target):
                left +=1
            elif(total>target):
                right -=1
            else:
                return [left,right]
```

剑指 Offer II 007. 数组中和为 0 的三个数

给你一个整数数组 nums ，判断是否存在三元组 [nums[i], nums[j], nums[k]] 满足 i != j、i != k 且 j != k ，同时还满足 nums[i] + nums[j] + nums[k] == 0 。请

你返回所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例 1：

输入：nums = [-1,0,1,2,-1,-4]
输出：[[-1,-1,2],[-1,0,1]]
解释：
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0 。
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0 。
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0 。
不同的三元组是 [-1,0,1] 和 [-1,-1,2] 。
注意，输出的顺序和三元组的顺序并不重要。
示例 2：

输入：nums = [0,1,1]
输出：[]
解释：唯一可能的三元组和不为 0 。
示例 3：

输入：nums = [0,0,0]
输出：[[0,0,0]]
解释：唯一可能的三元组和为 0 。

提示：

3 <= nums.length <= 3000
-105 <= nums[i] <= 105

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        n = len(nums)
        res = set()
        for i in range(n):
            j=i+1
            k=n-1
            target = 0-nums[i]
            while(j<k):
                total = nums[j]+nums[k]
                if total >target:
                    k -=1
                elif total <target:
                    j +=1
                else:
                    res.add((nums[i],nums[j],nums[k]))
                    k -=1
                    j +=1
        return list(res)
```

剑指 Offer II 008. 和大于等于 target 的最短子数组

给定一个含有 n 个正整数的数组和一个正整数 target 。

找出该数组中满足其和 ≥ target 的长度最小的 连续子数组 [numsl, numsl+1, ..., numsr-1, numsr] ，并返回其长度。如果不存在符合条件的子数组，返回 0 。

示例 1：

输入：target = 7, nums = [2,3,1,2,4,3]
输出：2
解释：子数组 [4,3] 是该条件下的长度最小的子数组。
示例 2：

输入：target = 4, nums = [1,4,4]
输出：1
示例 3：

输入：target = 11, nums = [1,1,1,1,1,1,1,1]
输出：0

提示：

1 <= target <= 109
1 <= nums.length <= 105
1 <= nums[i] <= 105

进阶：

如果你已经实现 O(n) 时间复杂度的解法，请尝试设计一个 O(n log(n)) 时间复杂度的解法。

```python
class Solution:
    def minSubArrayLen(self, target: int, nums: List[int]) -> int:

        window = []
        n = len(nums)
        left = right = 0
        res = n
        while(right<n):
            window.append(nums[right])
            right +=1
            while(sum(window)>=target):
                res = min(res,len(window))
                window.pop(0)
        return res if sum(nums)>=target else 0
```

剑指 Offer II 009. 乘积小于 K 的子数组

给定一个正整数数组 nums和整数 k ，请找出该数组内乘积小于 k 的连续的子数组的个数。

示例 1:

输入: nums = [10,5,2,6], k = 100
输出: 8
解释: 8 个乘积小于 100 的子数组分别为: [10], [5], [2], [6], [10,5], [5,2], [2,6], [5,2,6]。
需要注意的是 [10,5,2] 并不是乘积小于100的子数组。
示例 2:

输入: nums = [1,2,3], k = 0
输出: 0

提示:

1 <= nums.length <= 3 * 104
1 <= nums[i] <= 1000
0 <= k <= 106

```python
class Solution:
    def numSubarrayProductLessThanK(self, nums, k):
        left = ret = 0
        total = 1
        for right, num in enumerate(nums):
            total *= num
            while left <= right and total >= k:
                total //= nums[left]
                left += 1
            if left <= right:
                ret += right - left + 1
        return ret
```

剑指 Offer II 010. 和为 k 的子数组

给定一个整数数组和一个整数 k ，请找到该数组中和为 k 的连续子数组的个数。

示例 1:

输入:nums = [1,1,1], k = 2
输出: 2
解释: 此题 [1,1] 与 [1,1] 为两种不同的情况
示例 2:

输入:nums = [1,2,3], k = 3
输出: 2

提示：

1 <= nums.length <= 2 * 104
-1000 <= nums[i] <= 1000
-107 <= k <= 107

```python
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        n = len(nums)
        hashmap = {}
        ans = 0
        preSum = [0 for i in range(n + 1)]
        hashmap[0] = 1
        for i in range(1, n + 1):
            preSum[i] = preSum[i - 1] + nums[i - 1]
            if preSum[i] - k in hashmap:
                ans += hashmap[preSum[i] - k]
            if preSum[i] in hashmap:
                hashmap[preSum[i]] += 1
            else:
                hashmap[preSum[i]] = 1
        return ans
```

剑指 Offer II 011. 0 和 1 个数相同的子数组

给定一个二进制数组 nums ，找到含有相同数量的 0 和 1 的最长连续子数组，并返回该子数组的长度。

示例 1：

输入: nums = [0,1]
输出: 2
说明: [0, 1] 是具有相同数量 0 和 1 的最长连续子数组。
示例 2：

输入: nums = [0,1,0]
输出: 2
说明: [0, 1] (或 [1, 0]) 是具有相同数量 0 和 1 的最长连续子数组。

提示：

1 <= nums.length <= 105
nums[i] 不是 0 就是 1

```python
class Solution:
    def findMaxLength(self, nums: List[int]) -> int:
        n = len(nums)
```

```
        prefix = [0]*(n+1)
        dic = {}
        dic[0] = 0
        res = 0
        for i in range(1,n+1):
            tmp = 1 if nums[i-1] ==1 else -1
            prefix[i] = prefix[i-1] + tmp
            if prefix[i] in dic:
                res = max(res,i-dic[prefix[i]])
            else:
                dic[prefix[i]] =i
        return res
```

剑指 Offer II 012. 左右两边子数组的和相等

给你一个整数数组 nums ，请计算数组的 中心下标 。

数组 中心下标 是数组的一个下标，其左侧所有元素相加的和等于右侧所有元素相加的和。

如果中心下标位于数组最左端，那么左侧数之和视为 0 ，因为在下标的左侧不存在元素。这一点对于中心下标位于数组最右端同样适用。

如果数组有多个中心下标，应该返回 最靠近左边 的那一个。如果数组不存在中心下标，返回 -1 。

示例 1：

输入：nums = [1,7,3,6,5,6]
输出：3
解释：
中心下标是 3 。
左侧数之和 sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11 ，
右侧数之和 sum = nums[4] + nums[5] = 5 + 6 = 11 ，二者相等。
示例 2：

输入：nums = [1, 2, 3]
输出：-1
解释：
数组中不存在满足此条件的中心下标。
示例 3：

输入：nums = [2, 1, -1]
输出：0
解释：
中心下标是 0 。
左侧数之和 sum = 0 ，（下标 0 左侧不存在元素），
右侧数之和 sum = nums[1] + nums[2] = 1 + -1 = 0 。

提示：

1 <= nums.length <= 104
-1000 <= nums[i] <= 1000

```python
class Solution:
    def pivotIndex(self, nums: List[int]) -> int:
        for i in range(len(nums)):
            if i==0: leftSum =0
            else: leftSum =  sum(nums[:i])
            if i==len(nums)-1: rightSum = 0
            else: rightSum = sum(nums[i+1:])
            if leftSum==rightSum:
                return i
        return -1
```

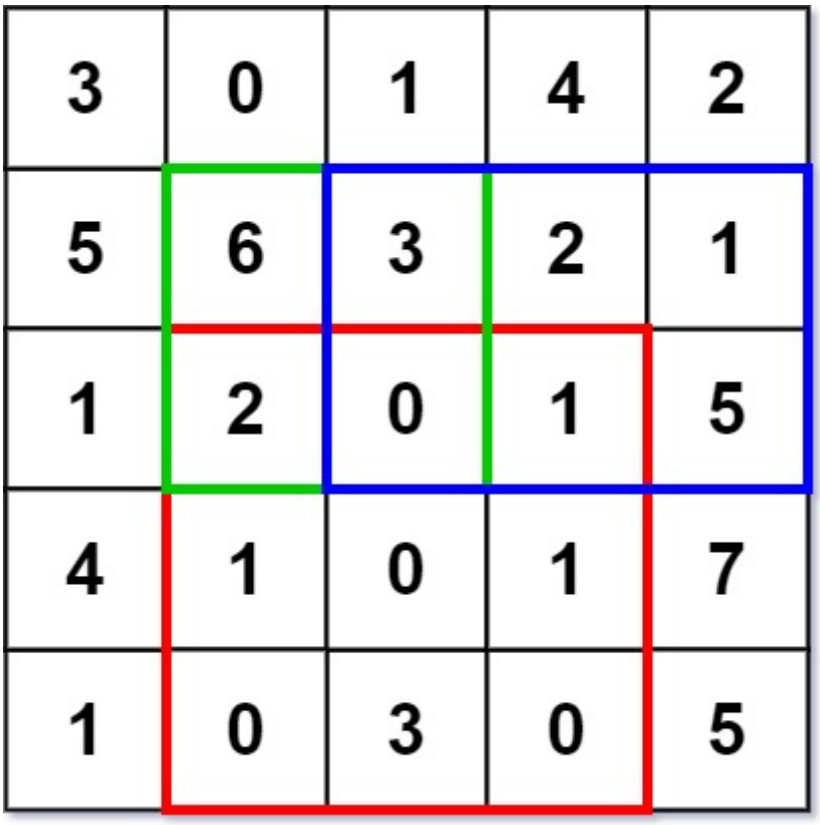
剑指 Offer II 013. 二维子矩阵的和

给定一个二维矩阵 matrix，以下类型的多个请求：

计算其子矩形范围内元素的总和，该子矩阵的左上角为 (row1, col1) ，右下角为 (row2, col2) 。
实现 NumMatrix 类：

NumMatrix(int[][] matrix) 给定整数矩阵 matrix 进行初始化
int sumRegion(int row1, int col1, int row2, int col2) 返回左上角 (row1, col1) 、右下角 (row2, col2) 的子矩阵的元素总和。

示例 1：



输入：
["NumMatrix","sumRegion","sumRegion","sumRegion"]

```
[[[[3,0,1,4,2],[5,6,3,2,1],[1,2,0,1,5],[4,1,0,1,7],[1,0,3,0,5]]],[2,1,4,3],
[1,1,2,2],[1,2,2,4]]
输出:
[null, 8, 11, 12]

解释:
NumMatrix numMatrix = new NumMatrix([[3,0,1,4,2],[5,6,3,2,1],[1,2,0,1,5],
[4,1,0,1,7],[1,0,3,0,5]]]);
numMatrix.sumRegion(2, 1, 4, 3); // return 8 (红色矩形框的元素总和)
numMatrix.sumRegion(1, 1, 2, 2); // return 11 (绿色矩形框的元素总和)
numMatrix.sumRegion(1, 2, 2, 4); // return 12 (蓝色矩形框的元素总和)


提示:

m == matrix.length
n == matrix[i].length
1 <= m, n <= 200
-105 <= matrix[i][j] <= 105
0 <= row1 <= row2 < m
0 <= col1 <= col2 < n
最多调用 104 次 sumRegion 方法
```

```python
class NumMatrix:

    def __init__(self, matrix: List[List[int]]):
        m, n = len(matrix), len(matrix[0])
        self.dp = [[0] * (n + 1) for _ in range(m + 1)]
        for i in range(m):
            for j in range(n):
                self.dp[i + 1][j + 1] = self.dp[i][j + 1] + self.dp[i + 1][j] -
self.dp[i][j] + matrix[i][j]
        return

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
        return self.dp[row2 + 1][col2 + 1] - self.dp[row2 + 1][col1] -
self.dp[row1][col2 + 1] + self.dp[row1][col1]
```

剑指 Offer II 014. 字符串中的变位词

```
给定两个字符串 s1 和 s2，写一个函数来判断 s2 是否包含 s1 的某个变位词。

换句话说，第一个字符串的排列之一是第二个字符串的 子串 。



示例 1：

输入: s1 = "ab" s2 = "eidbaooo"
输出: True
解释: s2 包含 s1 的排列之一 ("ba").
示例 2：
```

输入: s1= "ab" s2 = "eidboaoo"
输出: False

提示:

1 <= s1.length, s2.length <= 104
s1 和 s2 仅包含小写字母

```python
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        n = len(s2)
        window = {i:0 for i in s2}
        need = {i:0 for i in s1}
        for x in s1: need[x] +=1
        valid = 0

        left = right= 0
        while(right<n):
            c = s2[right]
            right+=1
            if c in need:
                window[c] +=1
                if window[c] == need[c]:
                    valid +=1

            while(right - left>=len(s1)):
                if valid== len(need):
                    return True
                d = s2[left]
                left +=1
                if d in need:
                    if window[d] == need[d]:
                        valid -=1
                    window[d] -=1
        return False
```

剑指 Offer II 015. 字符串中的所有变位词

给定两个字符串 s 和 p，找到 s 中所有 p 的 变位词 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

变位词 指字母相同，但排列不同的字符串。

示例 1:

输入: s = "cbaebabacd", p = "abc"
输出: [0,6]
解释:
起始索引等于 0 的子串是 "cba", 它是 "abc" 的变位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的变位词。

示例 2:

输入: s = "abab", p = "ab"
输出: [0,1,2]
解释:
起始索引等于 0 的子串是 "ab", 它是 "ab" 的变位词。
起始索引等于 1 的子串是 "ba", 它是 "ab" 的变位词。
起始索引等于 2 的子串是 "ab", 它是 "ab" 的变位词。

提示:

1 <= s.length, p.length <= 3 * 104
s 和 p 仅包含小写字母

```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        window = {i:0 for i in s}
        need = {i:0 for i in p}
        for x in p: need[x] +=1
        valid = 0
        left = right = 0
        n = len(s)
        res =[]
        while(right<n):
            c = s[right]
            right +=1
            if c in need:
                window[c] +=1
                if window[c]==need[c]:
                    valid+=1
            while(right-left>=len(p)):
                if valid ==len(need):
                    res.append(left)
                d = s[left]
                left +=1
                if d in need:
                    if window[d] == need[d]:
                        valid -=1
                    window[d] -=1
        return res
```

剑指 Offer II 016. 不含重复字符的最长子字符串

给定一个字符串 s ，请你找出其中不含有重复字符的 最长连续子字符串 的长度。

示例 1:

输入: s = "abcabcbb"

输出: 3
解释: 因为无重复字符的最长子字符串是 "abc"，所以其长度为 3。
示例 2:

输入: s = "bbbbb"
输出: 1
解释: 因为无重复字符的最长子字符串是 "b"，所以其长度为 1。
示例 3:

输入: s = "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
     请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
示例 4:

输入: s = ""
输出: 0

提示：

0 <= s.length <= 5 * 104
s 由英文字母、数字、符号和空格组成

```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        slow,fast=0,0
        res = 0
        window = {i:0 for i in s}
        while(fast<len(s)):
            c = s[fast]
            fast += 1
            window[c] +=1
            while(window[c]>1):
                d = s[slow]
                slow +=1
                window[d] -=1
            res = max(res,fast-slow)
        return res
```

剑指 Offer II 017. 含有所有字符的最短字符串

给定两个字符串 s 和 t 。返回 s 中包含 t 的所有字符的最短子字符串。如果 s 中不存在符合条件的子字符串，则返回空字符串 "" 。

如果 s 中存在多个符合条件的子字符串，返回任意一个。

注意： 对于 t 中重复字符，我们寻找的子字符串中该字符数量必须不少于 t 中该字符数量。

示例 1：

输入：s = "ADOBECODEBANC", t = "ABC"
输出："BANC"
解释：最短子字符串 "BANC" 包含了字符串 t 的所有字符 'A'、'B'、'C'
示例 2：

输入：s = "a", t = "a"
输出："a"
示例 3：

输入：s = "a", t = "aa"
输出：""
解释：t 中两个字符 'a' 均应包含在 s 的子串中，因此没有符合条件的子字符串，返回空字符串。

提示：

1 <= s.length, t.length <= 105
s 和 t 由英文字母组成

```python
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        left, right = 0,0
        valid = 0
        window = {i:0 for i in s}
        need = {i:0 for i in t}
        for i in t: need[i] +=1
        start = 0
        end = float('inf')
        while(right<len(s)):
            c= s[right]
            right +=1
            if c in need:
                window[c] +=1
                if window[c] == need[c]:
                    valid +=1
            while(valid==len(need)):
                if(right - left<end):
                    start = left
                    end = right - left
                d = s[left]
                left +=1
                if d in need:
                    if window[d]==need[d]:
                        valid-=1
                    window[d]-=1
        return "" if end == float('inf') else s[start:end+start]
```

剑指 Offer II 018. 有效的回文

给定一个字符串 s ，验证 s 是否是 回文串 ，只考虑字母和数字字符，可以忽略字母的大小写。

本题中，将空字符串定义为有效的 回文串 。

示例 1:

输入: s = "A man, a plan, a canal: Panama"
输出: true
解释："amanaplanacanalpanama" 是回文串
示例 2:

输入: s = "race a car"
输出: false
解释："raceacar" 不是回文串

提示：

1 <= s.length <= 2 * 105
字符串 s 由 ASCII 字符组成

```
class Solution:
    def isPalindrome(self, s: str) -> bool:
        words = set(list('abcdefghijklmnopqrstuvwxyz0123456789'))
        newS = ''
        for i in s.lower():
            if i in words:
                newS = newS+i
        return self.palindrome(newS)


    def palindrome(self,s):
        left,right = 0,len(s)-1

        while(left<right):
            if s[left]==s[right]:
                left +=1
                right-=1
            else:
                return False
        return True
```

剑指 Offer II 019. 最多删除一个字符得到回文

给定一个非空字符串 s，请判断如果 最多 从字符串中删除一个字符能否得到一个回文字符串。

示例 1:

输入: s = "aba"
输出: true
示例 2:

输入: s = "abca"
输出: true
解释: 可以删除 "c" 字符 或者 "b" 字符
示例 3:

输入: s = "abc"
输出: false

提示:

1 <= s.length <= 105
s 由小写英文字母组成

```
class Solution:
    def validPalindrome(self, s: str) -> bool:
        left,right = 0,len(s)-1
        while(left<right):
            if s[left]==s[right]:
                left +=1
                right-=1
            else:
                return self.palindrome(s,left+1,right) or
self.palindrome(s,left,right-1)
        return True


    def palindrome(self,s,left,right):

        while(left<right):
            if s[left]==s[right]:
                left +=1
                right-=1
            else:
                return False
        return True
```

(理解)剑指 Offer II 020. 回文子字符串的个数

给定一个字符串 s ，请计算这个字符串中有多少个回文子字符串。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

示例 1:

输入：s = "abc"
输出：3
解释：三个回文子串：“a”, “b”, “c”
示例 2：

输入：s = "aaa"
输出：6
解释：6个回文子串：“a”, “a”, “a”, “aa”, “aa”, “aaa”

提示：

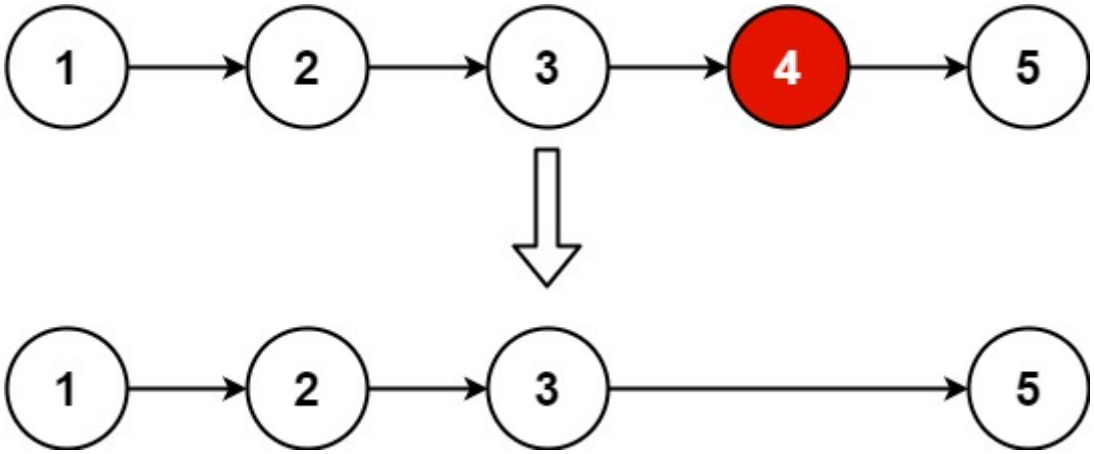
1 <= s.length <= 1000
s 由小写英文字母组成

```python
class Solution:
    def countSubstrings(self, s: str) -> int:
        n, ans = len(s), 0
        substr_map = [[0 for _ in range(n)] for _ in range(n)]
        for i in range(n-1, -1, -1):
            for j in range(i, n):
                if i == j: # 单独字符必定是回文串
                    substr_map[i][j] = 1
                elif i == j-1: # 相邻字符若相同则是回文串
                    substr_map[i][j] = 1 if s[i] == s[j] else 0
                else: # 其余情况，只与两端字符相等，且中间是回文串有关
                    substr_map[i][j] = substr_map[i+1][j-1] if s[i] == s[j] else
0
            ans += sum(substr_map[i])
        return ans
```

剑指 Offer II 021. 删除链表的倒数第 n 个结点

给定一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

示例 1：



```
输入：head = [1,2,3,4,5], n = 2
输出：[1,2,3,5]
示例 2：

输入：head = [1], n = 1
输出：[]
示例 3：

输入：head = [1,2], n = 1
输出：[1]


提示：

链表中结点的数目为 sz
1 <= sz <= 30
0 <= Node.val <= 100
1 <= n <= sz
```

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        dummy = ListNode(0, head)
        new_head = self.findFromEnd(dummy,n+1)
        new_head.next = new_head.next.next
        return dummy.next

    def findFromEnd(self,head,k) :
        L1 = L2 = head
        for i in range(k):
            L1 = L1.next
        while(L1 != None):
            L1 = L1.next
            L2 = L2.next
        return L2
```

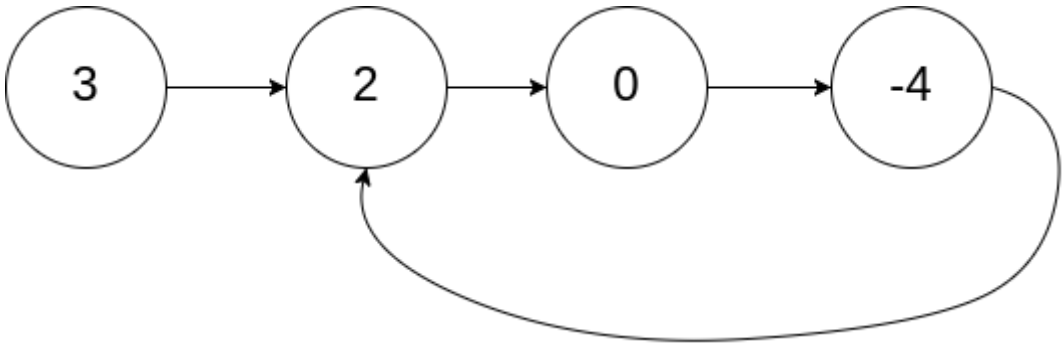
剑指 Offer II 022. 链表中环的入口节点

给定一个链表，返回链表开始入环的第一个节点。 从链表的头节点开始沿着 next 指针进入环的第一个节点为环的入口节点。如果链表无环，则返回 null。

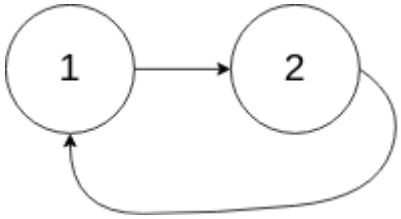
为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 pos 是 -1，则在该链表中没有环。注意，pos 仅仅是用于标识环的情况，并不会作为参数传递到函数中。

说明：不允许修改给定的链表。

示例 1：



输入：head = [3,2,0,-4], pos = 1
输出：返回索引为 1 的链表节点
解释：链表中有一个环，其尾部连接到第二个节点。
示例 2：



输入：head = [1,2], pos = 0
输出：返回索引为 0 的链表节点
解释：链表中有一个环，其尾部连接到第一个节点。
示例 3：

输入：head = [1], pos = -1
输出：返回 null
解释：链表中没有环。

提示：

链表中节点的数目范围在范围 [0, 104] 内

-105 <= Node.val <= 105
pos 的值为 -1 或者链表中的一个有效索引

进阶：是否可以使用 O(1) 空间解决此题？

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        slow = fast = head
        while(fast != None and fast.next != None):
            slow = slow.next
            fast = fast.next.next
            if (fast == slow):
                break
        if (fast == None or fast.next == None):
            return None
        slow = head
        while(slow != fast):
            slow = slow.next
            fast = fast.next
        return slow
```

剑指 Offer II 023. 两个链表的第一个重合节点

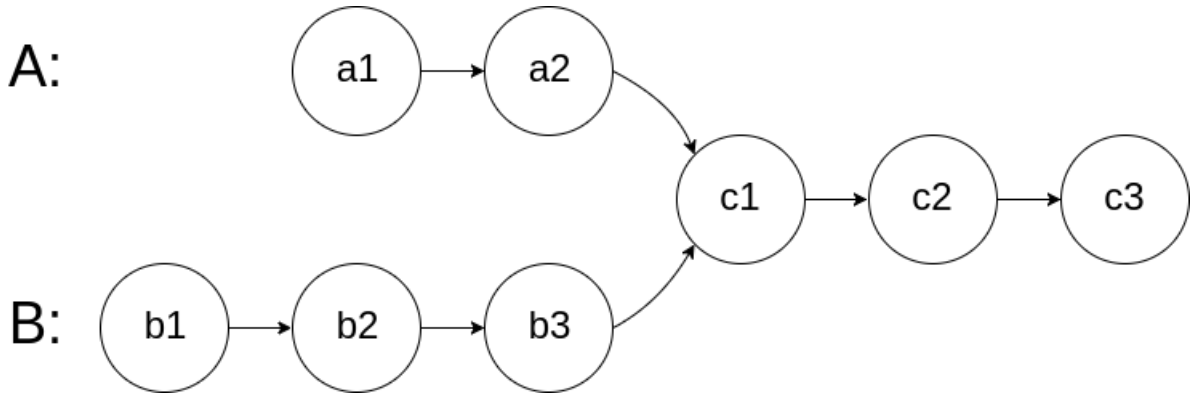
给定两个单链表的头节点 headA 和 headB ，请找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 null 。

图示两个链表在节点 c1 开始相交：

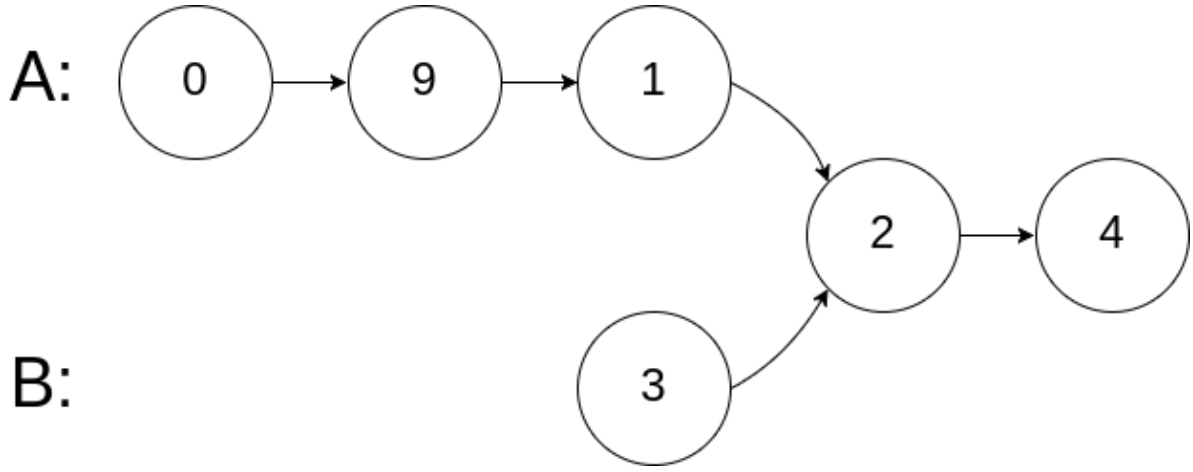
题目数据 保证 整个链式结构中不存在环。

注意，函数返回结果后，链表必须 保持其原始结构 。

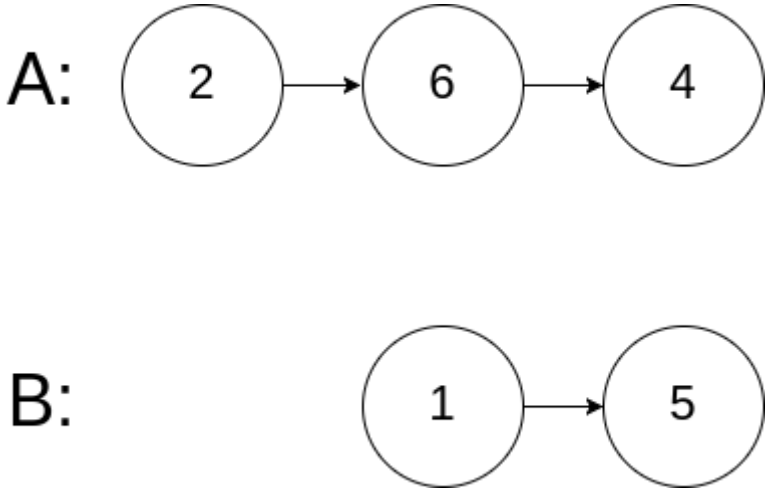
示例 1：



输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3
输出：Intersected at '8'
解释：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
示例 2：



输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
输出：Intersected at '2'
解释：相交节点的值为 2 （注意，如果两个链表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。
在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。
示例 3：



输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出：null
解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。
由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
这两个链表不相交，因此返回 null 。

提示：

listA 中节点数目为 m
listB 中节点数目为 n
0 <= m, n <= 3 * 104
1 <= Node.val <= 105
0 <= skipA <= m
0 <= skipB <= n
如果 listA 和 listB 没有交点，intersectVal 为 0
如果 listA 和 listB 有交点，intersectVal == listA[skipA + 1] == listB[skipB + 1]

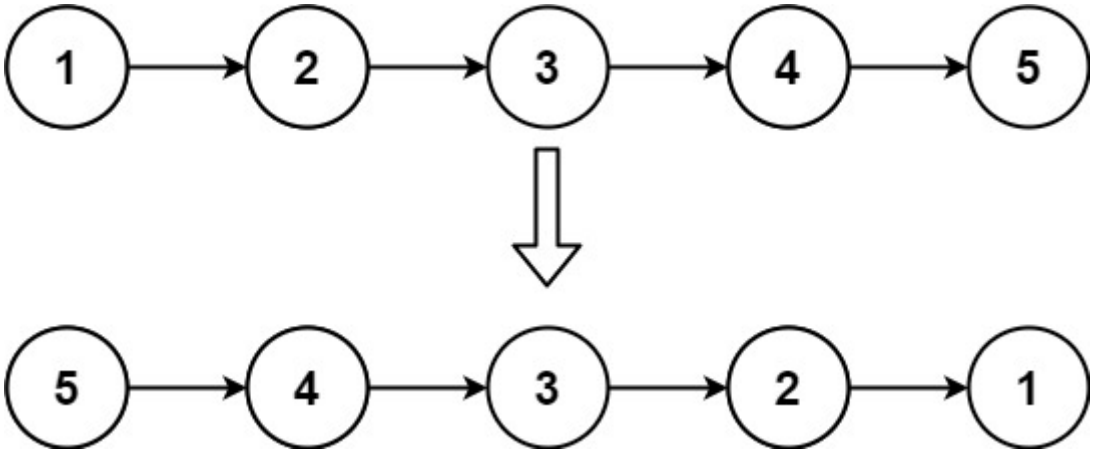
进阶：能否设计一个时间复杂度 O(n) 、仅用 O(1) 内存的解决方案？

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        L1,L2 = headA,headB
        while(L1!=L2):
            if(L1==None):L1=headB
            else:L1=L1.next
            if(L2==None):L2=headA
            else:L2=L2.next
        return L1
```

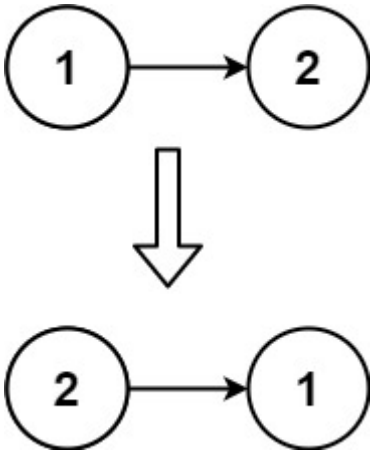
剑指 Offer II 024. 反转链表

给定单链表的头节点 head ，请反转链表，并返回反转后的链表的头节点。

示例 1：



输入：head = [1,2,3,4,5]
输出：[5,4,3,2,1]
示例 2：



输入：head = [1,2]
输出：[2,1]
示例 3：

输入：head = []
输出：[]

提示：

链表中节点的数目范围是 [0, 5000]
-5000 <= Node.val <= 5000

进阶：链表可以选用迭代或递归方式完成反转。你能否用两种方法解决这道题？

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
```

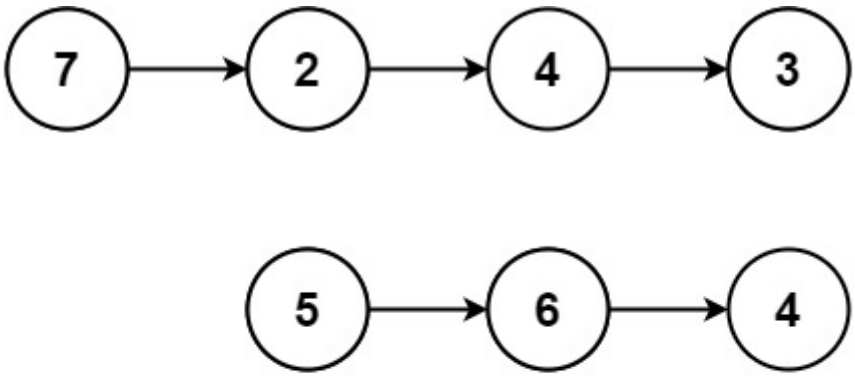
```python
    def reverseList(self, head: ListNode) -> ListNode:
        pre = None
        while head:
            tmp = head.next
            head.next = pre
            pre = head
            head = tmp
        return pre
```

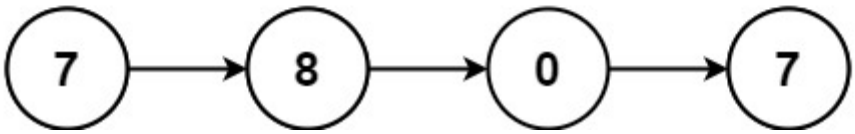
剑指 Offer II 025. 链表中的两数相加

给定两个 非空链表 l1和 l2 来代表两个非负整数。数字最高位位于链表开始位置。它们的每个节点只存储一位数字。将这两数相加会返回一个新的链表。

可以假设除了数字 0 之外，这两个数字都不会以零开头。

示例1：





输入：l1 = [7,2,4,3], l2 = [5,6,4]
输出：[7,8,0,7]
示例2：

输入：l1 = [2,4,3], l2 = [5,6,4]
输出：[8,0,7]
示例3：

输入：l1 = [0], l2 = [0]
输出：[0]

提示：

链表的长度范围为 [1, 100]
0 <= node.val <= 9
输入数据保证链表代表的数字无前导 0

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        l1list = []
        while (l1):
            l1list.append(l1.val)
            l1 = l1.next

        l2list = []
        while (l2):
            l2list.append(l2.val)
            l2 = l2.next

        llist = []
        carry = 0
        while l1list or l2list or carry!=0:
            tmp = (l1list.pop() if l1list else 0) + (l2list.pop() if l2list else
0) + carry

            if tmp >= 10:
                tmp -= 10
                carry = 1
            else:
                carry = 0
            llist.append(tmp)


        while len(llist) > 1 and llist[-1] == 0:
            llist.pop()

        duymp = ListNode(-1)
        cur = duymp
        while llist:
            node = ListNode(llist.pop())
            cur.next = node
            cur = cur.next
        return duymp.next
```

剑指 Offer II 026. 重排链表

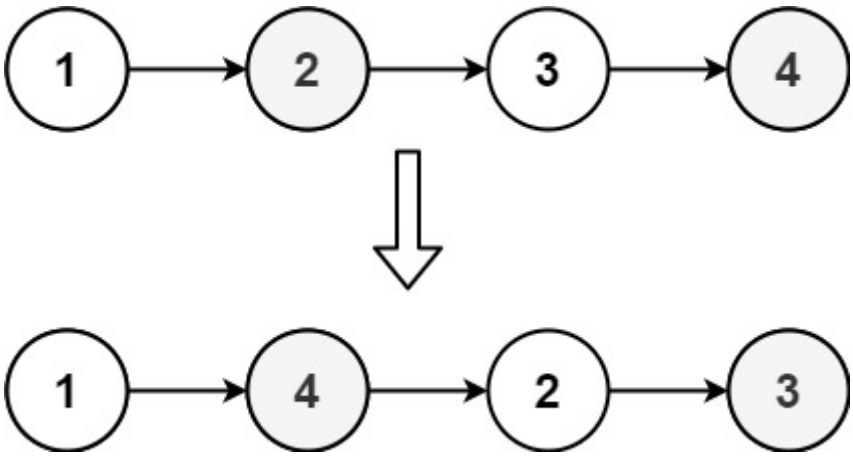
给定一个单链表 L 的头节点 head ，单链表 L 表示为：

 L0 → L1 → … → Ln-1 → Ln
请将其重新排列后变为：

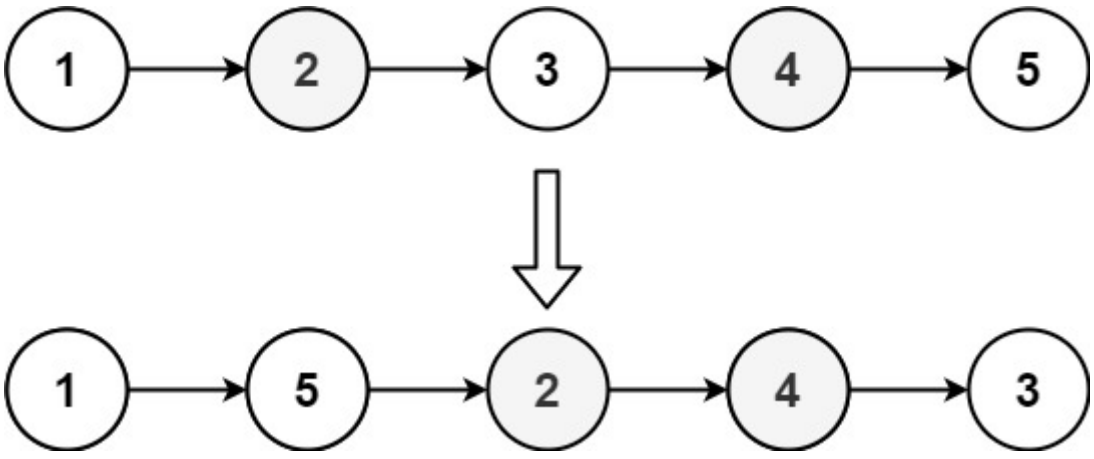
L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → …

不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例 1:



输入: head = [1,2,3,4]
输出: [1,4,2,3]
示例 2:



输入: head = [1,2,3,4,5]
输出: [1,5,2,4,3]

提示:

链表的长度范围为 [1, 5 * 104]
1 <= node.val <= 1000

```
# Definition for singly-linked list.
# class ListNode:
```

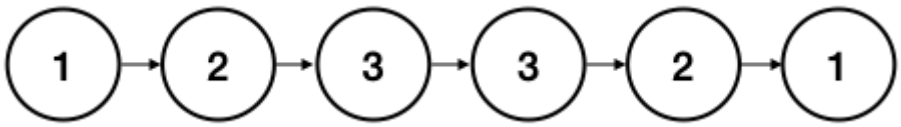
```
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head):
        pre, cur = None, head
        while cur:
            tmp = cur.next
            cur.next = pre
            pre, cur = cur, tmp
        return pre

    def reorderList(self, head: ListNode) -> None:
        pre = ListNode()
        pre.next = head
        slow = fast = pre
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        half = slow.next
        slow.next = None
        rev_half = self.reverseList(half)
        cur = pre.next
        while rev_half:
            tmp = cur.next
            cur.next = rev_half
            cur = cur.next
            rev_half = rev_half.next
            cur.next = tmp
            cur = cur.next
```

剑指 Offer II 027. 回文链表

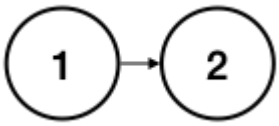
给定一个链表的 头节点 head ，请判断其是否为回文链表。

如果一个链表是回文，那么链表节点序列从前往后看和从后往前看是相同的。

示例 1：



输入: head = [1,2,3,3,2,1]
输出: true
示例 2：




```
输入: head = [1,2]
输出: false


提示:

链表 L 的长度范围为 [1, 105]
0 <= node.val <= 9


进阶:能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题?
```

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        slow = fast = head
        while(fast!=None and fast.next!=None):
            slow = slow.next
            fast = fast.next.next
        if (fast!=None):
            slow= slow.next
        left = head
        right = self.reverse(slow)

        while right!= None:
            if left.val !=right.val:
                return False
            left= left.next
            right = right.next
        return True

    def reverse(self,head):
        pre = None
        cur = head
        while(cur != None):
            tmp = cur.next
            cur.next=pre
            pre=cur
            cur = tmp
        return pre
```

(理解)剑指 Offer II 028. 展平多级双向链表

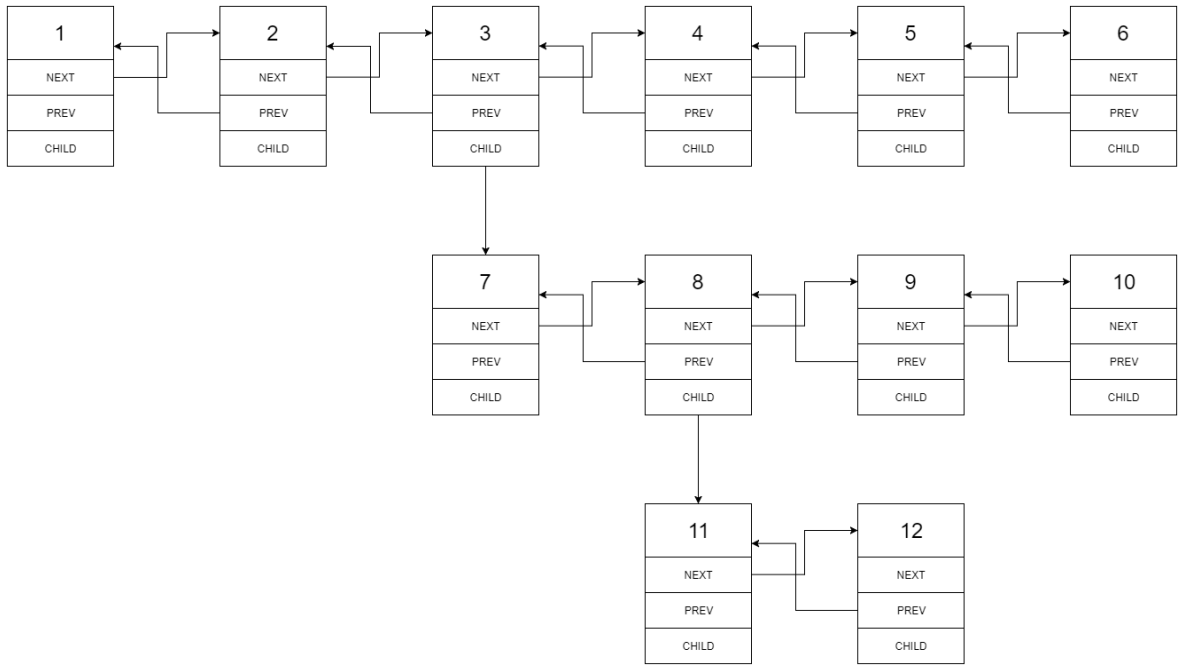
多级双向链表中，除了指向下一个节点和前一个节点指针之外，它还有一个子链表指针，可能指向单独的双向链表。这些子列表也可能会有一个或多个自己的子项，依此类推，生成多级数据结构，如下面的示例所示。

给定位于列表第一级的头节点，请扁平化列表，即将这样的多级双向链表展平成普通的双向链表，使所有结点出现在单级双链表中。

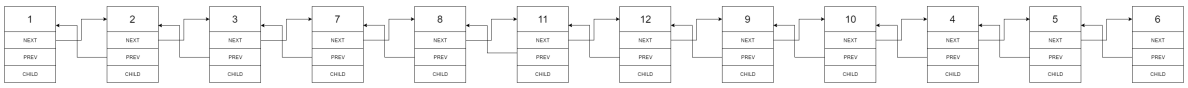
示例 1：

输入：head = [1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
输出：[1,2,3,7,8,11,12,9,10,4,5,6]
解释：

输入的多级列表如下图所示：



扁平化后的链表如下图：



示例 2：

输入：head = [1,2,null,3]
输出：[1,3,2]
解释：

输入的多级列表如下图所示：

```
  1---2---NULL
  |
  3---NULL
```

示例 3：

输入：head = []

输出：[]

如何表示测试用例中的多级链表？

以 示例 1 为例：

```
 1---2---3---4---5---6--NULL
         |
         7---8---9---10--NULL
             |
             11--12--NULL
```

序列化其中的每一级之后：

```
[1,2,3,4,5,6,null]
[7,8,9,10,null]
[11,12,null]
```

为了将每一级都序列化到一起，我们需要每一级中添加值为 null 的元素，以表示没有节点连接到上一级的上级节点。

```
[1,2,3,4,5,6,null]
[null,null,7,8,9,10,null]
[null,11,12,null]
```

合并所有序列化结果，并去除末尾的 null 。

```
[1,2,3,4,5,6,null,null,null,7,8,9,10,null,null,11,12]
```

提示：

节点数目不超过 1000
1 <= Node.val <= 10^5

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val, prev, next, child):
        self.val = val
        self.prev = prev
        self.next = next
        self.child = child
"""

class Solution:
    def flatten(self, head: 'Node') -> 'Node':
        if head == None:
            return head

        dummy = Node(-1, None, None, None)

        def dfs(pre: 'Node', cur: 'Node') -> 'Node':
            if cur == None:
                return pre

            pre.next = cur
```

```
            cur.prev = pre

            nxt_head = cur.next          #相当于4

            tail = dfs(cur, cur.child)  #相当于dfs(3, 7)
            cur.child = None

            return dfs(tail, nxt_head)  #相当于dfs(12, 4)

        dfs(dummy, head)
        dummy.next.prev = None
        return dummy.next
```

剑指 Offer II 029. 排序的循环链表

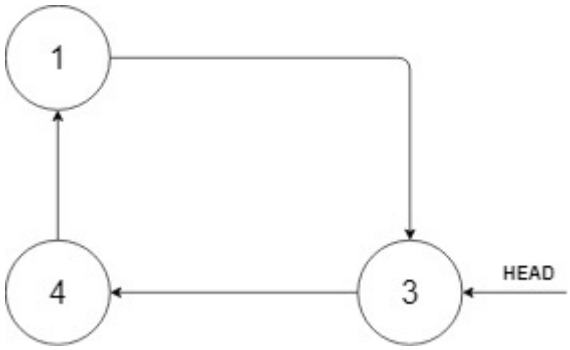
给定循环单调非递减列表中的一个点，写一个函数向这个列表中插入一个新元素 `insertVal` ，使这个列表仍然是循环升序的。

给定的可以是这个列表中任意一个顶点的指针，并不一定是这个列表中最小元素的指针。

如果有多个满足条件的插入位置，可以选择任意一个位置插入新的值，插入后整个列表仍然保持有序。

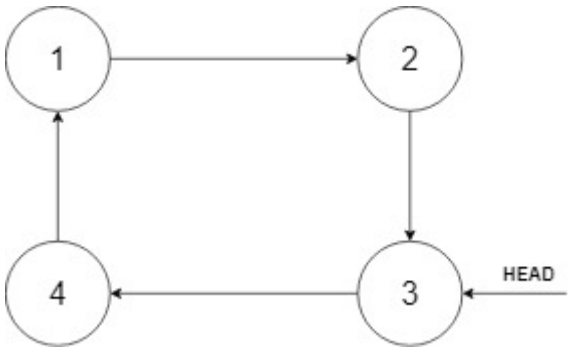
如果列表为空（给定的节点是 `null`），需要创建一个循环有序列表并返回这个节点。否则。请返回原先给定的节点。

示例 1：



输入：`head = [3,4,1], insertVal = 2`
输出：`[3,4,1,2]`
解释：在上图中，有一个包含三个元素的循环有序列表，你获得值为 `3` 的节点的指针，我们需要向表中插入元素 `2` 。新插入的节点应该在 `1` 和 `3` 之间，插入之后，整个列表如上图所示，最后返回节点 `3` 。

示例 2：



输入：head = [], insertVal = 1
输出：[1]
解释：列表为空（给定的节点是 null），创建一个循环有序列表并返回这个节点。
示例 3：

输入：head = [1], insertVal = 0
输出：[1,0]

提示：

0 <= Number of Nodes <= 5 * 10^4
-10^6 <= Node.val <= 10^6
-10^6 <= insertVal <= 10^6

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val=None, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    def insert(self, head: 'Node', insertVal: int) -> 'Node':
        inst = Node(insertVal)
        if not head:
            inst.next = inst
            return inst
        node = head
        while node.next != head:
            if node.next.val >= insertVal >= node.val:
                break
            if node.next.val < node.val and (node.val <= insertVal or insertVal
<= node.next.val):
                break
            node = node.next
        inst.next = node.next
        node.next = inst
        return head
```

(理解)剑指 Offer II 030. 插入、删除和随机访问都是 O(1) 的容器

设计一个支持在平均 时间复杂度 O(1) 下，执行以下操作的数据结构：

insert(val)：当元素 val 不存在时返回 true ，并向集合中插入该项，否则返回 false 。
remove(val)：当元素 val 存在时返回 true ，并从集合中移除该项，否则返回 false 。
getRandom：随机返回现有集合中的一项。每个元素应该有 相同的概率 被返回。

示例 ：

输入: inputs = ["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"]
[[], [1], [2], [2], [], [1], [2], []]
输出: [null, true, false, true, 2, true, false, 2]
解释:
RandomizedSet randomSet = new RandomizedSet(); // 初始化一个空的集合
randomSet.insert(1); // 向集合中插入 1 ， 返回 true 表示 1 被成功地插入

randomSet.remove(2); // 返回 false，表示集合中不存在 2

randomSet.insert(2); // 向集合中插入 2 返回 true ，集合现在包含 [1,2]

randomSet.getRandom(); // getRandom 应随机返回 1 或 2

randomSet.remove(1); // 从集合中移除 1 返回 true 。集合现在包含 [2]

randomSet.insert(2); // 2 已在集合中，所以返回 false

randomSet.getRandom(); // 由于 2 是集合中唯一的数字，getRandom 总是返回 2

提示：

-231 <= val <= 231 - 1
最多进行 2 * 105 次 insert ， remove 和 getRandom 方法调用
当调用 getRandom 方法时，集合中至少有一个元素

```python
import random


class RandomizedSet:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.lst = []
        self.dct = dict()

    def insert(self, val: int) -> bool:
        """
```

```
        Inserts a value to the set. Returns true if the set did not already
contain the specified element.
        """
        if val in self.dct:
            return False
        self.lst.append(val)
        self.dct[val] = len(self.lst) - 1
        return True

    def remove(self, val: int) -> bool:
        """
        Removes a value from the set. Returns true if the set contained the
specified element.
        """
        if val not in self.dct:
            return False
        i = self.dct[val]
        self.lst[i] = self.lst[-1]
        self.dct[self.lst[i]] = i
        self.lst.pop(-1)
        self.dct.pop(val)
        return True

    def getRandom(self) -> int:
        """
        Get a random element from the set.
        """
        return self.lst[random.randint(0, len(self.lst) - 1)]


# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()
```

(理解)剑指 Offer II 031. 最近最少使用缓存

运用所掌握的数据结构，设计和实现一个 LRU (Least Recently Used，最近最少使用) 缓存机制 。

实现 LRUCache 类：

LRUCache(int capacity) 以正整数作为容量 capacity 初始化 LRU 缓存
int get(int key) 如果关键字 key 存在于缓存中，则返回关键字的值，否则返回 -1 。
void put(int key, int value) 如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字-值」。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。

示例：

输入
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]

输出
[null, null, null, 1, null, -1, null, -1, 3, 4]

解释
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // 缓存是 {1=1}
lRUCache.put(2, 2); // 缓存是 {1=1, 2=2}
lRUCache.get(1);    // 返回 1
lRUCache.put(3, 3); // 该操作会使得关键字 2 作废，缓存是 {1=1, 3=3}
lRUCache.get(2);    // 返回 -1 (未找到)
lRUCache.put(4, 4); // 该操作会使得关键字 1 作废，缓存是 {4=4, 3=3}
lRUCache.get(1);    // 返回 -1 (未找到)
lRUCache.get(3);    // 返回 3
lRUCache.get(4);    // 返回 4
```

提示：

```
1 <= capacity <= 3000
0 <= key <= 10000
0 <= value <= 105
最多调用 2 * 105 次 get 和 put
```

进阶：是否可以在 O(1) 时间复杂度内完成这两种操作？

```python
class LRUCache:

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.dic = {}
        self.key_quene = []

    def get(self, key: int) -> int:
        if key in self.dic:
            self.key_quene.remove(key)
            self.key_quene.append(key)
        return self.dic.get(key, -1)

    def put(self, key: int, value: int) -> None:
        if key in self.dic:
            self.key_quene.remove(key)
            self.key_quene.append(key)
        else:
            self.key_quene.append(key)
        self.dic[key] = value
        if len(self.dic)>self.capacity:
            delate = self.key_quene.pop(0)
            del self.dic[delate]




# Your LRUCache object will be instantiated and called as such:
```

```
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

剑指 Offer II 032. 有效的变位词

给定两个字符串 s 和 t ，编写一个函数来判断它们是不是一组变位词（字母异位词）。

注意：若 s 和 t 中每个字符出现的次数都相同且字符顺序不完全相同，则称 s 和 t 互为变位词（字母异位词）。

示例 1:

输入: s = "anagram", t = "nagaram"
输出: true
示例 2:

输入: s = "rat", t = "car"
输出: false
示例 3:

输入: s = "a", t = "a"
输出: false

提示:

1 <= s.length, t.length <= 5 * 104
s and t 仅包含小写字母

进阶: 如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        return s != t and Counter(s) == Counter(t)
```

剑指 Offer II 033. 变位词组

给定一个字符串数组 strs ，将 变位词 组合在一起。 可以按任意顺序返回结果列表。

注意：若两个字符串中每个字符出现的次数都相同，则称它们互为变位词。

示例 1:

输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]
输出: [["bat"],["nat","tan"],["ate","eat","tea"]]
示例 2:

输入: strs = [""]
输出: [[""]]
示例 3:

输入: strs = ["a"]
输出: [["a"]]

提示:

1 <= strs.length <= 104
0 <= strs[i].length <= 100
strs[i] 仅包含小写字母

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        dct = defaultdict(list)
        for word in strs:
            lst = list(word)
            lst.sort()
            dct[tuple(lst)].append(word)
        return [dct[tup] for tup in dct]
```

剑指 Offer II 034. 外星语言是否排序

某种外星语也使用英文小写字母，但可能顺序 order 不同。字母表的顺序（order）是一些小写字母的排列。

给定一组用外星语书写的单词 words，以及其字母表的顺序 order，只有当给定的单词在这种外星语中按字典序排列时，返回 true；否则，返回 false。

示例 1：

输入：words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"
输出：true
解释：在该语言的字母表中，'h' 位于 'l' 之前，所以单词序列是按字典序排列的。
示例 2：

输入：words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"
输出：false
解释：在该语言的字母表中，'d' 位于 'l' 之后，那么 words[0] > words[1]，因此单词序列不是按字典序排列的。
示例 3：

输入：words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"
输出：false
解释：当前三个字符 "app" 匹配时，第二个字符串相对短一些，然后根据词典编纂规则 "apple" > "app"，因为 'l' > '∅'，其中 '∅' 是空白字符，定义为比任何其他字符都小（更多信息）。

提示：

1 <= words.length <= 100
1 <= words[i].length <= 20
order.length == 26
在 words[i] 和 order 中的所有字符都是英文小写字母。

```python
class Solution:
    def isAlienSorted(self, words: List[str], order: str) -> bool:
        ind = {w: i for i, w in enumerate(order)}
        lst = [[ind[w] for w in word] for word in words]
        n = len(lst)
        for i in range(n - 1):
            if lst[i] > lst[i + 1]:
                return False
        return True
```

剑指 Offer II 035. 最小时间差

给定一个 24 小时制（小时:分钟 "HH:MM"）的时间列表，找出列表中任意两个时间的最小时间差并以分钟数表示。

示例 1：

输入：timePoints = ["23:59","00:00"]
输出：1
示例 2：

输入：timePoints = ["00:00","23:59","00:00"]
输出：0

提示：

2 <= timePoints <= 2 * 104
timePoints[i] 格式为 "HH:MM"

```python
class Solution:
    def findMinDifference(self, timePoints: List[str]) -> int:

        def check(tm):
            hour, minute = tm.split(":")
            return int(hour) * 60 + int(minute)

        lst = sorted([check(time) for time in timePoints])
        n = len(lst)
        ans = lst[0] + 24 * 60 - lst[n - 1]
        for i in range(n-1):
            if lst[i + 1] - lst[i] < ans:
                ans = lst[i + 1] - lst[i]
```

```
        return ans
```

注意：

```
ans = timelst[0] + 24 * 60 - timelst[n - 1]
```

剑指 Offer II 036. 后缀表达式

根据 逆波兰表示法，求该后缀表达式的计算结果。

有效的算符包括 +、-、*、/ 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。

说明：

整数除法只保留整数部分。
给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。

示例 1：

输入：tokens = ["2","1","+","3","*"]
输出：9
解释：该算式转化为常见的中缀算术表达式为：((2 + 1) * 3) = 9
示例 2：

输入：tokens = ["4","13","5","/","+"]
输出：6
解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6
示例 3：

输入：tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
输出：22
解释：
该算式转化为常见的中缀算术表达式为：
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22

提示：

1 <= tokens.length <= 104
tokens[i] 要么是一个算符（"+"、"-"、"*" 或 "/"），要么是一个在范围 [-200, 200] 内的整数

逆波兰表达式：

逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。

平常使用的算式则是一种中缀表达式，如 (1 + 2) * (3 + 4) 。
该算式的逆波兰表达式写法为 ((1 2 +) (3 4 +) *) 。
逆波兰表达式主要有以下两个优点：

去掉括号后表达式无歧义，上式即便写成 1 2 + 3 4 + * 也可以依据次序计算出正确结果。
适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。

```python
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        for token in tokens:
            if token not in '+-*/':
                stack.append(int(token))
            else:
                b = stack.pop()
                a = stack.pop()

                if token == '+':
                    stack.append(a + b)
                elif token == '-':
                    stack.append(a - b)
                elif token == '*':
                    stack.append(a * b)
                else:
                    stack.append(int(a/b))

        return stack[0]
```

(理解)剑指 Offer II 037. 小行星碰撞

给定一个整数数组 asteroids，表示在同一行的小行星。

对于数组中的每一个元素，其绝对值表示小行星的大小，正负表示小行星的移动方向（正表示向右移动，负表示向左移动）。每一颗小行星以相同的速度移动。

找出碰撞后剩下的所有小行星。碰撞规则：两个行星相互碰撞，较小的行星会爆炸。如果两颗行星大小相同，则两颗行星都会爆炸。两颗移动方向相同的行星，永远不会发生碰撞。

示例 1：

输入：asteroids = [5,10,-5]
输出：[5,10]
解释：10 和 -5 碰撞后只剩下 10 。 5 和 10 永远不会发生碰撞。
示例 2：

输入：asteroids = [8,-8]
输出：[]
解释：8 和 -8 碰撞后，两者都发生爆炸。
示例 3：

输入：asteroids = [10,2,-5]
输出：[10]
解释：2 和 -5 发生碰撞后剩下 -5 。10 和 -5 发生碰撞后剩下 10 。
示例 4：

输入：asteroids = [-2,-1,1,2]
输出：[-2,-1,1,2]
解释：-2 和 -1 向左移动，而 1 和 2 向右移动。 由于移动方向相同的行星不会发生碰撞，所以最终没有行星发生碰撞。

提示：

2 <= asteroids.length <= 104
-1000 <= asteroids[i] <= 1000
asteroids[i] != 0

```python
class Solution:
    def asteroidCollision(self, asteroids: List[int]) -> List[int]:
        stack = []
        for a in asteroids:
            while stack and 0 < stack[-1] < a * -1:      # 只有栈顶炸
                stack.pop()
            if not stack or stack[-1] < 0 or a > 0:      # 不发生爆炸
                stack.append(a)
            elif a * -1 == stack[-1]:                    # 同时爆炸
                stack.pop()
        return stack
```

(理解)剑指 Offer II 038. 每日温度

请根据每日 气温 列表 temperatures ，重新生成一个列表，要求其对应位置的输出为：要想观测到更高的气温，至少需要等待的天数。如果气温在这之后都不会升高，请在该位置用 0 来代替。

示例 1:

输入: temperatures = [73,74,75,71,69,72,76,73]
输出: [1,1,4,2,1,1,0,0]
示例 2:

输入: temperatures = [30,40,50,60]
输出: [1,1,1,0]
示例 3:

输入: temperatures = [30,60,90]
输出: [1,1,0]

提示：

```
1 <= temperatures.length <= 105
30 <= temperatures[i] <= 100
```

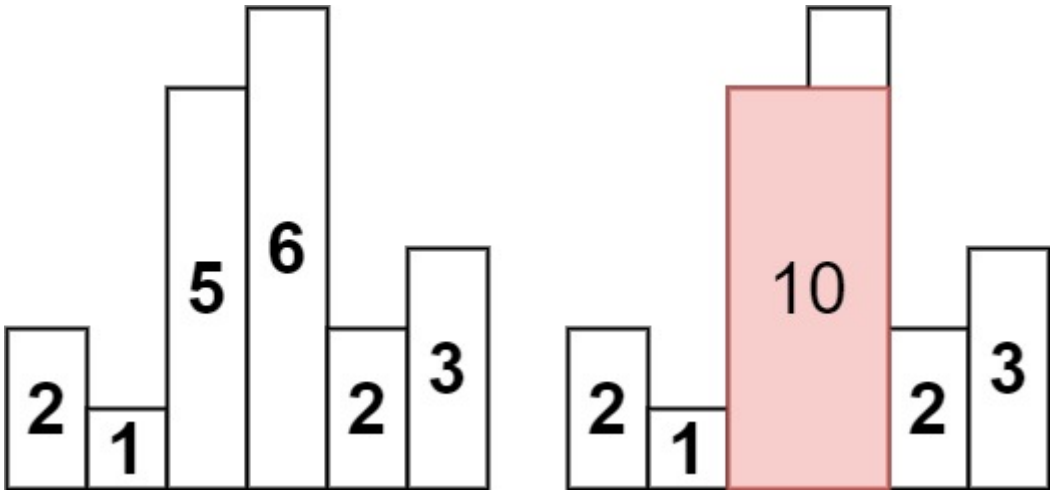
```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        n = len(temperatures)
        ans = [0] * n
        stack = []
        for i in range(n):
            while stack and temperatures[stack[-1]] < temperatures[i]:
                j = stack.pop()
                ans[j] = i - j
            stack.append(i)
        return ans
```

(困难)剑指 Offer II 039. 直方图最大矩形面积

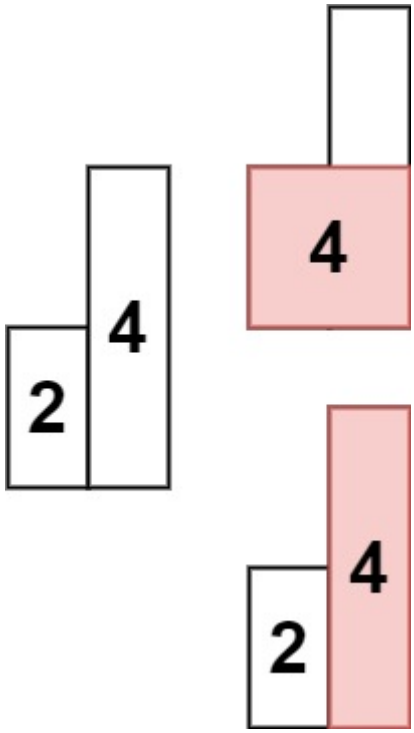
给定非负整数数组 heights ，数组中的数字用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

求在该柱状图中，能够勾勒出来的矩形的最大面积。

示例 1:



输入：heights = [2,1,5,6,2,3]
输出：10
解释：最大的矩形为图中红色区域，面积为 10
示例 2：



输入： heights = [2,4]
输出： 4

提示：

1 <= heights.length <=105
0 <= heights[i] <= 104

```
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:

        n = len(heights)

        post = [n - 1] * n
        stack = []
        for i in range(n):
            while stack and heights[stack[-1]] > heights[i]:
                post[stack.pop(-1)] = i - 1
            stack.append(i)

        pre = [0] * n
        for i in range(n - 1, -1, -1):
            while stack and heights[stack[-1]] > heights[i]:
                pre[stack.pop(-1)] = i + 1
            stack.append(i)

        ans = 0
        for i in range(n):
            cur = heights[i] * (post[i] - pre[i] + 1)
            if cur > ans:
                ans = cur
```

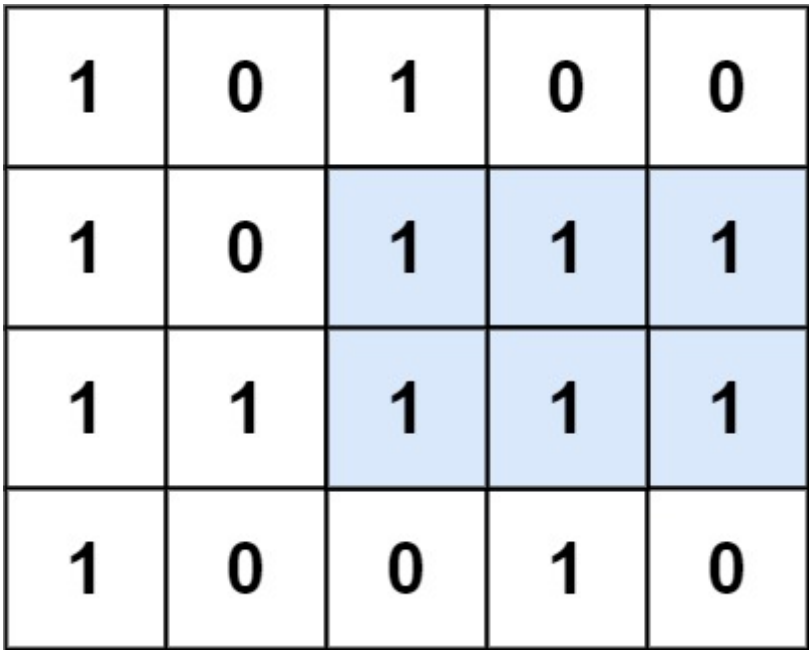
```
        return ans
```

(困难) 剑指 Offer II 040. 矩阵中最大的矩形

给定一个由 0 和 1 组成的矩阵 matrix ，找出只包含 1 的最大矩形，并返回其面积。

注意：此题 matrix 输入格式为一维 01 字符串数组。

示例 1：



输入：matrix = ["10100","10111","11111","10010"]
输出：6
解释：最大矩形如上图所示。
示例 2：

输入：matrix = []
输出：0
示例 3：

输入：matrix = ["0"]
输出：0
示例 4：

输入：matrix = ["1"]
输出：1
示例 5：

输入：matrix = ["00"]
输出：0

提示：

```
rows == matrix.length
cols == matrix[0].length
0 <= row, cols <= 200
matrix[i][j] 为 '0' 或 '1'
```

```python
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        n = len(heights)

        post = [n - 1] * n
        stack = []
        for i in range(n):
            while stack and heights[stack[-1]] > heights[i]:
                post[stack.pop(-1)] = i - 1
            stack.append(i)

        pre = [0] * n
        for i in range(n - 1, -1, -1):
            while stack and heights[stack[-1]] > heights[i]:
                pre[stack.pop(-1)] = i + 1
            stack.append(i)

        ans = 0
        for i in range(n):
            cur = heights[i] * (post[i] - pre[i] + 1)
            if cur > ans:
                ans = cur
        return ans

    def maximalRectangle(self, matrix: List[str]) -> int:
        if not matrix or not matrix[0]:
            return 0
        m, n = len(matrix), len(matrix[0])
        ans = 0
        cur = [0] * n
        for i in range(m):
            for j in range(n):
                if matrix[i][j] == '1':
                    cur[j] += 1
                else:
                    cur[j] = 0
            area = self.largestRectangleArea(cur)
            if area > ans:
                ans = area
        return ans
```

剑指 Offer II 041. 滑动窗口的平均值

给定一个整数数据流和一个窗口大小，根据该滑动窗口的大小，计算滑动窗口里所有数字的平均值。

实现 MovingAverage 类：

MovingAverage(int size) 用窗口大小 size 初始化对象。

double next(int val) 成员函数 next 每次调用的时候都会往滑动窗口增加一个整数，请计算并返回数据流中最后 size 个值的移动平均值，即滑动窗口里所有数字的平均值。

示例：

输入：
inputs = ["MovingAverage", "next", "next", "next", "next"]
inputs = [[3], [1], [10], [3], [5]]
输出：
[null, 1.0, 5.5, 4.66667, 6.0]

解释：
MovingAverage movingAverage = new MovingAverage(3);
movingAverage.next(1); // 返回 1.0 = 1 / 1
movingAverage.next(10); // 返回 5.5 = (1 + 10) / 2
movingAverage.next(3); // 返回 4.66667 = (1 + 10 + 3) / 3
movingAverage.next(5); // 返回 6.0 = (10 + 3 + 5) / 3

提示：

1 <= size <= 1000
-105 <= val <= 105
最多调用 next 方法 104 次

```python
class MovingAverage:

    def __init__(self, size: int):
        """
        Initialize your data structure here.
        """
        self.size=size
        self.window = []


    def next(self, val: int) -> float:
        if len(self.window)>=self.size:
            self.window.pop(0)
            self.window.append(val)
        else:
            self.window.append(val)
        res = sum(self.window)/len(self.window)
        return res
```

剑指 Offer II 042. 最近请求次数

写一个 RecentCounter 类来计算特定时间范围内最近的请求。

请实现 RecentCounter 类：

RecentCounter() 初始化计数器，请求数为 0 。

int ping(int t) 在时间 t 添加一个新请求，其中 t 表示以毫秒为单位的某个时间，并返回过去 3000 毫秒内发生的所有请求数（包括新请求）。确切地说，返回在 [t-3000, t] 内发生的请求数。
保证 每次对 ping 的调用都使用比之前更大的 t 值。

示例：

输入：
inputs = ["RecentCounter", "ping", "ping", "ping", "ping"]
inputs = [[], [1], [100], [3001], [3002]]
输出：
[null, 1, 2, 3, 3]

解释：
RecentCounter recentCounter = new RecentCounter();
recentCounter.ping(1);   // requests = [1]，范围是 [-2999,1]，返回 1
recentCounter.ping(100);   // requests = [1, 100]，范围是 [-2900,100]，返回 2
recentCounter.ping(3001); // requests = [1, 100, 3001]，范围是 [1,3001]，返回 3
recentCounter.ping(3002); // requests = [1, 100, 3001, 3002]，范围是 [2,3002]，返回 3

提示：

1 <= t <= 109
保证每次对 ping 调用所使用的 t 值都 严格递增
至多调用 ping 方法 104 次

```python
class RecentCounter:

    def __init__(self):
        self.dic = []


    def ping(self, t: int) -> int:
        self.dic.append(t)
        self.dic = [i for i in self.dic if t - i <= 3000]
        return len(self.dic)
```

(理解)剑指 Offer II 043. 往完全二叉树添加节点

完全二叉树是每一层（除最后一层外）都是完全填充（即，节点数达到最大，第 n 层有 2n-1 个节点）的，并且所有的节点都尽可能地集中在左侧。

设计一个用完全二叉树初始化的数据结构 CBTInserter，它支持以下几种操作：

CBTInserter(TreeNode root) 使用根节点为 root 的给定树初始化该数据结构；
CBTInserter.insert(int v) 向树中插入一个新节点，节点类型为 TreeNode，值为 v 。使树保持完全二叉树的状态，并返回插入的新节点的父节点的值；

CBTInserter.get_root() 将返回树的根节点。

示例 1：

输入：inputs = ["CBTInserter","insert","get_root"], inputs = [[[1]],[2],[]]
输出：[null,1,[1,2]]
示例 2：

输入：inputs = ["CBTInserter","insert","insert","get_root"], inputs =
[[[1,2,3,4,5,6]],[7],[8],[]]
输出：[null,3,4,[1,2,3,4,5,6,7,8]]

提示：

最初给定的树是完全二叉树，且包含 1 到 1000 个节点。
每个测试用例最多调用 CBTInserter.insert 操作 10000 次。
给定节点或插入节点的每个值都在 0 到 5000 之间。

注意： bfs

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class CBTInserter:

    def __init__(self, root: TreeNode):
        self.root = root


    def insert(self, v: int) -> int:
        st = []

        if self.root:
            st.append(self.root)

        while st:
            l = len(st)
            for i in range(l):
                node = st.pop(0)
                if node.left:
                    st.append(node.left)
                else:
                    node.left = TreeNode(v)
                    return node.val

                if node.right:
                    st.append(node.right)
                else:
                    node.right = TreeNode(v)
                    return node.val
```

```
    def get_root(self) -> TreeNode:
        return self.root


# Your CBTInserter object will be instantiated and called as such:
# obj = CBTInserter(root)
# param_1 = obj.insert(v)
# param_2 = obj.get_root()
```

剑指 Offer II 044. 二叉树每层的最大值

给定一棵二叉树的根节点 root ，请找出该二叉树中每一层的最大值。

示例1：

```
输入: root = [1,3,2,5,3,null,9]
输出: [1,3,9]
解释:
          1
         / \
        3   2
       / \   \
      5   3   9
```

示例2：

```
输入: root = [1,2,3]
输出: [1,3]
解释:
          1
         / \
        2   3
```

示例3：

```
输入: root = [1]
输出: [1]
```

示例4：

```
输入: root = [1,null,2]
输出: [1,2]
解释:
           1
            \
             2
```

示例5：

```
输入: root = []
输出: []
```

提示：

二叉树的节点个数的范围是 [0,104]
-231 <= Node.val <= 231 - 1

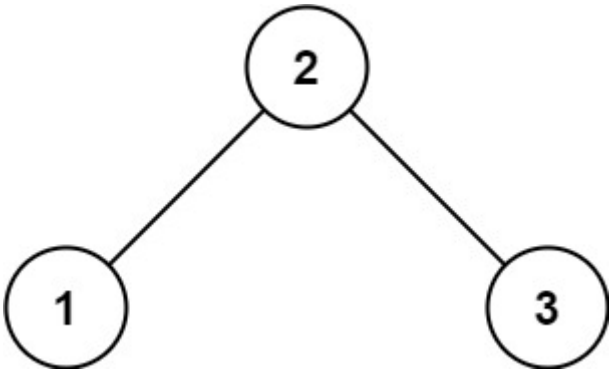
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def largestValues(self, root: TreeNode) -> List[int]:
        res = []
        if root == None: return []
        queue = [root]
        while(queue):
            tmp = []
            Restmp = []
            for i in range(len(queue)):
                node = queue[i]
                Restmp.append(node.val)
                if node.left != None:
                    tmp.append(node.left)
                if node.right != None:
                    tmp.append(node.right)
            res.append(max(Restmp))
            queue = tmp
        return res
```

剑指 Offer II 045. 二叉树最底层最左边的值

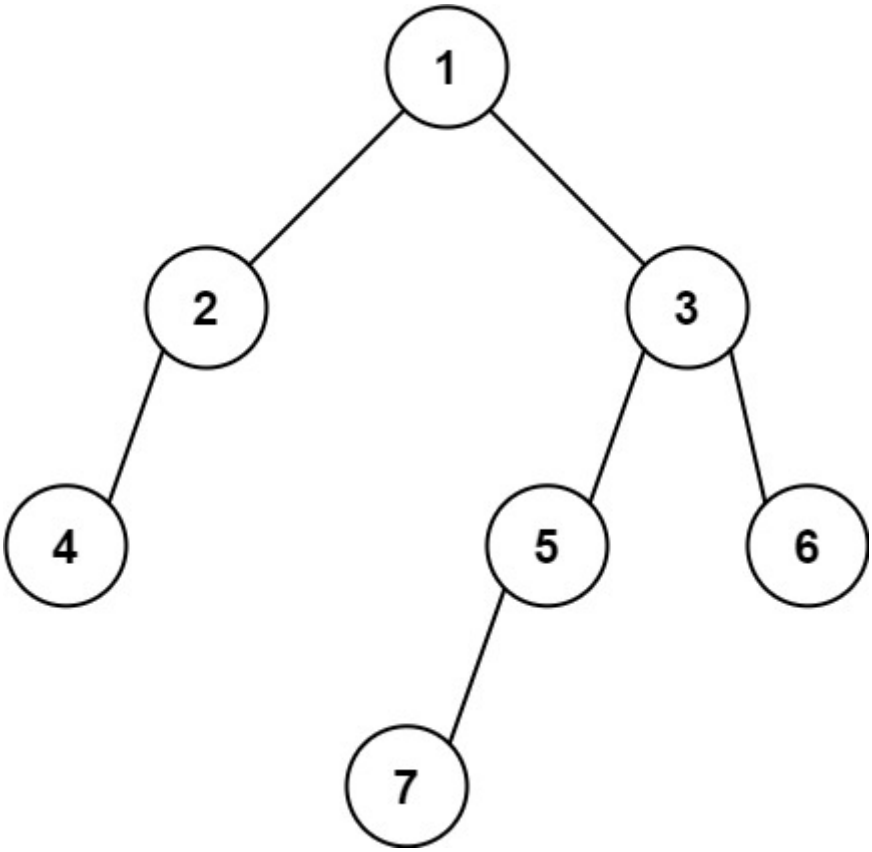
给定一个二叉树的 根节点 root，请找出该二叉树的 最底层 最左边 节点的值。

假设二叉树中至少有一个节点。

示例 1:



输入: root = [2,1,3]
输出: 1
示例 2:



输入: [1,2,3,4,null,5,6,null,null,7]
输出: 7

提示:

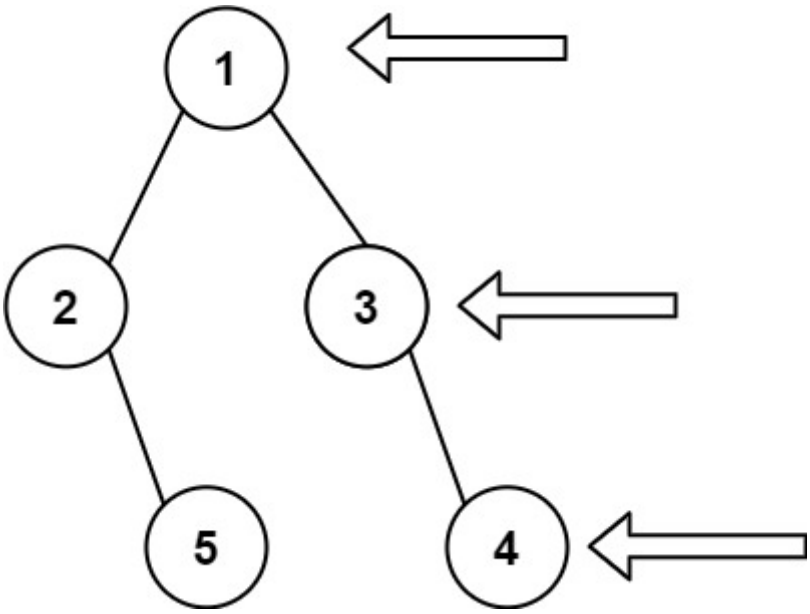
二叉树的节点个数的范围是 [1,104]
-231 <= Node.val <= 231 - 1

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def findBottomLeftValue(self, root: TreeNode) -> int:
        res = 0
        if root == None: return []
        queue = [root]
        while(queue):
            tmp = []
            res = queue[0].val
            for i in range(len(queue)):
                node = queue[i]
                if node.left != None:
                    tmp.append(node.left)
                if node.right != None:
                    tmp.append(node.right)
            queue = tmp
        return res
```

剑指 Offer II 046. 二叉树的右侧视图

给定一个二叉树的 根节点 root，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例 1:



输入: [1,2,3,null,5,null,4]
输出: [1,3,4]
示例 2:

输入: [1,null,3]
输出: [1,3]
示例 3:

输入: []
输出: []

提示:

二叉树的节点个数的范围是 [0,100]
-100 <= Node.val <= 100

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def rightSideView(self, root: TreeNode) -> List[int]:
        res = []
        if root == None: return []
```

```
        queue = [root]
        while(queue):
            tmp = []
            res.append(queue[-1].val)
            for i in range(len(queue)):
                node = queue[i]
                if node.left != None:
                    tmp.append(node.left)
                if node.right != None:
                    tmp.append(node.right)
            queue = tmp
        return res
```

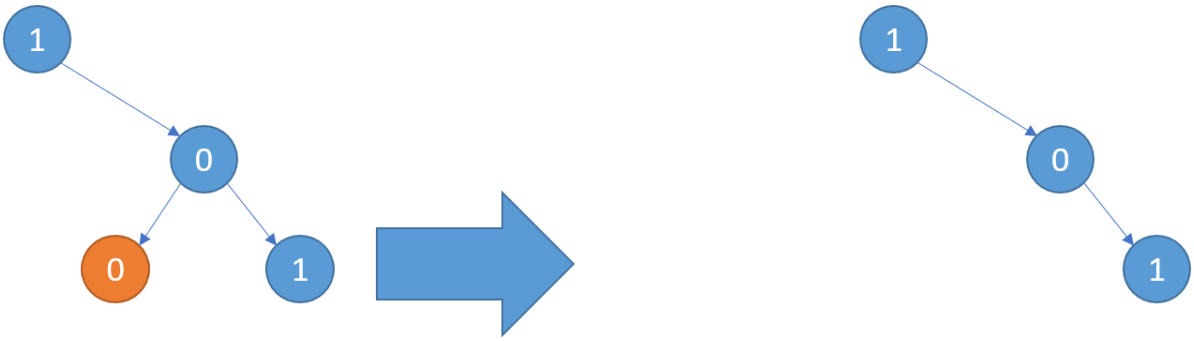
剑指 Offer II 047. 二叉树剪枝

给定一个二叉树 根节点 root ，树的每个节点的值要么是 0，要么是 1。请剪除该二叉树中所有节点的值为 0 的子树。

节点 node 的子树为 node 本身，以及所有 node 的后代。

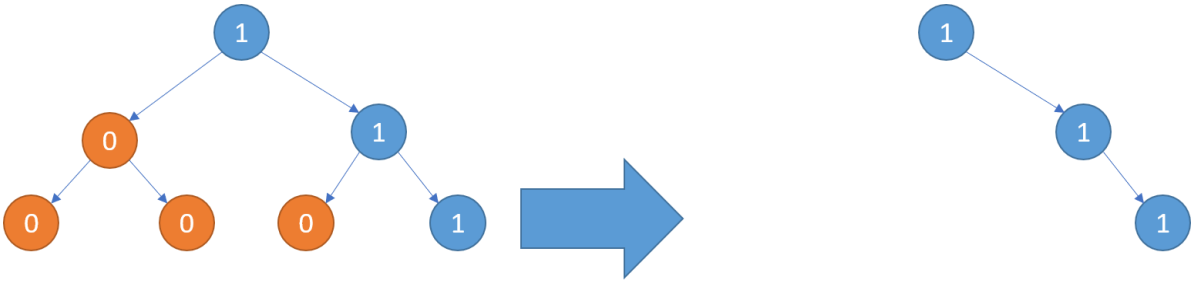
示例 1:

输入: [1,null,0,0,1]
输出: [1,null,0,null,1]
解释:
只有红色节点满足条件“所有不包含 1 的子树”。
右图为返回的答案。



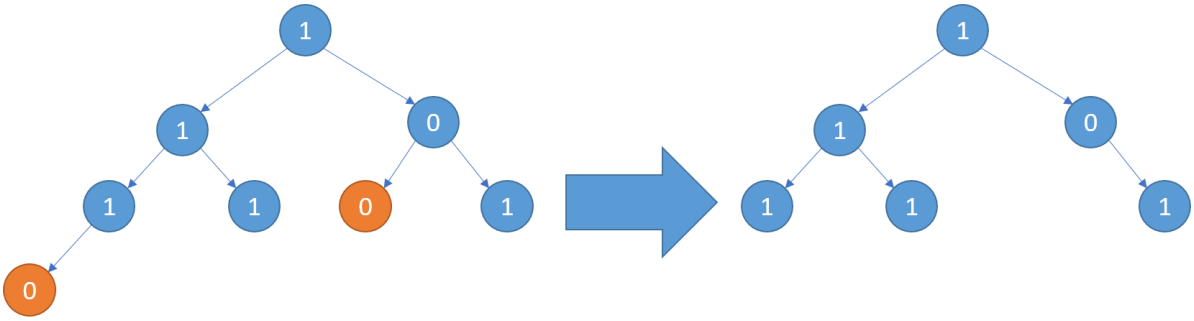
示例 2:

输入: [1,0,1,0,0,0,1]
输出: [1,null,1,null,1]
解释:



示例 3:

输入: [1,1,0,1,1,0,1,0]
输出: [1,1,0,1,1,null,1]
解释:



提示:

二叉树的节点个数的范围是 [1,200]
二叉树节点的值只会是 0 或 1

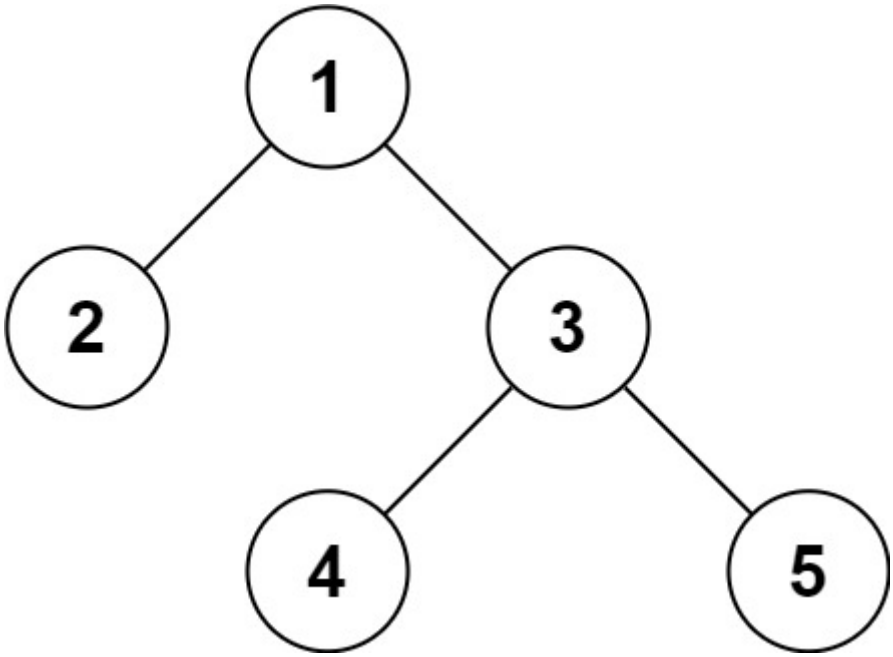
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def pruneTree(self, root: TreeNode) -> TreeNode:
        if root==None: return None

        root.left = self.pruneTree(root.left)
        root.right = self.pruneTree(root.right)

        if root.val==0 and root.left==None and root.right == None:
            return None
        return root
```

剑指 Offer II 048. 序列化与反序列化二叉树

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

示例 1:



输入：root = [1,2,3,null,null,4,5]
输出：[1,2,3,null,null,4,5]
示例 2：

输入：root = []
输出：[]
示例 3：

输入：root = [1]
输出：[1]
示例 4：

输入：root = [1,2]
输出：[1,2]

提示：

输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，也可以采用其他的方法解决这个问题。
树中结点数在范围 [0, 104] 内
-1000 <= Node.val <= 1000

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Codec:
    def __init__(self):
        self.res = []
        self.i = 0
    def serialize(self, root):
        def DFS(root):
```

```
            # 先序遍历：先处理当前节点
            # 如果当前节点为空，记录为字符‘N’
            if not root:
                self.res.append('N')
                return
            # 如果当前节点不为空，记录当前节点值
            else:
                self.res.append(str(root.val))
            # 依次处理左右孩子
            DFS(root.left)
            DFS(root.right)
        DFS(root)
        # 用逗号作为分隔符
        return ','.join(self.res)

    def deserialize(self, data):
        data = data.split(',')
        def DFS():
            # 先序遍历：先处理当前节点
            # 当前节点为空：指针后移，返回空节点
            if data[self.i] == 'N':
                self.i += 1
                return
            # 当前节点不为空：构建当前节点，指针后移
            node = TreeNode(int(data[self.i]))
            self.i += 1
            # 依次处理左右孩子
            node.left = DFS()
            node.right = DFS()
            # 返回根节点
            return node
        return DFS()



# Your Codec object will be instantiated and called as such:
# ser = Codec()
# deser = Codec()
# ans = deser.deserialize(ser.serialize(root))
```

剑指 Offer II 049. 从根节点到叶节点的路径数字之和

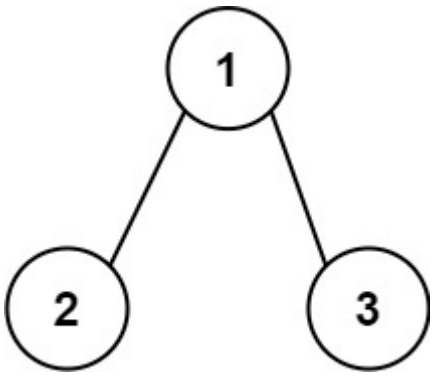
给定一个二叉树的根节点 root ，树中每个节点都存放有一个 0 到 9 之间的数字。

每条从根节点到叶节点的路径都代表一个数字：

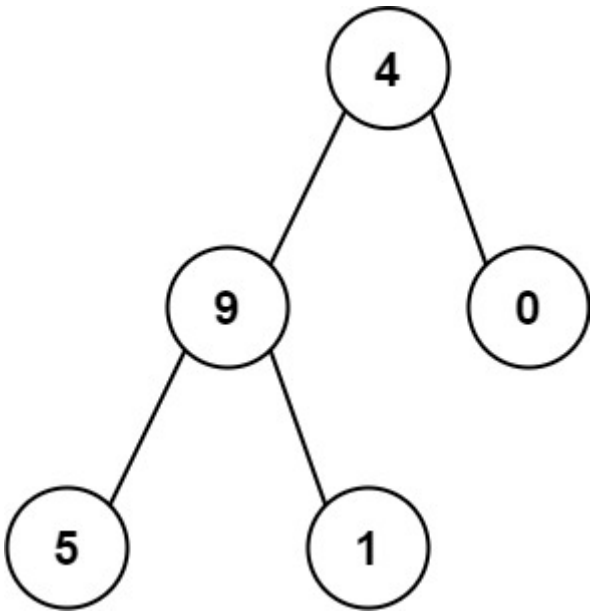
例如，从根节点到叶节点的路径 1 -> 2 -> 3 表示数字 123 。
计算从根节点到叶节点生成的 所有数字之和 。

叶节点 是指没有子节点的节点。

示例 1：



输入：root = [1,2,3]
输出：25
解释：
从根到叶子节点路径 1->2 代表数字 12
从根到叶子节点路径 1->3 代表数字 13
因此，数字总和 = 12 + 13 = 25
示例 2：



输入：root = [4,9,0,5,1]
输出：1026
解释：
从根到叶子节点路径 4->9->5 代表数字 495
从根到叶子节点路径 4->9->1 代表数字 491

从根到叶子节点路径 4->0 代表数字 40
因此，数字总和 = 495 + 491 + 40 = 1026

提示：

树中节点的数目在范围 [1, 1000] 内
0 <= Node.val <= 9
树的深度不超过 10

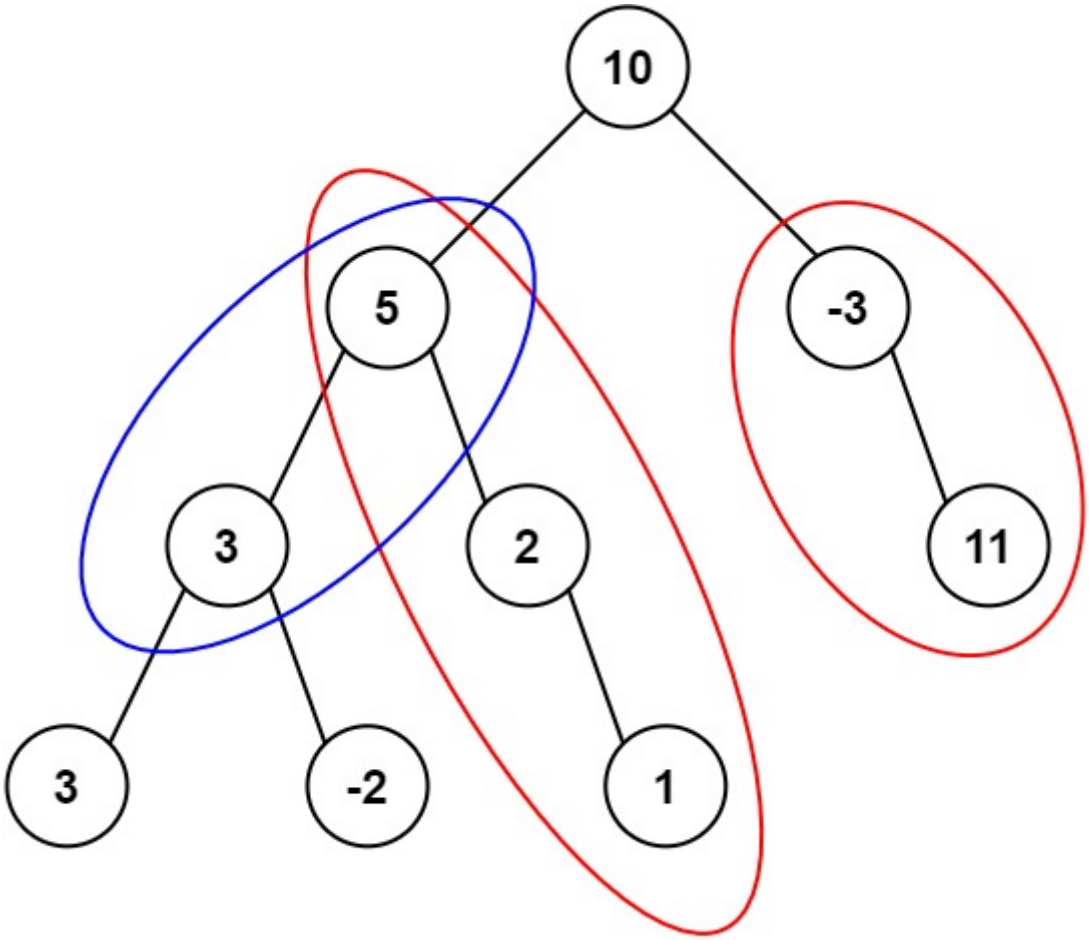
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sumNumbers(self, root: TreeNode) -> int:

        return self.dfs(root,0)

    def dfs(self,root,presum):
        if root==None:
            return 0
        sumtree = presum*10 +root.val

        if root.left == None and root.right == None:
            return sumtree

        return self.dfs(root.left,sumtree) +self.dfs(root.right,sumtree)
```

(理解)剑指 Offer II 050. 向下的路径节点之和

给定一个二叉树的根节点 root ，和一个整数 targetSum ，求该二叉树里节点值之和等于 targetSum 的 路径 的数目。

路径 不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。

示例 1：



输入：root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
输出：3
解释：和等于 8 的路径有 3 条，如图所示。
示例 2：

输入：root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
输出：3

提示：

二叉树的节点个数的范围是 [0,1000]
-109 <= Node.val <= 109
-1000 <= targetSum <= 1000

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def pathSum(self, root: TreeNode, targetSum: int) -> int:
        if root == None: return 0

        self.res = self.rootSum(root,targetSum)
        self.res += self.pathSum(root.left,targetSum)
```

```
        self.res += self.pathSum(root.right,targetSum)

        return self.res

    def rootSum(self,root,targetSum):
        if root == None:
            return 0
        self.res = 0
        if root.val == targetSum:
            self.res +=1
        self.res += self.rootSum(root.left,targetSum-root.val)
        self.res += self.rootSum(root.right,targetSum-root.val)
        return self.res
```

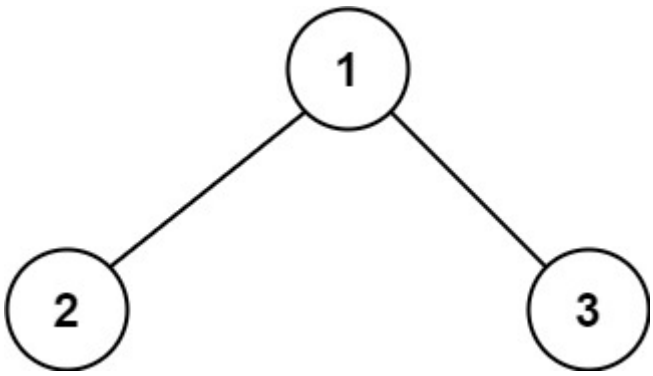
(困难)剑指 Offer II 051. 节点之和最大的路径

路径 被定义为一条从树中任意节点出发，沿父节点-子节点连接，达到任意节点的序列。同一个节点在一条路径序列中 至多出现一次 。该路径 至少包含一个 节点，且不一定经过根节点。

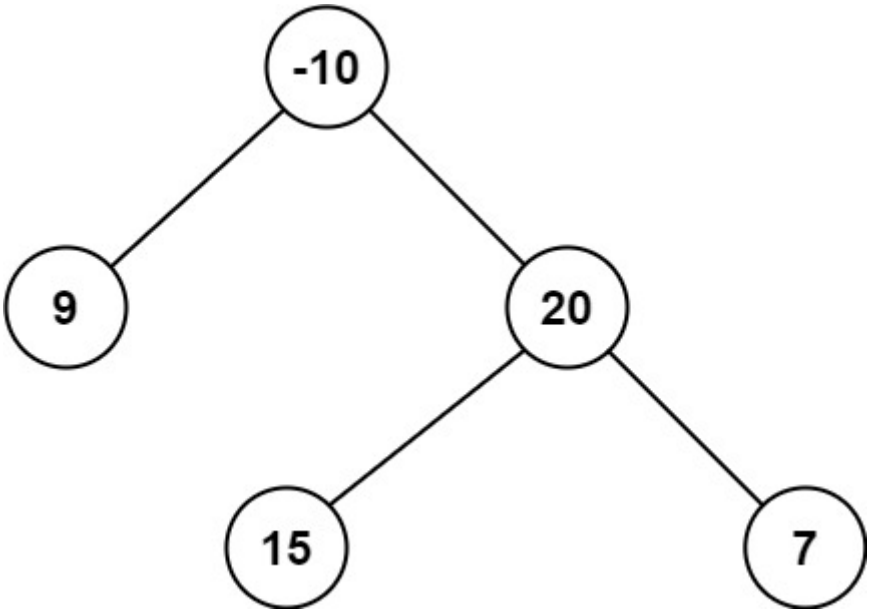
路径和 是路径中各节点值的总和。

给定一个二叉树的根节点 root ，返回其 最大路径和，即所有路径上节点值之和的最大值。

示例 1：



输入：root = [1,2,3]
输出：6
解释：最优路径是 2 -> 1 -> 3 ，路径和为 2 + 1 + 3 = 6
示例 2：



输入：root = [-10,9,20,null,null,15,7]
输出：42
解释：最优路径是 15 -> 20 -> 7 ，路径和为 15 + 20 + 7 = 42

提示：

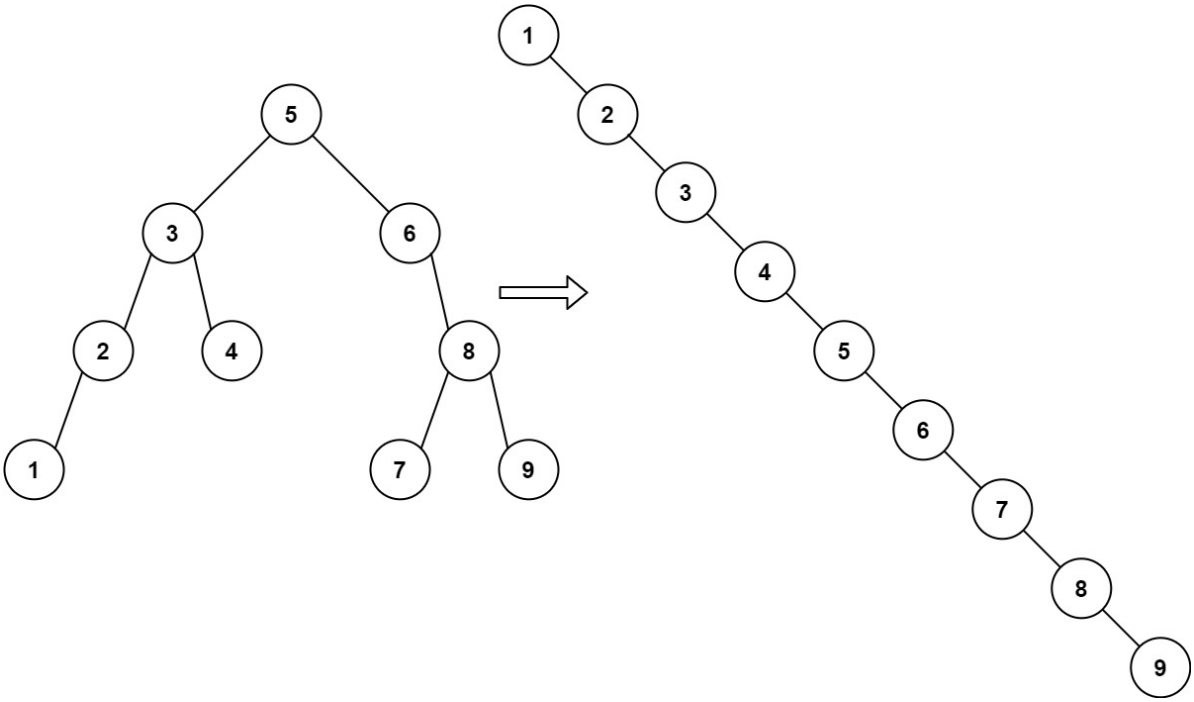
树中节点数目范围是 [1, 3 * 10^4]
-1000 <= Node.val <= 1000

```python
# python3
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def maxPathSum(self, root: TreeNode) -> int:
        if root==None: return 0
        self.res = root.val
        self.dfs(root)
        return self.res
    def dfs(self,root):
        if root == None: return 0

        leftDeep = max(0,self.dfs(root.left))
        rightDeep = max(0,self.dfs(root.right))

        tmp = leftDeep + rightDeep + root.val
        if tmp> self.res:
            self.res=tmp
        return root.val +max(leftDeep,rightDeep)
```

剑指 Offer II 052. 展平二叉搜索树

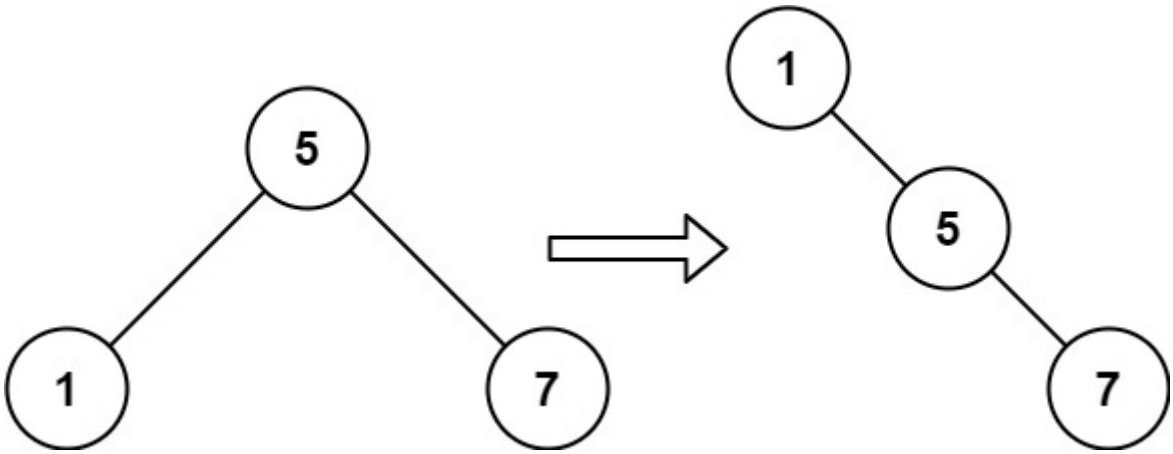
给你一棵二叉搜索树，请 按中序遍历 将其重新排列为一棵递增顺序搜索树，使树中最左边的节点成为树的根节点，并且每个节点没有左子节点，只有一个右子节点。

示例 1：




```
输入：root = [5,3,6,2,4,null,8,1,null,null,null,7,9]
输出：[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]
```

示例 2：




```
输入：root = [5,1,7]
输出：[1,null,5,null,7]
```

提示：

树中节点数的取值范围是 [1, 100]
0 <= Node.val <= 1000

```
# Definition for a binary tree node.
```

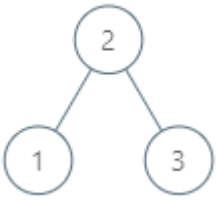
```python
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def increasingBST(self, root: TreeNode) -> TreeNode:
        self.res = []
        self.traverse(root)

        dumpy = TreeNode(-1)
        p = dumpy
        while self.res:
            p.right = TreeNode(self.res.pop(0))
            p = p.right
        return dumpy.right
    def traverse(self,root):
        if root==None: return

        self.traverse(root.left)
        self.res.append(root.val)
        self.traverse(root.right)
```

剑指 Offer II 053. 二叉搜索树中的中序后继

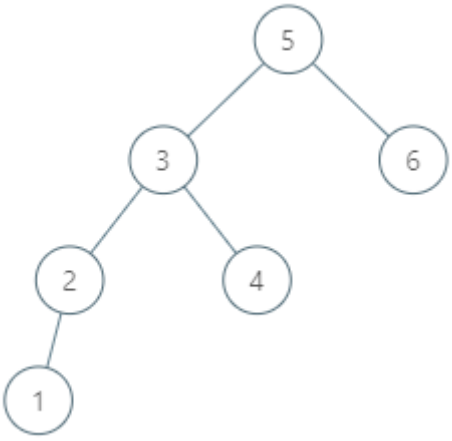
给定一棵二叉搜索树和其中的一个节点 p ，找到该节点在树中的中序后继。如果节点没有中序后继，请返回 null 。

节点 p 的后继是值比 p.val 大的节点中键值最小的节点，即按中序遍历的顺序节点 p 的下一个节点。

示例 1：



输入：root = [2,1,3], p = 1
输出：2
解释：这里 1 的中序后继是 2。请注意 p 和返回值都应是 TreeNode 类型。
示例 2：



输入：root = [5,3,6,2,4,null,null,1], p = 6
输出：null
解释：因为给出的节点没有中序后继，所以答案就返回 null 了。

提示：

树中节点的数目在范围 [1, 104] 内。
-105 <= Node.val <= 105
树中各节点的值均保证唯一。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def inorderSuccessor(self, root: 'TreeNode', p: 'TreeNode') -> 'TreeNode':
        self.res = []
        self.traverse(root)

        while self.res:
            tmp = self.res.pop(0)
            if tmp == p.val and len(self.res)>0:
                return TreeNode(self.res.pop(0))
        return None

    def traverse(self,root):
        if root==None: return

        self.traverse(root.left)
        self.res.append(root.val)
        self.traverse(root.right)
```

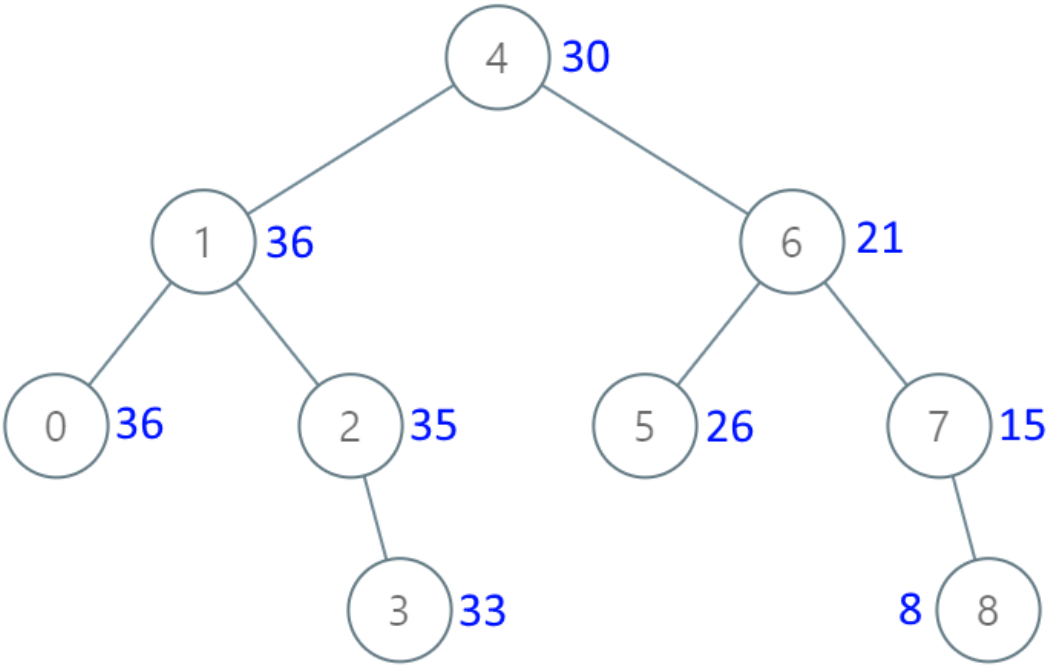
剑指 Offer II 054. 所有大于等于节点的值之和

给定一个二叉搜索树，请将它的每个节点的值替换成树中大于或者等于该节点值的所有节点值之和。

提醒一下，二叉搜索树满足下列约束条件：

节点的左子树仅包含键 小于 节点键的节点。
节点的右子树仅包含键 大于 节点键的节点。
左右子树也必须是二叉搜索树。

示例 1：



输入：root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
输出：[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
示例 2：

输入：root = [0,null,1]
输出：[1,null,1]
示例 3：

输入：root = [1,0,2]
输出：[3,3,2]
示例 4：

输入：root = [3,2,4,1]
输出：[7,9,4,10]

提示：

树中的节点数介于 0 和 104 之间。
每个节点的值介于 -104 和 104 之间。
树中的所有值 互不相同 。
给定的树为二叉搜索树。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def convertBST(self, root: TreeNode) -> TreeNode:
        self.total = 0
        self.dfs(root)
        return root
    def dfs(self,root):
        if  root == None:
            return
        self.dfs(root.right)
        self.total +=root.val
        root.val = self.total
        self.dfs(root.left)
```

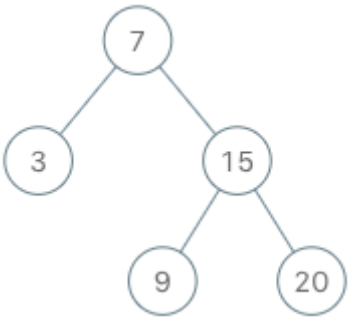
剑指 Offer II 055. 二叉搜索树迭代器

实现一个二叉搜索树迭代器类BSTIterator ，表示一个按中序遍历二叉搜索树（BST）的迭代器：

BSTIterator(TreeNode root) 初始化 BSTIterator 类的一个对象。BST 的根节点 root 会作为构造函数的一部分给出。指针应初始化为一个不存在于 BST 中的数字，且该数字小于 BST 中的任何元素。
boolean hasNext() 如果向指针右侧遍历存在数字，则返回 true ；否则返回 false 。
int next()将指针向右移动，然后返回指针处的数字。
注意，指针初始化为一个不存在于 BST 中的数字，所以对 next() 的首次调用将返回 BST 中的最小元素。

可以假设 next() 调用总是有效的，也就是说，当调用 next() 时，BST 的中序遍历中至少存在一个下一个数字。

示例：



输入
inputs = ["BSTIterator", "next", "next", "hasNext", "next", "hasNext", "next", "hasNext", "next", "hasNext"]
inputs = [[[7, 3, 15, null, null, 9, 20]], [], [], [], [], [], [], [], [], []]
输出
[null, 3, 7, true, 9, true, 15, true, 20, false]

解释
```
BSTIterator bSTIterator = new BSTIterator([7, 3, 15, null, null, 9, 20]);
bSTIterator.next();    // 返回 3
bSTIterator.next();    // 返回 7
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 9
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 15
bSTIterator.hasNext(); // 返回 True
bSTIterator.next();    // 返回 20
bSTIterator.hasNext(); // 返回 False
```

提示：

树中节点的数目在范围 [1, 105] 内
0 <= Node.val <= 106
最多调用 105 次 hasNext 和 next 操作

进阶：

你可以设计一个满足下述条件的解决方案吗？next() 和 hasNext() 操作均摊时间复杂度为 O(1) ，并使用 O(h) 内存。其中 h 是树的高度。

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class BSTIterator:

    def __init__(self, root: TreeNode):
        self.res=[]
        self.dfs(root)
```

```
    def next(self) -> int:
        return self.res.pop(0)

    def hasNext(self) -> bool:
        if len(self.res)>0:
            return True
        else:return False

    def dfs(self,root):
        if root==None:
            return
        self.dfs(root.left)
        self.res.append(root.val)
        self.dfs(root.right)
```

剑指 Offer II 056. 二叉搜索树中两个节点之和

给定一个二叉搜索树的 根节点 root 和一个整数 k ，请判断该二叉搜索树中是否存在两个节点它们的值之和等于 k 。假设二叉搜索树中节点的值均唯一。

示例 1：

输入: root = [8,6,10,5,7,9,11], k = 12
输出: true
解释: 节点 5 和节点 7 之和等于 12
示例 2：

输入: root = [8,6,10,5,7,9,11], k = 22
输出: false
解释: 不存在两个节点值之和为 22 的节点

提示：

二叉树的节点个数的范围是 [1, 104].
-104 <= Node.val <= 104
root 为二叉搜索树
-105 <= k <= 105

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def findTarget(self, root: TreeNode, k: int) -> bool:
        self.res=[]
        self.dfs(root)
```

```
        left, right=0,len(self.res)-1
        while left<right:
            total = self.res[left]+self.res[right]
            if total == k:
                return True
            elif total>k:
                right -=1
            elif total<k:
                left +=1
        return False



    def dfs(self,root):
        if root==None:
            return
        self.dfs(root.left)
        self.res.append(root.val)
        self.dfs(root.right)
```

(理解)剑指 Offer II 057. 值和下标之差都在给定的范围内

给你一个整数数组 nums 和两个整数 k 和 t 。请你判断是否存在 两个不同下标 i 和 j，使得 abs(nums[i] - nums[j]) <= t ，同时又满足 abs(i - j) <= k 。

如果存在则返回 true，不存在返回 false。

示例 1：

输入：nums = [1,2,3,1], k = 3, t = 0
输出：true
示例 2：

输入：nums = [1,0,1,1], k = 1, t = 2
输出：true
示例 3：

输入：nums = [1,5,9,1,5,9], k = 2, t = 3
输出：false

提示：

0 <= nums.length <= 2 * 104
-231 <= nums[i] <= 231 - 1
0 <= k <= 104
0 <= t <= 231 - 1

```python
class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) ->
bool:
        from sortedcontainers import SortedList
        st=SortedList()
        for i,num in enumerate(nums):
            st.add(num)
            index=bisect_left(st,num)
            if index<len(st)-1 and st[index+1]-st[index]<=t:return True
            if index>0 and st[index]-st[index-1]<=t:return True
            if len(st)>k:
                st.remove(nums[i-k])
        return False
```

剑指 Offer II 058. 日程表

请实现一个 MyCalendar 类来存放你的日程安排。如果要添加的时间内没有其他安排，则可以存储这个新的日程安排。

MyCalendar 有一个 book(int start, int end)方法。它意味着在 start 到 end 时间内增加一个日程安排，注意，这里的时间是半开区间，即 [start, end), 实数 x 的范围为， start <= x < end。

当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生重复预订。

每次调用 MyCalendar.book方法时，如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true。否则，返回 false 并且不要将该日程安排添加到日历中。

请按照以下步骤调用 MyCalendar 类: MyCalendar cal = new MyCalendar();
MyCalendar.book(start, end)

示例:

输入:
["MyCalendar","book","book","book"]
[[],[10,20],[15,25],[20,30]]
输出: [null,true,false,true]
解释:
MyCalendar myCalendar = new MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false ，第二个日程安排不能添加到日历中，因为时间 15 已经被第一个日程安排预定了
MyCalendar.book(20, 30); // returns true ，第三个日程安排可以添加到日历中，因为第一个日程安排并不包含时间 20

提示:

每个测试用例，调用 MyCalendar.book 函数最多不超过 1000次。
0 <= start < end <= 109

```python
from sortedcontainers import SortedDict as SD

class MyCalendar:

    def __init__(self):
        self.end_start = SD()

    def book(self, start: int, end: int) -> bool:
        ID = self.end_start.bisect_right(start)
        if 0 <= ID < len(self.end_start):
            if self.end_start.values()[ID] < end:
                return False
        self.end_start[end] = start
        return True


# Your MyCalendar object will be instantiated and called as such:
# obj = MyCalendar()
# param_1 = obj.book(start,end)
```

剑指 Offer II 059. 数据流的第 K 大数值

设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。

请实现 KthLargest 类：

KthLargest(int k, int[] nums) 使用整数 k 和整数流 nums 初始化对象。
int add(int val) 将 val 插入数据流 nums 后，返回当前数据流中第 k 大的元素。

示例：

输入：
["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
输出：
[null, 4, 5, 5, 8, 8]

解释：
KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
kthLargest.add(3);   // return 4
kthLargest.add(5);   // return 5
kthLargest.add(10); // return 5
kthLargest.add(9);   // return 8
kthLargest.add(4);   // return 8

提示：

1 <= k <= 104
0 <= nums.length <= 104
-104 <= nums[i] <= 104

-104 <= val <= 104
最多调用 add 方法 104 次
题目数据保证，在查找第 k 大元素时，数组中至少有 k 个元素

```
class KthLargest:

    def __init__(self, k: int, nums: List[int]):
        self.k = k
        self.minHeap = []
        for x in nums:
            heapq.heappush(self.minHeap, x)

    def add(self, val: int) -> int:
        heapq.heappush(self.minHeap, val)

        while len(self.minHeap) > self.k:
            heapq.heappop(self.minHeap)

        return self.minHeap[0]
```

注意：

1.heappush(heap,item)建立大、小根堆
heapq.heappush()是往堆中添加新值，此时自动建立了小根堆
不能直接建立大跟堆，所以每次push时给元素加一个负号（即取相反数），此时最小值变最大值，反之亦然，那么实际上的最大值就可以处于堆顶了，返回时再取负即可。
2.heapq.heappop()从堆中弹出并返回最小的值
普通list（即并没有进行heapify等操作的list），对他进行heappop操作并不会弹出list中最小的值，而是弹出第一个值。
对于小跟堆，会依次弹出最小的值。
所以针对最小 top k 问题用最大堆，求最大 top k 问题用最小堆
比如leetcode347中，求最大top k问题，先求出各个数字出现的频率，再建立一个小跟堆，每次把频率最小的值弹出来，最后剩下一个k大小的list，就是要求的前k个高频元素了。

剑指 Offer II 060. 出现频率最高的 k 个数字

给定一个整数数组 nums 和一个整数 k ，请返回其中出现频率前 k 高的元素。可以按 任意顺序 返回答案。

示例 1:

输入: nums = [1,1,1,2,2,3], k = 2
输出: [1,2]
示例 2:

输入: nums = [1], k = 1
输出: [1]

提示:

```
1 <= nums.length <= 105
k 的取值范围是 [1, 数组中不相同的元素的个数]
题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的
```

```python
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        num_freq = collections.Counter(nums)
        minHeap = []

        for x, f in num_freq.items():
            if len(minHeap) == k:
                if minHeap[0][0] < f:
                    heapq.heappop(minHeap)
                    heapq.heappush(minHeap, (f, x))
            else:
                heapq.heappush(minHeap, (f, x))

        res = []
        while minHeap:
            f, x = heapq.heappop(minHeap)
            res.append(x)
        return res
```

(理解)剑指 Offer II 061. 和最小的 k 个数对

```
给定两个以升序排列的整数数组 nums1 和 nums2 , 以及一个整数 k 。

定义一对值 (u,v)，其中第一个元素来自 nums1，第二个元素来自 nums2 。

请找到和最小的 k 个数对 (u1,v1),  (u2,v2)  ...  (uk,vk) 。



示例 1:

输入: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
输出: [1,2],[1,4],[1,6]
解释: 返回序列中的前 3 对数：
    [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
示例 2:

输入: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
输出: [1,1],[1,1]
解释: 返回序列中的前 2 对数：
     [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
示例 3:

输入: nums1 = [1,2], nums2 = [3], k = 3
输出: [1,3],[2,3]
解释: 也可能序列中所有的数对都被返回:[1,3],[2,3]
```

提示：

1 <= nums1.length, nums2.length <= 104
-109 <= nums1[i], nums2[i] <= 109
nums1, nums2 均为升序排列
1 <= k <= 1000

```
class Solution:
    def kSmallestPairs(self, nums1: List[int], nums2: List[int], k: int) ->
List[List[int]]:
        n1 = len(nums1)
        n2 = len(nums2)

        maxHeap = []
        for i in range(min(n1, k)):
            for j in range(min(n2, k)):
                if len(maxHeap) < k:
                    heapq.heappush(maxHeap, (-(nums1[i] + nums2[j]), nums1[i],
nums2[j]))
                else:
                    if -maxHeap[0][0] > nums1[i] + nums2[j]:
                        heapq.heappop(maxHeap)
                        heapq.heappush(maxHeap, (-(nums1[i] + nums2[j]),
nums1[i], nums2[j]) )

        res = []
        while maxHeap:
            _, x, y = heapq.heappop(maxHeap)
            res.append([x, y])
        return res
```

(理解)剑指 Offer II 062. 实现前缀树

Trie（发音类似 "try"）或者说 前缀树 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

Trie() 初始化前缀树对象。
void insert(String word) 向前缀树中插入字符串 word 。
boolean search(String word) 如果字符串 word 在前缀树中，返回 true（即，在检索之前已经插入）；否则，返回 false 。
boolean startsWith(String prefix) 如果之前已经插入的字符串 word 的前缀之一为 prefix ，返回 true ；否则，返回 false 。

示例：

输入
inputs = ["Trie", "insert", "search", "search", "startsWith", "insert",
"search"]

```
inputs = [[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True


提示：

1 <= word.length, prefix.length <= 2000
word 和 prefix 仅由小写英文字母组成
insert、search 和 startsWith 调用次数 总计 不超过 3 * 104 次
```

```python
class Trie:

    def __init__(self):
        self.next = dict()
        self.end = False

    def insert(self, word: str) -> None:
        cur = self
        for c in word:
            if not cur.next.get(c):
                cur.next[c] = Trie()
            cur = cur.next[c]
        cur.end = True

    def search(self, word: str) -> bool:
        cur = self
        for c in word:
            if not cur.next.get(c):
                return False
            cur = cur.next[c]
        return cur.end

    def startsWith(self, prefix: str) -> bool:
        cur = self
        for c in prefix:
            if not cur.next.get(c):
                return False
            cur = cur.next[c]
        return True


# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
```

```
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)
```

(理解)剑指 Offer II 063. 替换单词

在英语中，有一个叫做 词根(root) 的概念，它可以跟着其他一些词组成另一个较长的单词——我们称这个词为 继承词(successor)。例如，词根an，跟随着单词 other(其他)，可以形成新的单词 another(另一个)。

现在，给定一个由许多词根组成的词典和一个句子，需要将句子中的所有继承词用词根替换掉。如果继承词有许多可以形成它的词根，则用最短的词根替换它。

需要输出替换之后的句子。

示例 1：

输入：dictionary = ["cat","bat","rat"], sentence = "the cattle was rattled by the battery"
输出："the cat was rat by the bat"
示例 2：

输入：dictionary = ["a","b","c"], sentence = "aadsfasf absbs bbab cadsfafs"
输出："a a b c"
示例 3：

输入：dictionary = ["a", "aa", "aaa", "aaaa"], sentence = "a aa a aaaa aaa aaa aaa aaaaaa bbb baba ababa"
输出："a a a a a a a bbb baba a"
示例 4：

输入：dictionary = ["catt","cat","bat","rat"], sentence = "the cattle was rattled by the battery"
输出："the cat was rat by the bat"
示例 5：

输入：dictionary = ["ac","ab"], sentence = "it is abnormal that this solution is accepted"
输出："it is ab that this solution is ac"

提示：

1 <= dictionary.length <= 1000
1 <= dictionary[i].length <= 100
dictionary[i] 仅由小写字母组成。
1 <= sentence.length <= 10^6
sentence 仅由小写字母和空格组成。
sentence 中单词的总量在范围 [1, 1000] 内。
sentence 中每个单词的长度在范围 [1, 1000] 内。
sentence 中单词之间由一个空格隔开。
sentence 没有前导或尾随空格。

```
class Trie:
    def __init__(self):
        self.child = [None for _ in range(26)]
        self.isWord = False

    def insert(self, word: str) -> None:
        root = self
        for c in word:
            ID = ord(c) - ord('a')
            if root.child[ID] == None:
                root.child[ID] = Trie()
            root = root.child[ID]
        root.isWord = True

    def query(self, word: str) -> str:
        root = self
        i = 0
        for c in word:
            ID = ord(c) - ord('a')
            if root.child[ID] == None:      #如果没有前缀，直接返回word
                return word
            root = root.child[ID]
            i += 1
            if root.isWord == True:
                break
        return word[ : i]

class Solution:
    def replaceWords(self, dictionary: List[str], sentence: str) -> str:
        T = Trie()
        for word in dictionary:
            T.insert(word)

        s = sentence.split(" ")
        n = len(s)
        for i, word in enumerate(s):
            s[i] = T.query(word)
        return ' '.join(s)
```

(理解)剑指 Offer II 064. 神奇的字典

设计一个使用单词列表进行初始化的数据结构，单词列表中的单词 互不相同 。 如果给出一个单词，请判定能否只将这个单词中一个字母换成另一个字母，使得所形成的新单词存在于已构建的神奇字典中。

实现 MagicDictionary 类：

MagicDictionary() 初始化对象
void buildDict(String[] dictionary) 使用字符串数组 dictionary 设定该数据结构，dictionary 中的字符串互不相同

bool search(String searchWord) 给定一个字符串 searchWord ，判定能否只将字符串中 一个 字母换成另一个字母，使得所形成的新字符串能够与字典中的任一字符串匹配。如果可以，返回 true ；否则，返回 false 。

示例：

输入
inputs = ["MagicDictionary", "buildDict", "search", "search", "search", "search"]
inputs = [[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
输出
[null, null, false, true, false, false]

解释
MagicDictionary magicDictionary = new MagicDictionary();
magicDictionary.buildDict(["hello", "leetcode"]);
magicDictionary.search("hello"); // 返回 False
magicDictionary.search("hhllo"); // 将第二个 'h' 替换为 'e' 可以匹配 "hello" ，所以返回 True
magicDictionary.search("hell"); // 返回 False
magicDictionary.search("leetcoded"); // 返回 False

提示：

1 <= dictionary.length <= 100
1 <= dictionary[i].length <= 100
dictionary[i] 仅由小写英文字母组成
dictionary 中的所有字符串 互不相同
1 <= searchWord.length <= 100
searchWord 仅由小写英文字母组成
buildDict 仅在 search 之前调用一次
最多调用 100 次 search

```python
class MagicDictionary:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.dic = {}

    def buildDict(self, dictionary: List[str]) -> None:
        for s in dictionary:
            if len(s) not in self.dic:
                self.dic[len(s)] = [s]
            else:
                self.dic[len(s)].append(s)

    def search(self, searchWord: str) -> bool:
        if len(searchWord) not in self.dic:
            return False
```

```
        ismagic = False
        l = self.dic[len(searchWord)]
        for s in l: # 对每个相同长度词进行比较进行比较

            wrongtimes = 0
            for i in range(len(s)):
                if s[i] != searchWord[i]:
                    wrongtimes += 1
            if wrongtimes == 1:
                return True
        return False
```

剑指 Offer II 065. 最短的单词编码

单词数组 words 的 有效编码 由任意助记字符串 s 和下标数组 indices 组成，且满足：

words.length == indices.length
助记字符串 s 以 '#' 字符结尾
对于每个下标 indices[i] ，s 的一个从 indices[i] 开始、到下一个 '#' 字符结束（但不包括 '#'）的 子字符串 恰好与 words[i] 相等
给定一个单词数组 words ，返回成功对 words 进行编码的最小助记字符串 s 的长度 。

示例 1：

输入：words = ["time", "me", "bell"]
输出：10
解释：一组有效编码为 s = "time#bell#" 和 indices = [0, 2, 5] 。
words[0] = "time" ，s 开始于 indices[0] = 0 到下一个 '#' 结束的子字符串，如加粗部分所示 "time#bell#"
words[1] = "me" ，s 开始于 indices[1] = 2 到下一个 '#' 结束的子字符串，如加粗部分所示 "time#bell#"
words[2] = "bell" ，s 开始于 indices[2] = 5 到下一个 '#' 结束的子字符串，如加粗部分所示 "time#bell#"
示例 2：

输入：words = ["t"]
输出：2
解释：一组有效编码为 s = "t#" 和 indices = [0] 。

提示：

1 <= words.length <= 2000
1 <= words[i].length <= 7
words[i] 仅由小写字母组成

```
class Solution:
    def minimumLengthEncoding(self, words: List[str]) -> int:
        hashset = set()
        for word in words:
            hashset.add(word)
```

```
        n = len(words)
        for i in range(n):
            for j in range(1, len(words[i])):
                if words[i][j:] in hashset:
                    hashset.remove(words[i][j:])
        ans = 0
        for i in hashset:
            ans += len(i) + 1
        return ans
```

剑指 Offer II 066. 单词之和

实现一个 MapSum 类，支持两个方法，insert 和 sum：

MapSum() 初始化 MapSum 对象
void insert(String key, int val) 插入 key-val 键值对，字符串表示键 key ，整数表示值 val 。如果键 key 已经存在，那么原来的键值对将被替代成新的键值对。
int sum(string prefix) 返回所有以该前缀 prefix 开头的键 key 的值的总和。

示例：

输入：
inputs = ["MapSum", "insert", "sum", "insert", "sum"]
inputs = [[], ["apple", 3], ["ap"], ["app", 2], ["ap"]]
输出：
[null, null, 3, null, 5]

解释：
MapSum mapSum = new MapSum();
mapSum.insert("apple", 3);
mapSum.sum("ap");   // return 3 (apple = 3)
mapSum.insert("app", 2);
mapSum.sum("ap");   // return 5 (apple + app = 3 + 2 = 5)

提示：

1 <= key.length, prefix.length <= 50
key 和 prefix 仅由小写英文字母组成
1 <= val <= 1000
最多调用 50 次 insert 和 sum

```
class MapSum:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.map = {}
```

```
    def insert(self, key: str, val: int) -> None:
        self.map[key] = val


    def sum(self, prefix: str) -> int:
        count = 0

        for ii in self.map:
            if ii[0:len(prefix)] == prefix:
                count += self.map[ii]
        return count



# Your MapSum object will be instantiated and called as such:
# obj = MapSum()
# obj.insert(key,val)
# param_2 = obj.sum(prefix)
```

(理解) 剑指 Offer II 067. 最大的异或

给你一个整数数组 nums ，返回 nums[i] XOR nums[j] 的最大运算结果，其中 0 ≤ i ≤ j < n 。

示例 1：

输入：nums = [3,10,5,25,2,8]
输出：28
解释：最大运算结果是 5 XOR 25 = 28.
示例 2：

输入：nums = [14,70,53,83,49,91,36,80,92,51,66,70]
输出：127

提示：

1 <= nums.length <= 2 * 105
0 <= nums[i] <= 231 - 1

```
class Solution:
    def findMaximumXOR(self, nums: List[int]) -> int:
        l = len(nums)
        mask = 0
        res = 0
        for i in range(30, -1, -1):
            mask = mask | (1 << i)

            newSet = set()
            for num in nums:
                newSet.add(mask & num)
```

```
            targetMax = res | (1 << i)
            for prefix in newSet:
                if prefix ^ targetMax in newSet:
                    res = targetMax
                    break
        return res
```

剑指 Offer II 068. 查找插入位置

给定一个排序的整数数组 nums 和一个整数目标值 target ，请在数组中找到 target ，并返回其下标。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请必须使用时间复杂度为 O(log n) 的算法。

示例 1:

输入: nums = [1,3,5,6], target = 5
输出: 2
示例 2:

输入: nums = [1,3,5,6], target = 2
输出: 1
示例 3:

输入: nums = [1,3,5,6], target = 7
输出: 4
示例 4:

输入: nums = [1,3,5,6], target = 0
输出: 0
示例 5:

输入: nums = [1], target = 0
输出: 0

提示:

1 <= nums.length <= 104
-104 <= nums[i] <= 104
nums 为无重复元素的升序排列数组
-104 <= target <= 104

```python
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left , right = 0,len(nums)-1
        while(left <= right):
            mid = left +(right-left)//2
            if (nums[mid] == target):
                return mid
            elif (nums[mid] > target):
                right = mid-1
            elif (nums[mid] < target):
                left = mid+1
        return left
```

剑指 Offer II 069. 山峰数组的顶部

符合下列属性的数组 arr 称为 山峰数组（山脉数组） ：

arr.length >= 3
存在 i（0 < i < arr.length - 1）使得：
arr[0] < arr[1] < ... arr[i-1] < arr[i]
arr[i] > arr[i+1] > ... > arr[arr.length - 1]
给定由整数组成的山峰数组 arr ，返回任何满足 arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1] 的下标 i ，即山峰顶部。

示例 1：

输入：arr = [0,1,0]
输出：1
示例 2：

输入：arr = [1,3,5,4,2]
输出：2
示例 3：

输入：arr = [0,10,5,2]
输出：1
示例 4：

输入：arr = [3,4,5,1]
输出：2
示例 5：

输入：arr = [24,69,100,99,79,78,67,36,26,19]
输出：2

提示：

3 <= arr.length <= 104
0 <= arr[i] <= 106
题目数据保证 arr 是一个山脉数组

进阶：很容易想到时间复杂度 O(n) 的解决方案，你可以设计一个 O(log(n)) 的解决方案吗？

```python
class Solution:
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        for i in range(1, len(arr)):
            if arr[i - 1] > arr[i]:
                return i - 1
```

剑指 Offer II 070. 排序数组中只出现一次的数字

给定一个只包含整数的有序数组 nums ，每个元素都会出现两次，唯有一个数只会出现一次，请找出这个唯一的数字。

你设计的解决方案必须满足 O(log n) 时间复杂度和 O(1) 空间复杂度。

示例 1:

输入: nums = [1,1,2,3,3,4,4,8,8]
输出: 2
示例 2:

输入: nums = [3,3,7,7,10,11,11]
输出: 10

提示:

1 <= nums.length <= 105
0 <= nums[i] <= 105

```python
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        res = 0
        for x in nums:
            res ^= x
        return res
```

(理解)剑指 Offer II 071. 按权重生成随机数

给定一个正整数数组 w ，其中 w[i] 代表下标 i 的权重（下标从 0 开始），请写一个函数 pickIndex ，它可以随机地获取下标 i，选取下标 i 的概率与 w[i] 成正比。

例如，对于 w = [1, 3]，挑选下标 0 的概率为 1 / (1 + 3) = 0.25 （即，25%），而选取下标 1 的概率为 3 / (1 + 3) = 0.75（即，75%）。

也就是说，选取下标 i 的概率为 w[i] / sum(w) 。

示例 1：

输入：
inputs = ["Solution","pickIndex"]
inputs = [[[1]],[]]
输出：
[null,0]
解释：
Solution solution = new Solution([1]);
solution.pickIndex(); // 返回 0，因为数组中只有一个元素，所以唯一的选择是返回下标 0。
示例 2：

输入：
inputs =
["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"]
inputs = [[[1,3]],[],[],[],[],[]]
输出：
[null,1,1,1,1,0]
解释：
Solution solution = new Solution([1, 3]);
solution.pickIndex(); // 返回 1，返回下标 1，返回该下标概率为 3/4 。
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 1
solution.pickIndex(); // 返回 0，返回下标 0，返回该下标概率为 1/4 。

由于这是一个随机问题，允许多个答案，因此下列输出都可以被认为是正确的:
[null,1,1,1,1,0]
[null,1,1,1,1,1]
[null,1,1,1,0,0]
[null,1,1,1,0,1]
[null,1,0,1,0,0]
......
诸若此类。

提示：

1 <= w.length <= 10000
1 <= w[i] <= 10^5
pickIndex 将被调用不超过 10000 次

```python
class Solution:

    def __init__(self, w: List[int]):
        # 存储前缀和
        self.prefixnum=list(accumulate(w))
        # 求得权重和
        self.total=sum(w)

    def pickIndex(self) -> int:
```

```
        # 从1-10进行随机产生整数
        x=random.randint(1,self.total)
        # 进行随机产生数与前缀和判断
        # 返回比x大的第一个下标值
        return bisect_left(self.prefixnum,x)
```

剑指 Offer II 072. 求平方根

给定一个非负整数 x ，计算并返回 x 的平方根，即实现 int sqrt(int x) 函数。

正数的平方根有两个，只输出其中的正数平方根。

如果平方根不是整数，输出只保留整数的部分，小数部分将被舍去。

示例 1:

输入: x = 4
输出: 2
示例 2:

输入: x = 8
输出: 2
解释: 8 的平方根是 2.82842...，由于小数部分将被舍去，所以返回 2

提示:

0 <= x <= 231 - 1

```
class Solution:
    def mySqrt(self, x: int) -> int:
        return int(x**0.5)
```

(理解)剑指 Offer II 073. 狒狒吃香蕉

狒狒喜欢吃香蕉。这里有 n 堆香蕉，第 i 堆中有 piles[i] 根香蕉。警卫已经离开了，将在 h 小时后回来。

狒狒可以决定她吃香蕉的速度 k （单位：根/小时）。每个小时，她将会选择一堆香蕉，从中吃掉 k 根。如果这堆香蕉少于 k 根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉，下一个小时才会开始吃另一堆的香蕉。

狒狒喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。

返回她可以在 h 小时内吃掉所有香蕉的最小速度 k（k 为整数）。

示例 1:

输入：piles = [3,6,7,11], h = 8
输出：4
示例 2：

输入：piles = [30,11,23,4,20], h = 5
输出：30
示例 3：

输入：piles = [30,11,23,4,20], h = 6
输出：23

提示：

1 <= piles.length <= 104
piles.length <= h <= 109
1 <= piles[i] <= 109

```python
class Solution:
    def minEatingSpeed(self, piles: List[int], h: int) -> int:
        left, right = 1, max(piles)
        while left <= right:
            mid = (left + right) // 2
            T = 0
            for pile in piles:
                T += ceil(pile / mid)
                if T > h:
                    break
            if T == h:
                right = mid - 1
            elif T > h:
                left = mid + 1
            elif T < h:
                right = mid - 1
        return left
```

剑指 Offer II 074. 合并区间

以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。

示例 1：

输入：intervals = [[1,3],[2,6],[8,10],[15,18]]
输出：[[1,6],[8,10],[15,18]]
解释：区间 [1,3] 和 [2,6] 重叠，将它们合并为 [1,6].
示例 2：

输入：intervals = [[1,4],[4,5]]
输出：[[1,5]]

解释：区间 [1,4] 和 [4,5] 可被视为重叠区间。

提示：

1 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti <= endi <= 104

```python
class Solution:
    def merge(self, intervals: List[List[int]]) -> List[List[int]]:
        intervals.sort()
        res = []
        for s, e in intervals:
            if len(res) == 0 or res[-1][1] < s:
                res.append([s, e])
            else:
                res[-1][1] = max(res[-1][1], e)
        return res
```

剑指 Offer II 075. 数组相对排序

给定两个数组，arr1 和 arr2，

arr2 中的元素各不相同
arr2 中的每个元素都出现在 arr1 中
对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。

示例：

输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
输出：[2,2,2,1,4,3,3,9,6,7,19]

提示：

1 <= arr1.length, arr2.length <= 1000
0 <= arr1[i], arr2[i] <= 1000
arr2 中的元素 arr2[i] 各不相同
arr2 中的每个元素 arr2[i] 都出现在 arr1 中

```python
class Solution:
    def relativeSortArray(self, arr1: List[int], arr2: List[int]) -> List[int]:
        map = [0 for _ in range(1001)]
        res = [0 for _ in range(len(arr1))]
        loc = 0
        for i in arr1:
            map[i] += 1
        # 按照arr2的顺序，填充数字
```

```
        for i in range(len(arr2)):
            while (map[arr2[i]] > 0):
                res[loc] = arr2[i]
                map[arr2[i]] -= 1
                loc += 1
        # 未出现的数字，需要按照升序排列，所以，我们从0 ~ 1001遍历
        for i in range(len(map)):
            while (map[i] > 0):
                res[loc] = i
                map[i] -= 1
                loc += 1
        return res
```

注意，字典存储arr1的值

剑指 Offer II 076. 数组中的第 k 大的数字

给定整数数组 nums 和整数 k，请返回数组中第 k 个最大的元素。

请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

示例 1:

输入: [3,2,1,5,6,4] 和 k = 2
输出: 5
示例 2:

输入: [3,2,3,1,2,4,5,5,6] 和 k = 4
输出: 4

提示:

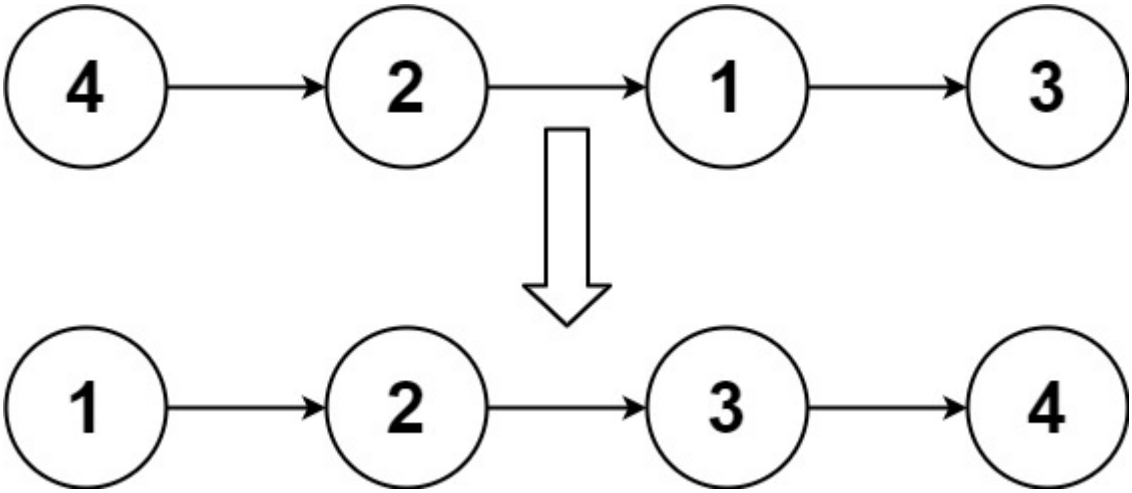
1 <= k <= nums.length <= 104
-104 <= nums[i] <= 104

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        nums.sort()
        return nums[-k]
```

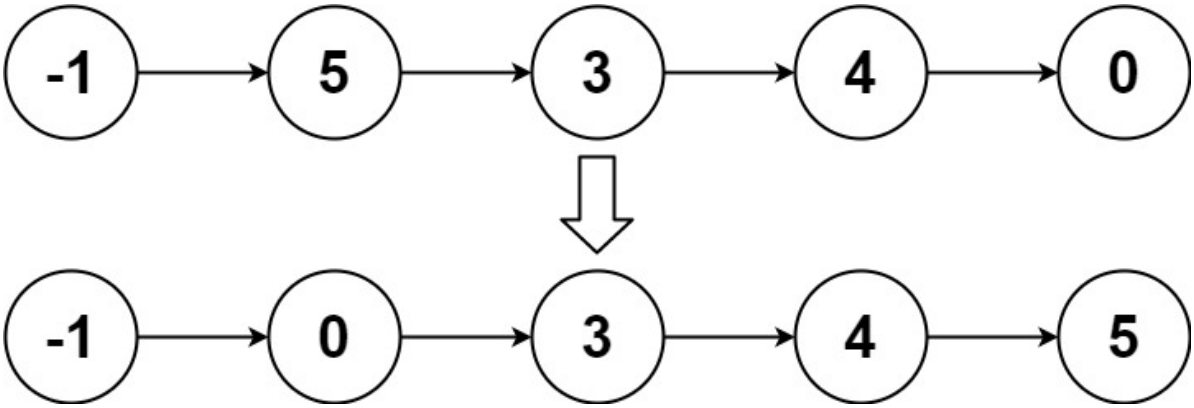
剑指 Offer II 077. 链表排序

给定链表的头结点 head ，请将其按 升序 排列并返回 排序后的链表 。

示例 1:



输入：head = [4,2,1,3]
输出：[1,2,3,4]
示例 2：



输入：head = [-1,5,3,4,0]
输出：[-1,0,3,4,5]
示例 3：

输入：head = []
输出：[]

提示：

链表中节点的数目在范围 [0, 5 * 104] 内
-105 <= Node.val <= 105

```python
# python3
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if not head:return None
        alist = []
        while head:
```

```
            alist.append(head.val)
            head = head.next
        alist.sort()
        dumpy = ListNode(-1)
        p = dumpy
        while alist:
            p.next =ListNode(alist.pop(0))
            p = p.next
        return dumpy.next
```

剑指 Offer II 078. 合并排序链表

给定一个链表数组，每个链表都已经按升序排列。

请将所有链表合并到一个升序链表中，返回合并后的链表。

示例 1：

输入：lists = [[1,4,5],[1,3,4],[2,6]]
输出：[1,1,2,3,4,4,5,6]
解释：链表数组如下：
[
  1->4->5,
  1->3->4,
  2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6
示例 2：

输入：lists = []
输出：[]
示例 3：

输入：lists = [[]]
输出：[]

提示：

k == lists.length
0 <= k <= 10^4
0 <= lists[i].length <= 500
-10^4 <= lists[i][j] <= 10^4
lists[i] 按 升序 排列
lists[i].length 的总和不超过 10^4

```
# Definition for singly-linked list.
# class ListNode:
```

```
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        que = []
        for head in lists:
            while(head):
                que.append(head.val)
                head = head.next
        que.sort()
        dumpy = ListNode(-1)
        p = dumpy
        while que:
            p.next =ListNode(que.pop(0))
            p = p.next
        return dumpy.next
```

剑指 Offer II 079. 所有子集

给定一个整数数组 nums ，数组中的元素 互不相同 。返回该数组所有可能的子集（幂集）。

解集 不能 包含重复的子集。你可以按 任意顺序 返回解集。

示例 1：

输入：nums = [1,2,3]
输出：[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
示例 2：

输入：nums = [0]
输出：[[],[0]]

提示：

1 <= nums.length <= 10
-10 <= nums[i] <= 10
nums 中的所有元素 互不相同

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        self.res = []
        used = [False] * len(nums)
        track = []
        self.backtrack(nums,used,track,0)
        return self.res

    def backtrack(self,nums,used,track,start):
        self.res.append(track[:])
```

```
        for i in range(start, len(nums)):
            if used[i]: continue
            track.append(nums[i])
            used[i] = True
            self.backtrack(nums,used,track,i+1)
            track.pop()
            used[i] = False
```

剑指 Offer II 080. 含有 k 个元素的组合

给定两个整数 n 和 k，返回 1 ... n 中所有可能的 k 个数的组合。

示例 1:

输入: n = 4, k = 2
输出:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
示例 2:

输入: n = 1, k = 1
输出: [[1]]

提示:

1 <= n <= 20
1 <= k <= n

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        self.res = []
        used = [False] * (n+1)
        track = []
        self.backtrack(n,k,used,track,1)
        return self.res
    def backtrack(self,n,k,used,track,start):
        if len(track)==k:
            self.res.append(track[:])
        for i in range(start,n+1):
            track.append(i)
            used[i] = True
            self.backtrack(n,k,used,track,i+1)
            track.pop()
```

```
                used[i] = False
```

剑指 Offer II 081. 允许重复选择元素的组合

给定一个无重复元素的正整数数组 candidates 和一个正整数 target ，找出 candidates 中所有可以使数字和为目标数 target 的唯一组合。

candidates 中的数字可以无限制重复被选取。如果至少一个所选数字数量不同，则两种组合是不同的。

对于给定的输入，保证和为 target 的唯一组合数少于 150 个。

示例 1：

输入: candidates = [2,3,6,7], target = 7
输出: [[7],[2,2,3]]
示例 2：

输入: candidates = [2,3,5], target = 8
输出: [[2,2,2,2],[2,3,3],[3,5]]
示例 3：

输入: candidates = [2], target = 1
输出: []
示例 4：

输入: candidates = [1], target = 1
输出: [[1]]
示例 5：

输入: candidates = [1], target = 2
输出: [[1,1]]

提示：

1 <= candidates.length <= 30
1 <= candidates[i] <= 200
candidate 中的每个元素都是独一无二的。
1 <= target <= 500

```python
class Solution:
    def combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
        self.res = []
        track = []
        self.backtrack(candidates,target,track)
        return self.res

    def backtrack(self,candidates,target,track):
        if(sum(track)==target):
            self.res.append(track[:])
```

```
            return
        if(sum(track)> target):
            return
        for candidate in candidates:
            if (len(track)>0 and track[-1] > candidate): continue
            track.append(candidate)
            self.backtrack(candidates,target,track)
            track.pop()
```

剑指 Offer II 082. 含有重复元素集合的组合

给定一个可能有重复数字的整数数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的每个数字在每个组合中只能使用一次，解集不能包含重复的组合。

示例 1:

输入: candidates = [10,1,2,7,6,1,5], target = 8,
输出:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
示例 2:

输入: candidates = [2,5,2,1,2], target = 5,
输出:
[
[1,2,2],
[5]
]

提示:

1 <= candidates.length <= 100
1 <= candidates[i] <= 50
1 <= target <= 30

```
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) ->
List[List[int]]:
        self.res = []
        track = []
        used = [False] * len(candidates)
        candidates.sort()
        self.backtrack(candidates,target,track,used,0)
        return self.res
```

```
    def backtrack(self,candidates,target,track,used,start):
        if(sum(track)==target):
            self.res.append(track[:])
            return
        if(sum(track)> target):
            return
        for i in range(start,len(candidates)):
            if used[i]: continue
            if i > start and candidates[i] == candidates[i-1]: continue

            track.append(candidates[i])
            used[i] = True
            self.backtrack(candidates,target,track,used,i+1)
            track.pop()
            used[i] = False
```

剑指 Offer II 083. 没有重复元素集合的全排列

给定一个不含重复数字的整数数组 nums ，返回其 所有可能的全排列 。可以 按任意顺序 返回答案。

示例 1：

输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
示例 2：

输入：nums = [0,1]
输出：[[0,1],[1,0]]
示例 3：

输入：nums = [1]
输出：[[1]]

提示：

1 <= nums.length <= 6
-10 <= nums[i] <= 10
nums 中的所有整数 互不相同

```
class Solution:

    def permute(self, nums: List[int]) -> List[List[int]]:
        self.res = []
        track = []
        used = [False] *  len(nums)
        self.backtrack(nums,track,used)
        return self.res
    def backtrack(self,nums,track,used):
```

```
        if len(track)==len(nums):
            self.res.append(track[:])
        for i in range(len(nums)):
            if used[i]: continue
            track.append(nums[i])
            used[i] = True
            self.backtrack(nums,track,used)
            track.pop()
            used[i] = False
```

剑指 Offer II 084. 含有重复元素集合的全排列

给定一个可包含重复数字的整数集合 nums ，按任意顺序 返回它所有不重复的全排列。

示例 1：

输入：nums = [1,1,2]
输出：
[[1,1,2],
 [1,2,1],
 [2,1,1]]
示例 2：

输入：nums = [1,2,3]
输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

提示：

1 <= nums.length <= 8
-10 <= nums[i] <= 10

```
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        self.res = []
        track = []
        used = [False] *  len(nums)
        nums.sort()
        self.backtrack(nums,track,used)
        return self.res
    def backtrack(self,nums,track,used):
        if len(track)==len(nums):
            self.res.append(track[:])
        for i in range(len(nums)):
            if used[i]: continue
            if i>0 and nums[i]==nums[i-1] and not used[i-1]: continue
            track.append(nums[i])
            used[i] = True
```

```
            self.backtrack(nums,track,used)
            track.pop()
            used[i] = False
```

(理解)剑指 Offer II 085. 生成匹配的括号

正整数 n 代表生成括号的对数，请设计一个函数，用于能够生成所有可能的并且 有效的 括号组合。

示例 1：

输入：n = 3
输出：["((()))","(()())","(())()","()(())","()()()"]
示例 2：

输入：n = 1
输出：["()"]

提示：

1 <= n <= 8

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:

        def backtrace(left: int, right: int) -> None:
            nonlocal path
            if left == n and right == n:
                res.append(''.join(path))

            if left < n:
                path.append('(')
                backtrace(left + 1, right)
                path.pop()
            if left > right:
                path.append(')')
                backtrace(left, right + 1)
                path.pop()

        res = []
        path = []
        backtrace(0, 0)
        return res
```

(理解)剑指 Offer II 086. 分割回文子字符串

给定一个字符串 s ，请将 s 分割成一些子串，使每个子串都是 回文串 ，返回 s 所有可能的分割方案。

回文串 是正着读和反着读都一样的字符串。

示例 1：

输入：s = "google"
输出：[["g","o","o","g","l","e"],["g","oo","g","l","e"],["goog","l","e"]]
示例 2：

输入：s = "aab"
输出：[["a","a","b"],["aa","b"]]
示例 3：

输入：s = "a"
输出：[["a"]]

提示：

1 <= s.length <= 16
s 仅由小写英文字母组成

```python
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        res,path = [],[]
        def backTrace(index):
            if index > len(s):
                return
            if index == len(s):
                res.append(list(path))
            for i in range(index,len(s)):
                temp_a = s[index:i+1]
                temp_b = temp_a[::-1]
                if temp_a == temp_b:
                    path.append(temp_a)
                    backTrace(i+1)
                    path.pop()
        backTrace(0)
        return res
```

(理解)剑指 Offer II 087. 复原 IP

给定一个只包含数字的字符串 s ，用以表示一个 IP 地址，返回所有可能从 s 获得的 有效 IP 地址 。你可以按任何顺序返回答案。

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 0），整数之间用 '.' 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 有效 IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效 IP 地址。

示例 1：

输入：s = "25525511135"
输出：["255.255.11.135","255.255.111.35"]
示例 2：

输入：s = "0000"
输出：["0.0.0.0"]
示例 3：

输入：s = "1111"
输出：["1.1.1.1"]
示例 4：

输入：s = "010010"
输出：["0.10.0.10","0.100.1.0"]
示例 5：

输入：s = "10203040"
输出：["10.20.30.40","102.0.30.40","10.203.0.40"]

提示：

0 <= s.length <= 3000
s 仅由数字组成

```python
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        res, tem = [], []
        def dfs(index):
            if len(tem) > 4: return
            if len(tem) == 4 and index == len(s):
                res.append('.'.join(tem[:]))
                return
            for i in range(index, min(len(s), index + 4)): #范围最大为4
                cur = s[index : i + 1]
                if (len(cur) > 1 and cur[0] == '0') or int(cur) > 255:
                    continue
                tem.append(cur)
                dfs(i + 1)
                tem.pop()
        dfs(0)
        return res
```

剑指 Offer II 088. 爬楼梯的最少成本

数组的每个下标作为一个阶梯，第 i 个阶梯对应着一个非负数的体力花费值 cost[i]（下标从 0 开始）。

每当爬上一个阶梯都要花费对应的体力值，一旦支付了相应的体力值，就可以选择向上爬一个阶梯或者爬两个阶梯。

请找出达到楼层顶部的最低花费。在开始时，你可以选择从下标为 0 或 1 的元素作为初始阶梯。

示例 1：

输入：cost = [10, 15, 20]
输出：15
解释：最低花费是从 cost[1] 开始，然后走两步即可到阶梯顶，一共花费 15 。
 示例 2：

输入：cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]
输出：6
解释：最低花费方式是从 cost[0] 开始，逐个经过那些 1 ，跳过 cost[3] ，一共花费 6 。

提示：

2 <= cost.length <= 1000
0 <= cost[i] <= 999

```python
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        n = len(cost)
        dp = [0]*(n+1)
        dp[0], dp[1] = 0, 0
        for i in range(2,n+1):
            dp[i] = min(dp[i-1]+cost[i-1],dp[i-2]+cost[i-2])
        return dp[n]
```

剑指 Offer II 089. 房屋偷盗

一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响小偷偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。

给定一个代表每个房屋存放金额的非负整数数组 nums ，请计算 不触动警报装置的情况下 ，一夜之内能够偷窃到的最高金额。

示例 1：

输入：nums = [1,2,3,1]
输出：4
解释：偷窃 1 号房屋 (金额 = 1) ，然后偷窃 3 号房屋 (金额 = 3)。
      偷窃到的最高金额 = 1 + 3 = 4 。
示例 2：

输入：nums = [2,7,9,3,1]
输出：12
解释：偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋 (金额 = 1)。
      偷窃到的最高金额 = 2 + 9 + 1 = 12 。

提示：

1 <= nums.length <= 100
0 <= nums[i] <= 400

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 1: return nums[0]
        dp = [0] * n
        dp[0],dp[1]=nums[0],max(nums[0],nums[1])
        for i in range(2,n):
            dp[i] = max(dp[i-2]+nums[i],dp[i-1])
        return dp[n-1]
```

剑指 Offer II 090. 环形房屋偷盗

一个专业的小偷，计划偷窃一个环形街道上沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都围成一圈 ，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警 。

给定一个代表每个房屋存放金额的非负整数数组 nums ，请计算 在不触动警报装置的情况下 ，今晚能够偷窃到的最高金额。

示例 1：

输入：nums = [2,3,2]
输出：3
解释：你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2），因为他们是相邻的。
示例 2：

输入：nums = [1,2,3,1]
输出：4
解释：你可以先偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。
     偷窃到的最高金额 = 1 + 3 = 4 。
示例 3：

输入：nums = [0]
输出：0

提示：

1 <= nums.length <= 100
0 <= nums[i] <= 1000

```python
class Solution:
    def rob(self, nums: List[int]) -> int:
        if len(nums) <= 1:
            return nums[0]
        dp = [0, 0]
        for i in nums[1:]:
            dp.append(max(dp[-1], dp[-2] + i))
        dp_1 = [0, 0]
        for i in nums[:-1]:
            dp_1.append(max(dp_1[-1], dp_1[-2] + i))
        return max(dp[-1], dp_1[-1])
```

剑指 Offer II 091. 粉刷房子

假如有一排房子，共 n 个，每个房子可以被粉刷成红色、蓝色或者绿色这三种颜色中的一种，你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。

当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也是不同的。每个房子粉刷成不同颜色的花费是以一个 n x 3 的正整数矩阵 costs 来表示的。

例如，costs[0][0] 表示第 0 号房子粉刷成红色的成本花费；costs[1][2] 表示第 1 号房子粉刷成绿色的花费，以此类推。

请计算出粉刷完所有房子最少的花费成本。

示例 1：

```
输入: costs = [[17,2,17],[16,16,5],[14,3,19]]
输出: 10
解释: 将 0 号房子粉刷成蓝色，1 号房子粉刷成绿色，2 号房子粉刷成蓝色。
     最少花费: 2 + 5 + 3 = 10。
```

示例 2：

```
输入: costs = [[7,6,2]]
输出: 2
```

提示：

```
costs.length == n
costs[i].length == 3
1 <= n <= 100
1 <= costs[i][j] <= 20
```

```python
class Solution:
    def minCost(self, costs: List[List[int]]) -> int:
        # 第 i 间房子刷成 r, g, b 颜色的最小花费成本
        cost_r, cost_g, cost_b = 0, 0, 0
        for r, g, b in costs:
            cr = r + min(cost_g, cost_b)
            cg = g + min(cost_r, cost_b)
            cb = b + min(cost_r, cost_g)
            cost_r, cost_g, cost_b = cr, cg, cb
        return min(cost_r, cost_g, cost_b)
```

剑指 Offer II 092. 翻转字符

如果一个由 '0' 和 '1' 组成的字符串，是以一些 '0'（可能没有 '0'）后面跟着一些 '1'（也可能没有 '1'）的形式组成的，那么该字符串是 单调递增 的。

我们给出一个由字符 '0' 和 '1' 组成的字符串 s，我们可以将任何 '0' 翻转为 '1' 或者将 '1' 翻转为 '0'。

返回使 s 单调递增 的最小翻转次数。

示例 1：

输入：s = "00110"
输出：1
解释：我们翻转最后一位得到 00111.
示例 2：

输入：s = "010110"
输出：2
解释：我们翻转得到 011111，或者是 000111。
示例 3：

输入：s = "00011000"
输出：2
解释：我们翻转得到 00000000。

提示：

1 <= s.length <= 20000
s 中只包含字符 '0' 和 '1'

```python
class Solution:
    def minFlipsMonoIncr(self, s: str) -> int:
        dp0,dp1=0,0
        for i in s:
            curdp0,curdp1=dp0,min(dp0,dp1) #curdp1是在之前dp0和dp1中选择小的那一个
            if i=='0':
                curdp1+=1
            else:
                curdp0+=1
            dp0,dp1=curdp0,curdp1 #更新dp0和dp1

        return min(dp0,dp1)
```

剑指 Offer II 093. 最长斐波那契数列

如果序列 X_1, X_2, ..., X_n 满足下列条件，就说它是 斐波那契式 的：

n >= 3
对于所有 i + 2 <= n，都有 X_i + X_{i+1} = X_{i+2}
给定一个严格递增的正整数数组形成序列 arr ，找到 arr 中最长的斐波那契式的子序列的长度。如果一个不存在，返回 0 。

（回想一下，子序列是从原序列 arr 中派生出来的，它从 arr 中删掉任意数量的元素（也可以不删），而不改变其余元素的顺序。例如， [3, 5, 8] 是 [3, 4, 5, 6, 7, 8] 的一个子序列）

示例 1：

输入: arr = [1,2,3,4,5,6,7,8]
输出: 5
解释: 最长的斐波那契式子序列为 [1,2,3,5,8] 。
示例 2：

输入: arr = [1,3,7,11,12,14,18]
输出: 3
解释: 最长的斐波那契式子序列有 [1,11,12]、[3,11,14] 以及 [7,11,18] 。

提示：

3 <= arr.length <= 1000
1 <= arr[i] < arr[i + 1] <= 10^9

```python
class Solution:
    def lenLongestFibSubseq(self, arr: List[int]) -> int:
        # 定义一个二维的状态方程dp[i][j]
        dp=[[0 for _ in range(len(arr))] for _ in range(len(arr))]

        tmp={v:k for k,v in enumerate(arr)}
```

```
        ans=0

        for i in range(2,len(arr)):
            for j in range(i-1,-1,-1):
                if arr[i]-arr[j]>=arr[j]:
                    break
                if arr[i]-arr[j] in tmp:
                    k=tmp[arr[i]-arr[j]]
                    dp[i][j]=max(3,dp[j][k]+1)
                ans = max(ans,dp[i][j])

        return ans
```

(困难)剑指 Offer II 094. 最少回文分割

给定一个字符串 s，请将 s 分割成一些子串，使每个子串都是回文串。

返回符合要求的 最少分割次数 。

示例 1：

输入：s = "aab"
输出：1
解释：只需一次分割就可将 s 分割成 ["aa","b"] 这样两个回文子串。
示例 2：

输入：s = "a"
输出：0
示例 3：

输入：s = "ab"
输出：1

提示：

1 <= s.length <= 2000
s 仅由小写英文字母组成

```
class Solution:
    # 时间复杂度：O(n^2)，空间复杂度：O(n^2) 动态规划，单序列问题，自下往上
    def minCut(self, s: str) -> int:
        n = len(s)
        isPal = [[False] * n for _ in range(n)]  # 判断每个子字符串是不是回文
        for i in range(n):
            for j in range(i + 1):
                ch1 = s[i]
                ch2 = s[j]
                if ch1 == ch2 and (i <= j + 1 or isPal[j + 1][i - 1]):
                    isPal[j][i] = True
```

```
        dp = [0] * n  # 状态转移方程
        for i in range(n):
            if isPal[0][i]:
                dp[i] = 0
            else:
                dp[i] = i
                for j in range(1, i + 1):
                    if isPal[j][i]:
                        dp[i] = min(dp[i], dp[j - 1] + 1)
        return dp[n - 1]
```

剑指 Offer II 095. 最长公共子序列

给定两个字符串 text1 和 text2，返回这两个字符串的最长 公共子序列 的长度。如果不存在 公共子序列 ，返回 0 。

一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。

例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。
两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。

示例 1：

输入：text1 = "abcde", text2 = "ace"
输出：3
解释：最长公共子序列是 "ace" ，它的长度为 3 。
示例 2：

输入：text1 = "abc", text2 = "abc"
输出：3
解释：最长公共子序列是 "abc" ，它的长度为 3 。
示例 3：

输入：text1 = "abc", text2 = "def"
输出：0
解释：两个字符串没有公共子序列，返回 0 。

提示：

1 <= text1.length, text2.length <= 1000
text1 和 text2 仅由小写英文字符组成。

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        n1 = len(text1)
        n2 = len(text2)

        res = 0
```

```
        dp = [[0 for _ in range(n2 + 1)] for _ in range(n1 + 1)]
        for i in range(1, n1 + 1):
            for j in range(1, n2 + 1):
                if text1[i - 1] == text2[j - 1]:
                    dp[i][j] = max(dp[i][j], dp[i-1][j-1] + 1)
                    res = max(res, dp[i][j])
                else:
                    dp[i][j] = max(dp[i-1][j], dp[i][j-1])

        return res
```

剑指 Offer II 096. 字符串交织

给定三个字符串 s1、s2、s3，请判断 s3 能不能由 s1 和 s2 交织（交错） 组成。

两个字符串 s 和 t 交织 的定义与过程如下，其中每个字符串都会被分割成若干 非空 子字符串：

```
s = s1 + s2 + ... + sn
t = t1 + t2 + ... + tm
|n - m| <= 1
交织 是 s1 + t1 + s2 + t2 + s3 + t3 + ... 或者 t1 + s1 + t2 + s2 + t3 + s3 + ...
提示：a + b 意味着字符串 a 和 b 连接。
```

示例 1：

```
输入：s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
输出：true
```

示例 2：

```
输入：s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
输出：false
```

示例 3：

```
输入：s1 = "", s2 = "", s3 = ""
输出：true
```

提示：

```
0 <= s1.length, s2.length <= 100
0 <= s3.length <= 200
s1、s2、和 s3 都由小写英文字母组成
```

```
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
        n1 = len(s1)
        n2 = len(s2)
```

```
        n3 = len(s3)
        if n1 + n2 != n3:
            return False

        dp = [[False for _ in range(n2 + 1)] for _ in range(n1 + 1)]
        dp[0][0] = True
        for i in range(0, n1 + 1):
            for j in range(0, n2 + 1):
                if 0 <= i - 1 and s1[i - 1] == s3[i + j - 1]:
                    dp[i][j] |= dp[i-1][j]
                if 0 <= j - 1 and s2[j - 1] == s3[i + j - 1]:
                    dp[i][j] |= dp[i][j-1]

        return dp[n1][n2]
```

剑指 Offer II 097. 子序列的数目

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。

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）

题目数据保证答案符合 32 位带符号整数范围。

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。
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。
babgbag
babgbag
babgbag
babgbag
babgbag

提示：

0 <= s.length, t.length <= 1000
s 和 t 由英文字母组成

```python
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        INT_MAX = 2 ** 31 - 1

        sn = len(s)
        tn = len(t)

        if sn < tn:
            return 0

        dp = [[0 for _ in range(sn + 1)] for _ in range(tn + 1)]
        for si in range(sn + 1):
            dp[0][si] = 1
        for ti in range(1, tn + 1):
            for si in range(1, sn + 1):
                dp[ti][si] = dp[ti][si - 1]
                if s[si - 1] == t[ti - 1]:
                    if dp[ti][si] + dp[ti - 1][si - 1] <= INT_MAX:
                        dp[ti][si] += dp[ti - 1][si - 1]

        return dp[tn][sn]
```

剑指 Offer II 098. 路径的数目

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish” ）。

问总共有多少条不同的路径？

示例 1：

输入：m = 3, n = 7
输出：28
示例 2：

输入：m = 3, n = 2
输出：3
解释：
从左上角开始，总共有 3 条路径可以到达右下角。

1. 向右 -> 向下 -> 向下
2. 向下 -> 向下 -> 向右
3. 向下 -> 向右 -> 向下
   示例 3：

输入：m = 7, n = 3
输出：28
示例 4：

输入：m = 3, n = 3
输出：6

提示：

1 <= m, n <= 100
题目数据保证答案小于等于 2 * 109

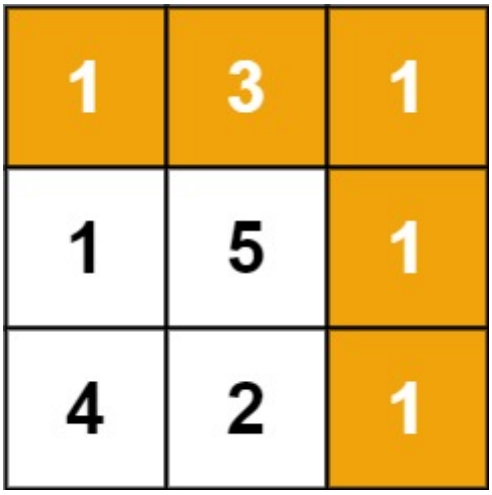
```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[1 for _ in range(n)] for _ in range(m)]
        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i-1][j]+dp[i][j-1]
        return dp[-1][-1]
```

剑指 Offer II 099. 最小路径之和

给定一个包含非负整数的 m x n 网格 grid ，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

说明：一个机器人每次只能向下或者向右移动一步。

示例 1：



输入：grid = [[1,3,1],[1,5,1],[4,2,1]]
输出：7

解释：因为路径 1→3→1→1→1 的总和最小。
示例 2：

输入：grid = [[1,2,3],[4,5,6]]
输出：12

提示：

m == grid.length
n == grid[i].length
1 <= m, n <= 200
0 <= grid[i][j] <= 100

```python
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        m, n = len(grid), len(grid[0])
        dp = [[float('inf')] * n for _ in range(m)] # 初始化为无穷是为了i=0/j=0这些格子索引到另一端不会取这些值
        for i in range(m):
            for j in range(n):
                if i == 0 and j == 0:
                    dp[i][j] = grid[0][0]
                else:
                    dp[i][j] = grid[i][j] + min(dp[i-1][j], dp[i][j-1])
        return dp[-1][-1]
```

剑指 Offer II 100. 三角形中最小路径之和

给定一个三角形 triangle ，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。相邻的结点 在这里指的是 下标 与 上一层结点下标 相同或者等于 上一层结点下标 + 1 的两个结点。也就是说，如果正位于当前行的下标 i ，那么下一步可以移动到下一行的下标 i 或 i + 1 。

示例 1：

输入：triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
输出：11
解释：如下面简图所示：
```
   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）。
示例 2：

输入：triangle = [[-10]]
输出：-10

提示：

1 <= triangle.length <= 200
triangle[0].length == 1
triangle[i].length == triangle[i - 1].length + 1
-104 <= triangle[i][j] <= 104

```python
class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        M = len(triangle)
        dp = [[float("inf")] * M for _ in range(M)]
        dp[0][0] = triangle[0][0]

        for r in range(1, M):
            dp[r][0] = dp[r-1][0] + triangle[r][0]
            for c in range(1, r):
                dp[r][c] = min(dp[r-1][c-1], dp[r-1][c]) + triangle[r][c]
            dp[r][r] = dp[r-1][r-1] + triangle[r][r]

        return min(dp[-1])
```

剑指 Offer II 101. 分割等和子集

给定一个非空的正整数数组 nums ，请判断能否将这些数字分成元素和相等的两部分。

示例 1：

输入：nums = [1,5,11,5]
输出：true
解释：nums 可以分割成 [1, 5, 5] 和 [11] 。
示例 2：

输入：nums = [1,2,3,5]
输出：false
解释：nums 不可以分为和相等的两部分

提示：

1 <= nums.length <= 200
1 <= nums[i] <= 100

```python
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        if sum(nums) % 2 != 0 : return False
        target = sum(nums) // 2
        dp = [False] * (target + 1)
        dp[0] = True
        for num in nums:
            for i in range(target, num - 1, -1):
                dp[i] = dp[i] | dp[i - num]
        return dp[target]
```

&：按位和，同一位上都为1时，才为1，否则为0

```
print(2 & 1)
# 0
```

因为1的二进制为01,2的二进制为10，所以2 & 1为0

|：按位或，同一位上至少有1个1时，则为1，否则为0

```
print(2 | 1)
# 3
```

因为1的二进制为01,2的二进制为10，所以2 | 1为3

^：按位异或，同一位上不同时为1，相同时为0
参加运算的两个对象，如果两个相应位为“异”(值不同)，则该位结果为1，否则为0。

```
print(3 ^ 1)
# 2
```

因为1的二进制为01,3的二进制为11，所以3 & 1为2

剑指 Offer II 102. 加减的目标值

给定一个正整数数组 nums 和一个整数 target 。

向数组中的每个整数前添加 '+' 或 '-' ，然后串联起所有整数，可以构造一个 表达式 ：

例如，nums = [2, 1] ，可以在 2 之前添加 '+' ，在 1 之前添加 '-' ，然后串联起来得到表达式 "+2-1" 。
返回可以通过上述方法构造的、运算结果等于 target 的不同 表达式 的数目。

示例 1：

输入：nums = [1,1,1,1,1], target = 3
输出：5
解释：一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3

+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
示例 2：

输入：nums = [1], target = 1
输出：1

提示：

1 <= nums.length <= 20
0 <= nums[i] <= 1000
0 <= sum(nums[i]) <= 1000
-1000 <= target <= 1000

```python
class Solution:
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        # x + y = sum
        # x - y = target
        # --> 2 * x = sum + target
        # 用01背包
        n = len(nums)

        tot_sum = sum(nums)
        if (tot_sum + target) % 2 == 1:
            return 0
        ttt = (tot_sum + target) // 2
        if ttt < 0:
            return 0

        #---- 01背包
        dp = [0 for _ in range(ttt + 1)]
        dp[0] = 1
        for i in range(n):
            y = nums[i]
            for t in range(ttt, y - 1, -1):
                x = t - y
                dp[t] += dp[x]

        return dp[ttt]
```

剑指 Offer II 103. 最少的硬币数目

给定不同面额的硬币 coins 和一个总金额 amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。

你可以认为每种硬币的数量是无限的。

示例 1：

输入：coins = [1, 2, 5], amount = 11
输出：3
解释：11 = 5 + 5 + 1
示例 2：

输入：coins = [2], amount = 3
输出：-1
示例 3：

输入：coins = [1], amount = 0
输出：0
示例 4：

输入：coins = [1], amount = 1
输出：1
示例 5：

输入：coins = [1], amount = 2
输出：2

提示：

1 <= coins.length <= 12
1 <= coins[i] <= 231 - 1
0 <= amount <= 104

```python
class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        if amount == 0: return 0
        coins.sort()
        if coins[0] > amount: return -1

        dp = [0] + [float('inf')] * amount # dp[i]: 硬币能组成金额为i所需最小个数
        for coin in coins: # 每次新拿一种硬币
            for i in range(coin, amount + 1): # 从大于等于这个硬币的面值开始，这个硬币才可能开始起作用
                dp[i] = min(dp[i], dp[i - coin] + 1)

        return dp[-1] if dp[-1] != float('inf') else -1
```

剑指 Offer II 104. 排列的数目

给定一个由 不同 正整数组成的数组 nums ，和一个目标整数 target 。请从 nums 中找出并返回总和为 target 的元素组合的个数。数组中的数字可以在一次排列中出现任意次，但是顺序不同的序列被视作不同的组合。

题目数据保证答案符合 32 位整数范围。

示例 1：

输入：nums = [1,2,3], target = 4
输出：7
解释：
所有可能的组合为：
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
请注意，顺序不同的序列被视作不同的组合。
示例 2：

输入：nums = [9], target = 3
输出：0

提示：

1 <= nums.length <= 200
1 <= nums[i] <= 1000
nums 中的所有元素 互不相同
1 <= target <= 1000

```python
class Solution:
    def combinationSum4(self, nums: List[int], target: int) -> int:
        dp = [0 for _ in range(target + 1)]
        dp[0] = 1
        for t in range(1, target + 1):
            for y in nums:
                x = t - y
                if 0 <= x:
                    dp[t] += dp[x]
        return dp[target]
```

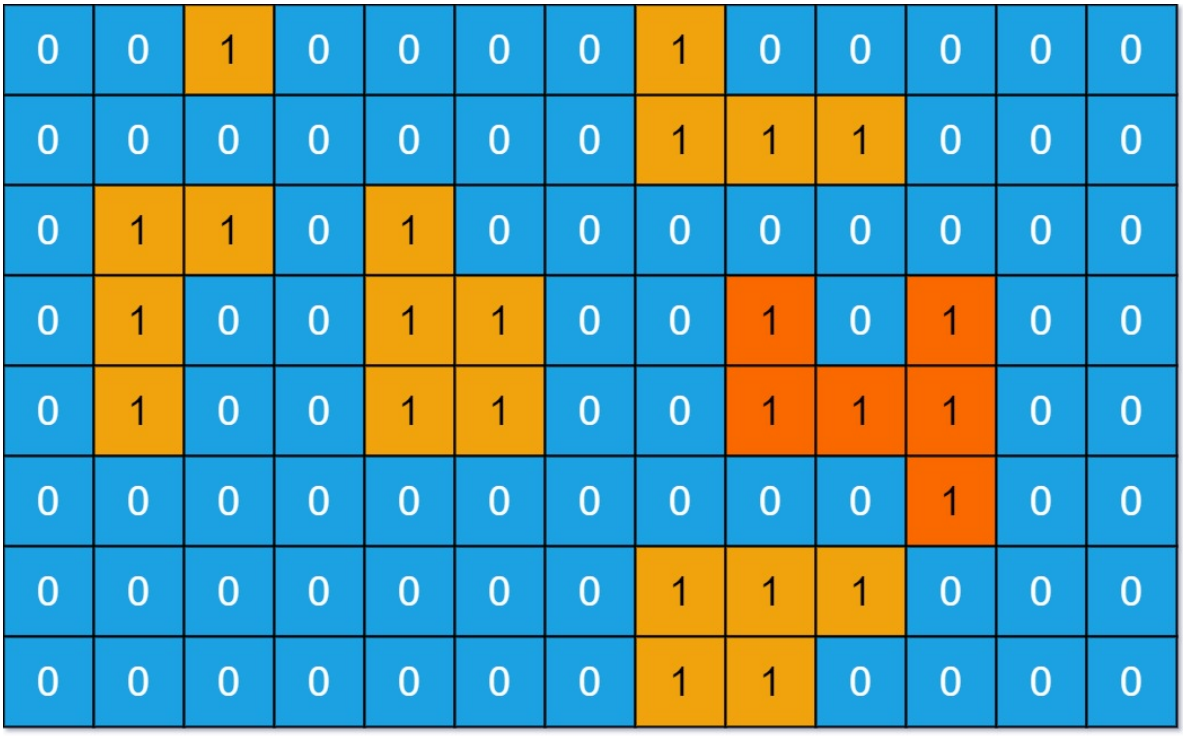
剑指 Offer II 105. 岛屿的最大面积

给定一个由 0 和 1 组成的非空二维数组 grid ，用来表示海洋岛屿地图。

一个 岛屿 是由一些相邻的 1 (代表土地) 构成的组合，这里的「相邻」要求两个 1 必须在水平或者竖直方向上相邻。你可以假设 grid 的四个边缘都被 0（代表水）包围着。

找到给定的二维数组中最大的岛屿面积。如果没有岛屿，则返回面积为 0 。

示例 1:



输入: grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0,0],
[0,1,0,0,1,1,0,0,1,1,1,0,0],[0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]
输出: 6
解释: 对于上面这个给定矩阵应返回 6。注意答案不应该是 11 ，因为岛屿只能包含水平或垂直的四个方向的 1 。
示例 2:

输入: grid = [[0,0,0,0,0,0,0,0]]
输出: 0

提示:

m == grid.length
n == grid[i].length
1 <= m, n <= 50
grid[i][j] is either 0 or 1

```python
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        m,n = len(grid),len(grid[0])
        max_num = 0
        now_num = [0]
        def count_num(i,j):
            grid[i][j] = 0
            now_num[-1] += 1
            if 0 <= i-1 < m and grid[i-1][j] == 1:
                count_num(i-1,j)

            if 0 <= i+1 < m and grid[i+1][j] == 1:

                count_num(i+1,j)
```

```
            if 0 <= j-1 < n and grid[i][j-1] == 1:

                count_num(i,j-1)
            if 0 <= j+1 < n and grid[i][j+1] == 1:

                count_num(i,j+1)
            return

        for i in range(0,m):
            for j in range(0,n):
                if grid[i][j] == 1:
                    now_num = [0]
                    count_num(i,j)
                    max_num = max(max_num,now_num[-1])
        return max_num
```

剑指 Offer II 106. 二分图

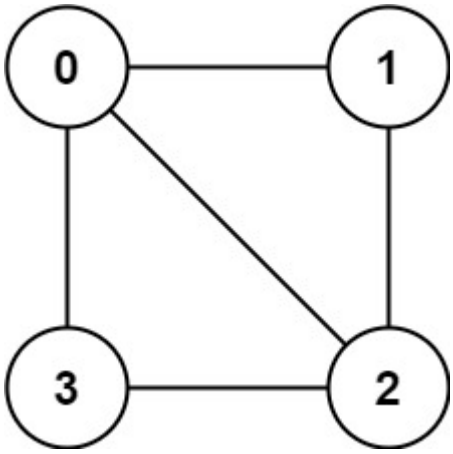
存在一个 无向图 ，图中有 n 个节点。其中每个节点都有一个介于 0 到 n - 1 之间的唯一编号。

给定一个二维数组 graph ，表示图，其中 graph[u] 是一个节点数组，由节点 u 的邻接节点组成。形式上，对于 graph[u] 中的每个 v ，都存在一条位于节点 u 和节点 v 之间的无向边。该无向图同时具有以下属性：

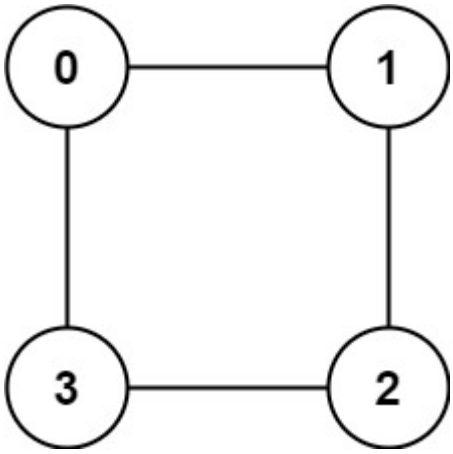
不存在自环（graph[u] 不包含 u）。
不存在平行边（graph[u] 不包含重复值）。
如果 v 在 graph[u] 内，那么 u 也应该在 graph[v] 内（该图是无向图）
这个图可能不是连通图，也就是说两个节点 u 和 v 之间可能不存在一条连通彼此的路径。
二分图 定义：如果能将一个图的节点集合分割成两个独立的子集 A 和 B ，并使图中的每一条边的两个节点一个来自 A 集合，一个来自 B 集合，就将这个图称为 二分图 。

如果图是二分图，返回 true ；否则，返回 false 。

示例 1：



输入：graph = [[1,2,3],[0,2],[0,1,3],[0,2]]
输出：false
解释：不能将节点分割成两个独立的子集，以使每条边都连通一个子集中的一个节点与另一个子集中的一个节点。
示例 2：



输入：graph = [[1,3],[0,2],[1,3],[0,2]]
输出：true
解释：可以将节点分成两组：{0, 2} 和 {1, 3} 。

提示：

graph.length == n
1 <= n <= 100
0 <= graph[u].length < n
0 <= graph[u][i] <= n - 1
graph[u] 不会包含 u
graph[u] 的所有值 互不相同
如果 graph[u] 包含 v，那么 graph[v] 也会包含 u

```python
class Solution:
    def isBipartite(self, graph: List[List[int]]) -> bool:
        def dfs(x: int, cur_color) -> bool:
            color[x] = cur_color
            for y in graph[x]:
                if color[y] == color[x]:
                    return False
                elif color[y] == 1 - color[x]:
                    continue
                else:
                    if dfs(y, 1 - cur_color) == False:
                        return False
            return True


        n = len(graph)
        color = [-1 for _ in range(n)]
        RED = 0
        BLUE = 1
```

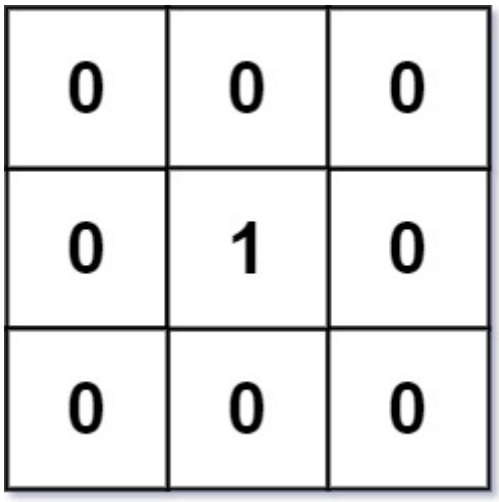
```
        for x in range(n):
            if color[x] == -1:
                if dfs(x, RED) == False:
                    return False
        return True
```

剑指 Offer II 107. 矩阵中的距离剑指 Offer II 107. 矩阵中的距离

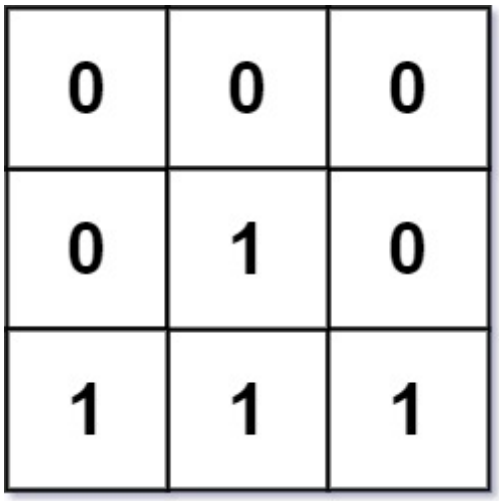
给定一个由 0 和 1 组成的矩阵 mat ，请输出一个大小相同的矩阵，其中每一个格子是 mat 中对应位置元素到最近的 0 的距离。

两个相邻元素间的距离为 1 。

示例 1：



输入：mat = [[0,0,0],[0,1,0],[0,0,0]]
输出：[[0,0,0],[0,1,0],[0,0,0]]
示例 2：



输入：mat = [[0,0,0],[0,1,0],[1,1,1]]

输出：[[0,0,0],[0,1,0],[1,2,1]]

提示：

m == mat.length
n == mat[i].length
1 <= m, n <= 104
1 <= m * n <= 104
mat[i][j] is either 0 or 1.
mat 中至少有一个 0

```
class Solution:
    def updateMatrix(self, mat: List[List[int]]) -> List[List[int]]:
        """
            题目分析：
                改题可分为3种情况
                1. 如果该位置是0，则为不变。
                2. 如果该位置的四周(上下左右)存在0，则该位置置1.
                3. 如果该位置被1包裹，则查询4周的1到0的距离，该位置的值则为4周离0最近的距离+1

            题目做法：
                1. 广度优先遍历(bfs)
                    1.1 创建一个新的二维数组dists，用于存放每一个位置离0的距离
                    1.2 记录值为0的位置，并把该位置放进广度优先队列queue，同时，在dists上，把值为0的位置的值置0，值为1的位置置为无穷大
                    1.3 对于队列里的每一个元素(一开始都是0所处的位置)，把每一个元素的4个方向与0的距离都算出来（如果某个位置的值也为0，则不进行任何操作，如果为1，则dists该位置的距离为1，并把该位置添加进队列里）
                    1.4 重复1.3的操作，以0为起点一直走，直到队列里面没有元素，所有的位置都走完了
        """
        m, n = len(mat), len(mat[0])
        # 初始化
        dists = [[[0] for i in range(n)] for j in range(m)]
        queue = []
        for i in range(m):
            for j in range(n):
                if mat[i][j] == 0:
                    queue.append((i, j))
                    dists[i][j] = 0
                else:
                    dists[i][j] = float('inf')

        dirs = [(-1, 0), (1, 0), (0, -1), (0, 1)] # 上/下/左/右
        while queue:
            pos = queue.pop(0) # 拿出队头
            for d in dirs:
                row, col = pos[0] + d[0], pos[1] + d[1]
                if row < 0 or row >=m or col <0 or col >= n: continue
                else:
                    if dists[pos[0]][pos[1]] + 1 >= dists[row][col]: continue
```

```
                    else:
                        dists[row][col] = dists[pos[0]][pos[1]] + 1
                        queue.append((row, col))

        return dists
```

剑指 Offer II 108. 单词演变

在字典（单词列表） wordList 中，从单词 beginWord 和 endWord 的 转换序列 是一个按下述规格形成的序列：

序列中第一个单词是 beginWord 。
序列中最后一个单词是 endWord 。
每次转换只能改变一个字母。
转换过程中的中间单词必须是字典 wordList 中的单词。
给定两个长度相同但内容不同的单词 beginWord 和 endWord 和一个字典 wordList ，找到
从 beginWord 到 endWord 的 最短转换序列 中的 单词数目 。如果不存在这样的转换序列，返回 0。

示例 1：

输入：beginWord = "hit", endWord = "cog", wordList =
["hot","dot","dog","lot","log","cog"]
输出：5
解释：一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog", 返回它的长度 5。
示例 2：

输入：beginWord = "hit", endWord = "cog", wordList =
["hot","dot","dog","lot","log"]
输出：0
解释：endWord "cog" 不在字典中，所以无法进行转换。

提示：

1 <= beginWord.length <= 10
endWord.length == beginWord.length
1 <= wordList.length <= 5000
wordList[i].length == beginWord.length
beginWord、endWord 和 wordList[i] 由小写英文字母组成
beginWord != endWord
wordList 中的所有字符串 互不相同

```
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) ->
int:
        if endWord not in wordList: return 0

        word_id = dict()  # 存储每个单词及其虚拟单词（简称节点）的id，key为单词或虚拟单词
（即节点），value为id值
```

```
        edge = collections.defaultdict(list)  # 存储与每个节点 相连接 的其他节点， key为源节点的id， value为目标节点的id（可有多个目标节点）
        nodeNum = 0  # 节点 id，初始化为0

        # 图构建
        for word in [beginWord] + wordList:
            # 单词赋予id
            word_id[word] = nodeNum
            id1 = nodeNum
            nodeNum += 1
            # 虚拟单词赋予id，并构建边
            chars = list(word)
            for i in range(len(chars)):
                tmp = chars[i]
                chars[i] = "*"
                new_word = "".join(chars)
                # 单词的连接 就是通过具有相同的虚拟单词来间接连接
                if new_word not in word_id:
                    word_id[new_word] = nodeNum
                    nodeNum += 1
                id2 = word_id[new_word]
                edge[id1].append(id2)
                edge[id2].append(id1)
                chars[i] = tmp

        begin_id, end_id = word_id[beginWord], word_id[endWord]
        dis = [float("inf")] * nodeNum  # dis[i] 表示从起始单词转换到 id 为i的单词 所需的转换次数
        dis[begin_id] = 0

        # 广度优先搜索
        que = collections.deque([begin_id])
        while que:
            x = que.popleft()
            if x == end_id:
                return dis[x] // 2 +1
            for it in edge[x]: # 遍历x所连的所有边
                if dis[it] == float("inf"):  # 因为是双向边，不加限制的话会逆回去遍历，这里就限定了没遍历过的边
                    dis[it] = dis[x] + 1
                    que.append(it)

        return 0
```

剑指 Offer II 109. 开密码锁

一个密码锁由 4 个环形拨轮组成，每个拨轮都有 10 个数字： '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' 。每个拨轮可以自由旋转：例如把 '9' 变为 '0'，'0' 变为 '9' 。每次旋转都只能旋转一个拨轮的一位数字。

锁的初始数字为 '0000' ，一个代表四个拨轮的数字的字符串。

列表 deadends 包含了一组死亡数字，一旦拨轮的数字和列表里的任何一个元素相同，这个锁将会被永久锁定，无法再被旋转。

字符串 target 代表可以解锁的数字，请给出解锁需要的最小旋转次数，如果无论如何不能解锁，返回 -1 。

示例 1:

输入：deadends = ["0201","0101","0102","1212","2002"], target = "0202"
输出：6
解释：
可能的移动序列为 "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202"。
注意 "0000" -> "0001" -> "0002" -> "0102" -> "0202" 这样的序列是不能解锁的，因为当拨动到 "0102" 时这个锁就会被锁定。
示例 2:

输入: deadends = ["8888"], target = "0009"
输出：1
解释：
把最后一位反向旋转一次即可 "0000" -> "0009"。
示例 3:

输入: deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"], target = "8888"
输出：-1
解释：
无法旋转到目标数字且不被锁定。
示例 4:

输入: deadends = ["0000"], target = "8888"
输出：-1

提示：

1 <= deadends.length <= 500
deadends[i].length == 4
target.length == 4
target 不在 deadends 之中
target 和 deadends[i] 仅由若干位数字组成

```python
class Solution:
    def openLock(self, deadends: List[str], target: str) -> int:
        deadSet = set()
        for deadend in deadends:
            deadSet.add(deadend)
        visitied = set()
        from queue import Queue
        q = Queue()
        q.put('0000')
        visitied.add('0000')
        step = 0
        while(not q.empty()):
```

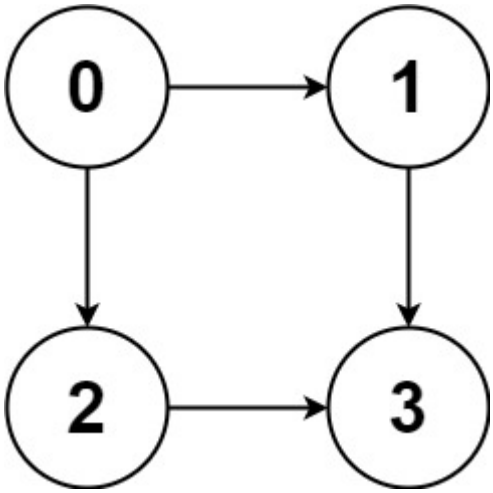
```
            for i in range(q.qsize()):
                cur = q.get()
                if(cur in deadSet):continue
                if (cur == target):return step
                for i in range(4):
                    up = self.upOne(cur,i)
                    if(up not in visitied):
                        q.put(up)
                        visitied.add(up)
                    down = self.downOne(cur,i)
                    if(down not in visitied):
                        q.put(down)
                        visitied.add(down)
            step +=1
        return -1

    def upOne(self,s,i):
        # 上拨s的第i位数字
        l = list(s)
        if l[i] == '9':
            l[i] = '0'
        else:
            l[i] = str(int(l[i])+1)
        return ''.join(l)

    def downOne(self,s,i):
        # 下拨s的第i位数字
        l = list(s)
        if l[i] == '0':
            l[i] = '9'
        else:
            l[i] = str(int(l[i])-1)
        return ''.join(l)
```

剑指 Offer II 110. 所有路径

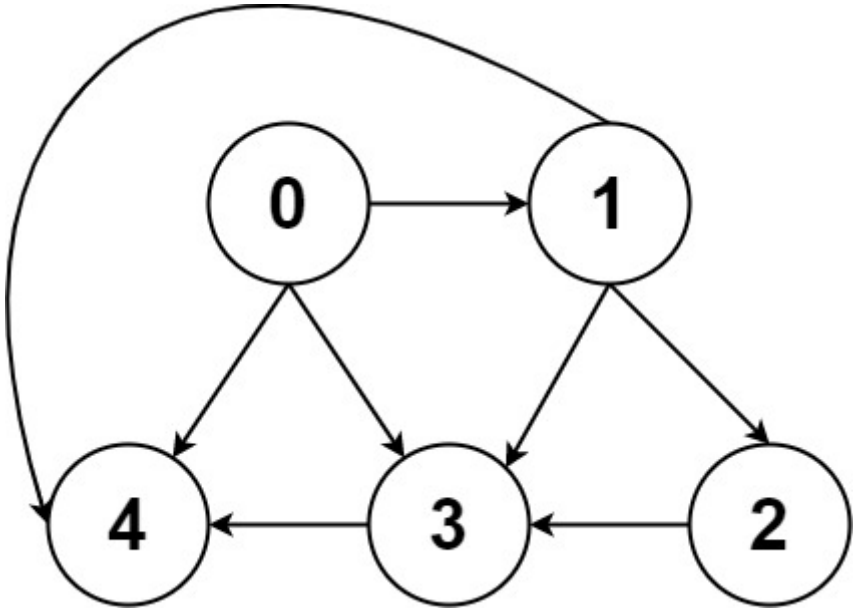
给定一个有 n 个节点的有向无环图，用二维数组 graph 表示，请找到所有从 0 到 n-1 的路径并输出（不要求按顺序）。

graph 的第 i 个数组中的单元都表示有向图中 i 号节点所能到达的下一些结点（译者注：有向图是有方向的，即规定了 a→b 你就不能从 b→a ），若为空，就是没有下一个节点了。

示例 1：



输入：graph = [[1,2],[3],[3],[]]
输出：[[0,1,3],[0,2,3]]
解释：有两条路径 0 -> 1 -> 3 和 0 -> 2 -> 3
示例 2：



输入：graph = [[4,3,1],[3,2,4],[3],[4],[]]
输出：[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]
示例 3：

输入：graph = [[1],[]]
输出：[[0,1]]
示例 4：

输入：graph = [[1,2,3],[2],[3],[]]
输出：[[0,1,2,3],[0,2,3],[0,3]]
示例 5：

输入：graph = [[1,3],[2],[3],[]]
输出：[[0,1,2,3],[0,3]]

提示：

n == graph.length

2 <= n <= 15
0 <= graph[i][j] < n
graph[i][j] != i
保证输入为有向无环图 (GAD)

```python
class Solution:
    def allPathsSourceTarget(self, graph: List[List[int]]) -> List[List[int]]:

        def DFS(cur_path):
            if cur_path[-1] == n - 1:
                ans.append(cur_path[:])
            else:
                cur_node = cur_path[-1]
                for child in graph[cur_node]:
                    if not visited[child]:
                        visited[child] = True
                        DFS(cur_path + [child])
                        visited[child] = False

        n, ans = len(graph), []
        visited = [False] * n
        DFS([0])
        return ans
```

剑指 Offer II 111. 计算除法

给定一个变量对数组 equations 和一个实数值数组 values 作为已知条件，其中 equations[i] = [Ai, Bi] 和 values[i] 共同表示等式 Ai / Bi = values[i] 。每个 Ai 或 Bi 是一个表示单个变量的字符串。

另有一些以数组 queries 表示的问题，其中 queries[j] = [Cj, Dj] 表示第 j 个问题，请你根据已知条件找出 Cj / Dj = ? 的结果作为答案。

返回 所有问题的答案 。如果存在某个无法确定的答案，则用 -1.0 替代这个答案。如果问题中出现了给定的已知条件中没有出现的字符串，也需要用 -1.0 替代这个答案。

注意：输入总是有效的。可以假设除法运算中不会出现除数为 0 的情况，且不存在任何矛盾的结果。

示例 1：

输入：equations = [["a","b"],["b","c"]], values = [2.0,3.0], queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]
输出：[6.00000,0.50000,-1.00000,1.00000,-1.00000]
解释：
条件：a / b = 2.0, b / c = 3.0
问题：a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
结果：[6.0, 0.5, -1.0, 1.0, -1.0]
示例 2：

输入：equations = [["a","b"],["b","c"],["bc","cd"]], values = [1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]

输出：[3.75000,0.40000,5.00000,0.20000]
示例 3：

输入：equations = [["a","b"]], values = [0.5], queries = [["a","b"],["b","a"],["a","c"],["x","y"]]
输出：[0.50000,2.00000,-1.00000,-1.00000]

提示：

1 <= equations.length <= 20
equations[i].length == 2
1 <= Ai.length, Bi.length <= 5
values.length == equations.length
0.0 < values[i] <= 20.0
1 <= queries.length <= 20
queries[i].length == 2
1 <= Cj.length, Dj.length <= 5
Ai, Bi, Cj, Dj 由小写英文字母与数字组成

```python
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) -> List[float]:
        n = len(equations)

        #---- 给word编号。
        ID = 0
        word_ID = dict()
        for x, y in equations:
            if x not in word_ID:
                word_ID[x] = ID
                ID += 1
            if y not in word_ID:
                word_ID[y] = ID
                ID += 1

        #---- 邻接矩阵
        edge = [[-1.0 for _ in range(ID)] for _ in range(ID)]
        for [x, y], val in zip(equations, values):
            xID, yID = word_ID[x], word_ID[y]
            edge[xID][yID] = val
            edge[yID][xID] = 1.0 / val
        #---- floyd算法
        for mid in range(ID):
            for x in range(ID):
                for y in range(ID):
                    if edge[x][mid] > 0 and edge[mid][y] > 0:
                        edge[x][y] = edge[x][mid] * edge[mid][y]

        res = []
        for x, y in queries:
            if x in word_ID and y in word_ID:
                xID = word_ID[x]
                yID = word_ID[y]
```

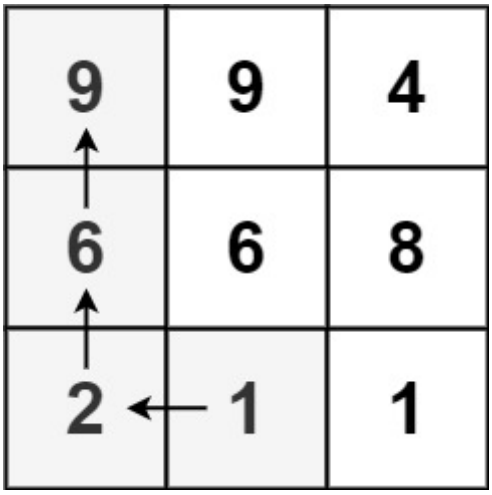
```
                res.append(edge[xID][yID])
            else:
                res.append(-1.0)
        return res
```

剑指 Offer II 112. 最长递增路径

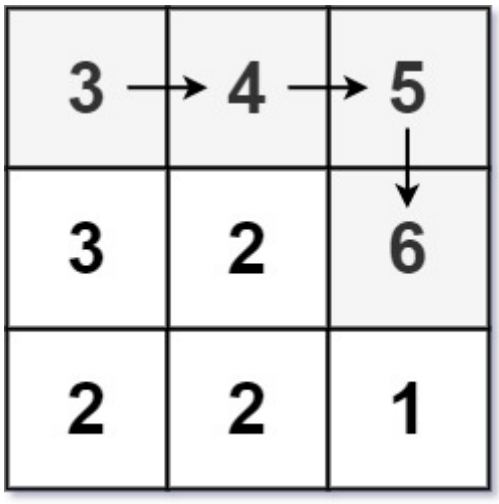
给定一个 m x n 整数矩阵 matrix ，找出其中 最长递增路径 的长度。

对于每个单元格，你可以往上，下，左，右四个方向移动。 不能 在 对角线 方向上移动或移动到 边界外（即不允许环绕）。

示例 1：



输入：matrix = [[9,9,4],[6,6,8],[2,1,1]]
输出：4
解释：最长递增路径为 [1, 2, 6, 9]。
示例 2：



输入：matrix = [[3,4,5],[3,2,6],[2,2,1]]
输出：4
解释：最长递增路径是 [3, 4, 5, 6]。注意不允许在对角线方向上移动。
示例 3：

```
输入：matrix = [[1]]
输出：1


提示：

m == matrix.length
n == matrix[i].length
1 <= m, n <= 200
0 <= matrix[i][j] <= 231 - 1
```

```python
class Solution:
    def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
        # 思路：DFS：对于每个节点执行一次DFS搜索
        # dp[i][j] 记录当前节点的最大递增路径值 兼 visited
        m, n = len(matrix), len(matrix[0])
        dp = [[0 for _ in range(n)] for _ in range(m)]

        maxLen = 1
        for i in range(m):
            for j in range(n):
                if dp[i][j] == 0:
                    dp[i][j] = self.dfs(i, j, matrix, dp)
                    maxLen = max(dp[i][j], maxLen)
        # print(dp)
        return maxLen

    def dfs(self, i, j, matrix, visited):
        # 返回当前节点的深度
        # 如果当前节点已经计算过，则直接返回
        if visited[i][j]:
            return visited[i][j]

        m, n = len(matrix), len(matrix[0])

        maxDepth = 1    # 当前节点的最大深度
        for (u, v) in [(i+1, j), (i-1, j), (i, j+1), (i, j-1)]:
            if u >= 0 and u < m and v >= 0 and v < n:
                if matrix[u][v] > matrix[i][j]:
                    depth = self.dfs(u, v, matrix, visited) + 1
                    maxDepth = max(maxDepth, depth)

        visited[i][j] = maxDepth
        return maxDepth
```

剑指 Offer II 113. 课程顺序

```
现在总共有 numCourses 门课需要选，记为 0 到 numCourses-1。
```

给定一个数组 prerequisites ，它的每一个元素 prerequisites[i] 表示两门课程之间的先修顺序。 例如 prerequisites[i] = [ai, bi] 表示想要学习课程 ai ，需要先完成课程 bi 。

请根据给出的总课程数 numCourses 和表示先修顺序的 prerequisites 得出一个可行的修课序列。

可能会有多个正确的顺序，只要任意返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

示例 1:

输入: numCourses = 2, prerequisites = [[1,0]]
输出: [0,1]
解释: 总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1] 。
示例 2:

输入: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
输出: [0,1,2,3] or [0,2,1,3]
解释: 总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。
 因此，一个正确的课程顺序是 [0,1,2,3] 。另一个正确的排序是 [0,2,1,3] 。
示例 3:

输入: numCourses = 1, prerequisites = []
输出: [0]
解释: 总共 1 门课，直接修第一门课就可。

```python
class Solution(object):
    def findOrder(self, numCourses, prerequisites):
        R = [0] * numCourses
        map = [[] for i in range(numCourses)]
        study = []
        que = []
        # 数据处理
        for i in range(len(prerequisites)):
            R[prerequisites[i][0]] += 1
            map[prerequisites[i][1]].append(prerequisites[i][0])
            # 查找
        for i in range(len(R)):
            if R[i] == 0:
                study.append(i)

        while (len(study)):
            learn = study[0]
            del study[0]
            que.append(learn)
            for i in map[learn]:
                R[i]-=1
                if R[i] == 0:
                    study.append(i)
        if len(que) != numCourses:
            que = []
        return que
```

剑指 Offer II 114. 外星文字典

现有一种使用英语字母的外星文语言，这门语言的字母顺序与英语顺序不同。

给定一个字符串列表 words ，作为这门语言的词典，words 中的字符串已经 按这门新语言的字母顺序进行了排序 。

请你根据该词典还原出此语言中已知的字母顺序，并 按字母递增顺序 排列。若不存在合法字母顺序，返回 "" 。若存在多种可能的合法字母顺序，返回其中 任意一种 顺序即可。

字符串 s 字典顺序小于 字符串 t 有两种情况：

在第一个不同字母处，如果 s 中的字母在这门外星语言的字母顺序中位于 t 中字母之前，那么 s 的字典顺序小于 t 。
如果前面 min(s.length, t.length) 字母都相同，那么 s.length < t.length 时，s 的字典顺序也小于 t 。

示例 1：

输入：words = ["wrt","wrf","er","ett","rftt"]
输出："wertf"
示例 2：

输入：words = ["z","x"]
输出："zx"
示例 3：

输入：words = ["z","x","z"]
输出：""
解释：不存在合法字母顺序，因此返回 "" 。

提示：

1 <= words.length <= 100
1 <= words[i].length <= 100
words[i] 仅由小写英文字母组成

```python
# 这题因为做了113题 然后会联想到 这是不是也是一个拓扑排序的问题
# 一个字母比另一个字母大  则代表一个有向连线
# 如果该图是有项无环图 则说明结果成立
# 否则 则说明不正确

# 做法
# 遍历列表 两两相比字符串 将第一个不同的比较顺序 存入到边集中    这里有个注意的地方 就是 abc 《abcd  abc要排在前面 这种情况要在遍历里看看成不成立 不成立直接返回
import collections


class Solution:
    def alienOrder(self, words: List[str]) -> str:
        # 只有一个字符串的 直接返回这个字符串就好了 任何顺序都行得通
        if len(words) == 1:
```

```
        return words[0]
    # 把所有边的关系 都放进set中
    edgs_set = set()
    # 记录每个点的出度边信息  用上新的数据结构
    edgs = collections.defaultdict(list)

    word_dict = {}

    # 遍历所有单词 把所有字母都加到字典中 初始化入度为0
    for i in words:
        for j in i:
            word_dict[j] = 0

    # 初始化边集 和 出度、入度情况 两两比较
    for i in range(len(words) - 1):
        # 需要把第一个单词的所有字符遍历完整 防止 abcd 在abc前面 这种特殊情况
        word1 = words[i]
        word2 = words[i + 1]
        # flag = 0 代表还未比对出不同
        flag = 0
        for j in range(len(word1)):
            if flag == 0:
                # 防止 abcd 在abc前面 这种特殊情况
                if j >= len(word2):
                    return ""
                else:
                    # 找出第一个不同的字母  需要第一个字母开始对比
                    if word1[j] != word2[j]:
                        if (word1[j], word2[j]) not in edgs_set:
                            edgs_set.add((word1[j], word2[j]))
                            edgs[word1[j]].append(word2[j])
                            # 入度+1
                            word_dict[word2[j]] += 1
                        flag = 1

            else:
                break

    queue = collections.deque()
    for key, value in word_dict.items():
        if value == 0:
            queue.append(key)

    result = ""
    while queue:
        point = queue.popleft()
        result += point
        for i in edgs[point]:
            # 删掉边 指向的点入度-1
            word_dict[i] -= 1
            if word_dict[i] == 0:
                queue.append(i)

    # 结果的长度等于所有字符长度 说明没有换 有解 否则无解
    if len(result) != len(word_dict):
```

```
            # 存在环
            return ""
        else:
            return result
```

剑指 Offer II 115. 重建序列

给定一个长度为 n 的整数数组 nums ，其中 nums 是范围为 [1，n] 的整数的排列。还提供了一个 2D 整数数组 sequences ，其中 sequences[i] 是 nums 的子序列。
检查 nums 是否是唯一的最短 超序列 。最短 超序列 是 长度最短 的序列，并且所有序列 sequences[i] 都是它的子序列。对于给定的数组 sequences ，可能存在多个有效的 超序列 。

例如，对于 sequences = [[1,2],[1,3]] ，有两个最短的 超序列 ，[1,2,3] 和 [1,3,2] 。
而对于 sequences = [[1,2],[1,3],[1,2,3]] ，唯一可能的最短 超序列 是 [1,2,3] 。
[1,2,3,4] 是可能的超序列，但不是最短的。
如果 nums 是序列的唯一最短 超序列 ，则返回 true ，否则返回 false 。
子序列 是一个可以通过从另一个序列中删除一些元素或不删除任何元素，而不改变其余元素的顺序的序列。

示例 1：

输入：nums = [1,2,3], sequences = [[1,2],[1,3]]
输出：false
解释：有两种可能的超序列：[1,2,3]和[1,3,2]。
序列 [1,2] 是[1,2,3]和[1,3,2]的子序列。
序列 [1,3] 是[1,2,3]和[1,3,2]的子序列。
因为 nums 不是唯一最短的超序列，所以返回false。
示例 2：

输入：nums = [1,2,3], sequences = [[1,2]]
输出：false
解释：最短可能的超序列为 [1,2]。
序列 [1,2] 是它的子序列：[1,2]。
因为 nums 不是最短的超序列，所以返回false。
示例 3：

输入：nums = [1,2,3], sequences = [[1,2],[1,3],[2,3]]
输出：true
解释：最短可能的超序列为[1,2,3]。
序列 [1,2] 是它的一个子序列：[1,2,3]。
序列 [1,3] 是它的一个子序列：[1,2,3]。
序列 [2,3] 是它的一个子序列：[1,2,3]。
因为 nums 是唯一最短的超序列，所以返回true。

提示：

n == nums.length
1 <= n <= 104
nums 是 [1, n] 范围内所有整数的排列
1 <= sequences.length <= 104
1 <= sequences[i].length <= 104
1 <= sum(sequences[i].length) <= 105

1 <= sequences[i][j] <= n
sequences 的所有数组都是 唯一 的
sequences[i] 是 nums 的一个子序列

```
class Solution:
    def sequenceReconstruction(self, nums: List[int], sequences:
List[List[int]]) -> bool:

        d  = {}
        for seq in sequences:
            for i in range(len(seq)-1):
                if seq[i] not in d:
                    d[seq[i]] = set()
                d[seq[i]].add(seq[i+1])
        for i in range(len(nums)-1):
            if nums[i] not in d or nums[i+1] not in d[nums[i]]:
                return False
        return True
```

剑指 Offer II 116. 省份数量

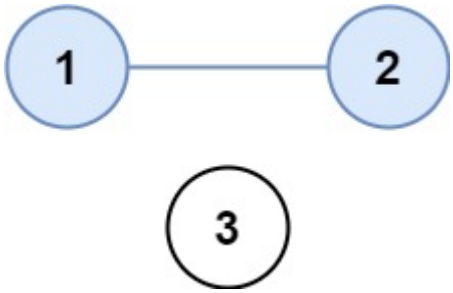
有 n 个城市，其中一些彼此相连，另一些没有相连。如果城市 a 与城市 b 直接相连，且城市 b 与城市 c 直接相连，那么城市 a 与城市 c 间接相连。

省份 是一组直接或间接相连的城市，组内不含其他没有相连的城市。

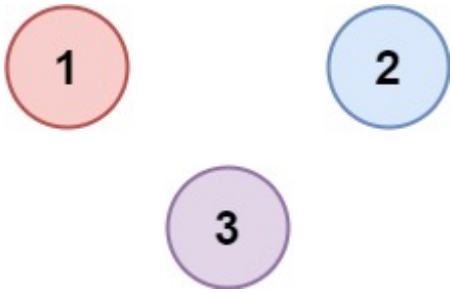
给你一个 n x n 的矩阵 isConnected ，其中 isConnected[i][j] = 1 表示第 i 个城市和第 j 个城市直接相连，而 isConnected[i][j] = 0 表示二者不直接相连。

返回矩阵中 省份 的数量。

示例 1：



输入：isConnected = [[1,1,0],[1,1,0],[0,0,1]]
输出：2
示例 2：



输入：isConnected = [[1,0,0],[0,1,0],[0,0,1]]
输出：3

提示：

1 <= n <= 200
n == isConnected.length
n == isConnected[i].length
isConnected[i][j] 为 1 或 0
isConnected[i][i] == 1
isConnected[i][j] == isConnected[j][i]

```python
class Solution:
    def findCircleNum(self, isConnected: List[List[int]]) -> int:
        #进行dfs寻找相连城市
        def dfs(i,matrix):
            if i in viewed:
                return
            else:
                viewed.add(i)
            for j in range(len(matrix[0])):
                if matrix[i][j] == 1: dfs(j,matrix)
            return
        #初始化已经遍历过的城市
        viewed = set()
        ans = 0
        #进行城市的遍历
        for i in range(len(isConnected)):
            #还未遍历过，进行ans+1同时dfs
            if i not in viewed:
                ans +=1
                dfs(i,isConnected)
        return ans
```

剑指 Offer II 117. 相似的字符串

如果交换字符串 X 中的两个不同位置的字母，使得它和字符串 Y 相等，那么称 X 和 Y 两个字符串相似。如果这两个字符串本身是相等的，那它们也是相似的。

例如，"tars" 和 "rats" 是相似的 (交换 0 与 2 的位置)； "rats" 和 "arts" 也是相似的，但是 "star" 不与 "tars"，"rats"，或 "arts" 相似。

总之，它们通过相似性形成了两个关联组：{"tars", "rats", "arts"} 和 {"star"}。注意，"tars" 和 "arts" 是在同一组中，即使它们并不相似。形式上，对每个组而言，要确定一个单词在组中，只需要这个词和该组中至少一个单词相似。

给定一个字符串列表 strs。列表中的每个字符串都是 strs 中其它所有字符串的一个 字母异位词 。请问 strs 中有多少个相似字符串组？

字母异位词（anagram），一种把某个字符串的字母的位置（顺序）加以改换所形成的新词。

示例 1：

输入：strs = ["tars","rats","arts","star"]
输出：2
示例 2：

输入：strs = ["omv","ovm"]
输出：1

提示：

1 <= strs.length <= 300
1 <= strs[i].length <= 300
strs[i] 只包含小写字母。
strs 中的所有单词都具有相同的长度，且是彼此的字母异位词。

```python
class Solution:
    # 并查集，邻接矩阵，由于题目中搜友字符串的长度相同并且两两互为变位词，所以两个字符串之间的对应位置不同字符的字符的个数不超过两个，那么它们一定相似
    # # 时间复杂度：O(a(n))，空间复杂度：O(n^2)
    def numSimilarGroups(self, strs: List[str]) -> int:
        fathers = [0] * len(strs)
        for i in range(len(fathers)):  # 初始化每个节点的祖先节点为自己
            fathers[i] = i
        groups = len(strs)
        for i in range(len(strs)):
            for j in range(i+1, len(strs)):
                if self.similar(strs[i], strs[j]) and self.union(fathers, i, j):
                    groups -= 1
        return groups

    def similar(self, str1, str2):
        diffCount = 0
        for i in range(len(str1)):
            if str1[i] != str2[i]:
                diffCount += 1
        return diffCount <= 2

    def findFather(self, fathers, i):
        if fathers[i] != i:
            fathers[i] = self.findFather(fathers, fathers[i])
        return fathers[i]
```

```
    def union(self, fathers, i, j):
        fatherOfI = self.findFather(fathers, i)
        fatherOfJ = self.findFather(fathers, j)
        if fatherOfI != fatherOfJ:
            fathers[fatherOfI] = fatherOfJ
            return True
        return False
```

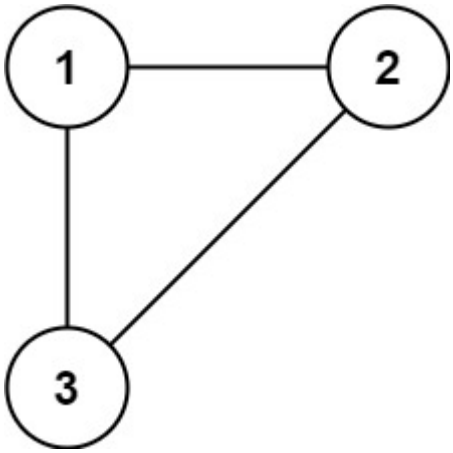
剑指 Offer II 118. 多余的边

树可以看成是一个连通且 无环 的 无向 图。

给定往一棵 n 个节点 (节点值 1～n) 的树中添加一条边后的图。添加的边的两个顶点包含在 1 到 n 中间，且这条附加的边不属于树中已存在的边。图的信息记录于长度为 n 的二维数组 edges ，edges[i] = [ai, bi] 表示图中在 ai 和 bi 之间存在一条边。

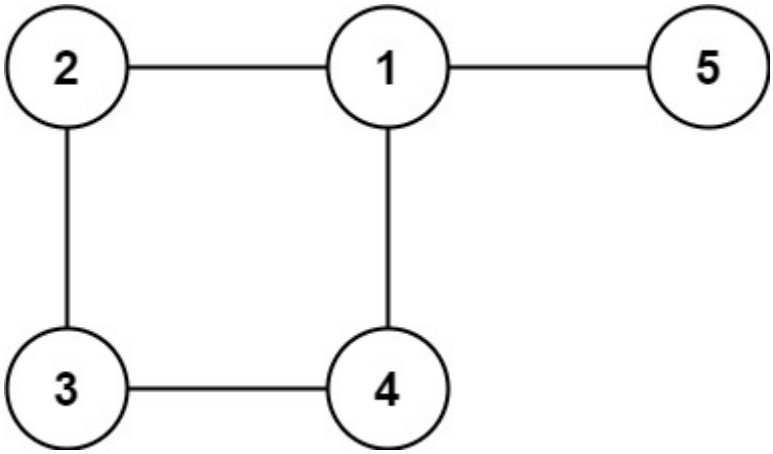
请找出一条可以删去的边，删除后可使得剩余部分是一个有着 n 个节点的树。如果有多个答案，则返回数组 edges 中最后出现的边。

示例 1：




```
输入: edges = [[1,2],[1,3],[2,3]]
输出: [2,3]
```

示例 2：



输入: edges = [[1,2],[2,3],[3,4],[1,4],[1,5]]
输出: [1,4]

提示:

n == edges.length
3 <= n <= 1000
edges[i].length == 2
1 <= ai < bi <= edges.length
ai != bi
edges 中无重复元素
给定的图是连通的

```python
class Solution:
    def find_root(self, num, root_list):
        if num == root_list[num]:
            return num
        else:
            root_list[num] = self.find_root(root_list[num], root_list)
            return root_list[num]


    def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:
        n = len(edges)
        # 没有0点 所以要n+1
        root_list = [i for i in range(n + 1)]
        for i in edges:
            root1 = self.find_root(i[0], root_list)
            root2 = self.find_root(i[1], root_list)
            if root1 == root2:
                return i
            else:
                root_list[root1] = root2
```

剑指 Offer II 119. 最长连续序列

给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。

示例 1：

输入：nums = [100,4,200,1,3,2]
输出：4
解释：最长数字连续序列是 [1, 2, 3, 4]。它的长度为 4。
示例 2：

输入：nums = [0,3,7,2,5,8,4,6,0,1]
输出：9

提示：

```
0 <= nums.length <= 104
-109 <= nums[i] <= 109
```

```python
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        if not nums : return 0
        nums=list(set(nums))
        nums.sort()
        maxlen=1
        start=0
        for end in range(1,len(nums)):
            if nums[end]-nums[end-1]==1:
                maxlen=max(maxlen,end-start+1)
            else:
                start=end
        return maxlen
```