

四月LeetCode刷题合集

闪电内推公众号

关注闪电内推，你将获得字节等大厂的内推信息与面经分享



迪乐阅读公众号


关注迪乐阅读，你将获得每日LeetCode题目题解

刷题交流QQ群

群号：613531872，每日刷题，下一个字节大佬就是你

资料整理人



干紫 

四月LeetCode刷题合集

闪电内推公众号

迪乐阅读公众号

刷题交流QQ群

资料整理人

Day 1 Move Zeroes

283.移动零

Day 2 Best Time to Buy and Sell Stock II

图文题解公众号：迪乐阅读

122.买卖股票的最佳时机 II
Day 3 Group Anagrams
49.字母异位词分组
Day 4 Counting Elements
1426.元素计数
Day 5 Middle of the Linked List
876.链表的中间节点
Day 6 Backspace String Compare
844.比较含退格的字符串
Day 7 Min Stack
155.最小栈
Day 8 Diameter of Binary Tree
543.二叉树的直径
Day 9 Last Stone Weight
247.最后一块石头的重量
Day 10 Contiguous Array
525.连续数组
Day 11 Perform String Shifts
1427.Perform String Shifts
Day 12 Product of Array Except Self
238.除自身以外数组的乘积
Day 13 Valid Parenthesis String
678.有效的括号字符串
Day 14 Number of Islands
200.岛屿数量
Day 15 Minimum Path Sum
64.最小路径和
Day 16 Search in Rotated Sorted Array
33.搜索旋转排序数组
Day 17 Construct Binary Search Tree from Preorder Traversal
1008.先序遍历构造二叉树
Day 18 Leftmost Column with at Least a One
1428.至少有一个1的最左端列
Day 19 Subarray Sum Equals K
560.和为K的子数组
Day 20 Bitwise AND of Numbers Range
201.数字范围按位与
Day 21 LRU Cache
146. LRU缓存机制
Day 22 Jump Game
55.跳跃游戏
Day 23 Longest Common Subsequence
1143.最长公共子序列
Day 24 Maximal Square
221.最大正方形
Day 25 First Unique Number
1429.首个唯一的数字
Day 26 Binary Tree Maximum Path Sum
124.二叉树中的最大路径和
Day 27 Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree
1430.检查字符串是否是二叉树中从根到叶路径的有效序列

Day 1 Move Zeroes

将数组的零移动到末尾

Given an array *nums*, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Example:

```
1 Input: [0,1,0,3,12]
2 Output: [1,3,12,0,0]
```

Note:

- 1、不能创建一个新的数组来做操作
- 2、尽量少的操作次数

283.移动零

给定一个数组 *nums*，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。

示例:

```
1 输入: [0,1,0,3,12]
2 输出: [1,3,12,0,0]
```

Python 题解:

```
1 class Solution:
2     def moveZeroes(self, nums):
3         for i in range(len(nums)-1::-1):
4             if nums[i] == 0:
5                 #若该元素为0删除该元素
6                 nums.pop(i)
7                 #末尾补0
8                 nums.append(0)
```

指针从末尾开始，寻找0元素，删除，末尾补齐。

Day 2 Best Time to Buy and Sell Stock II

Say you have an array for which the *i* element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

```
1 Input: [7,1,5,3,6,4]
2 Output: 7
3 Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5),
  profit = 5-1 = 4.
4 Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-
  3 = 3.
```

Example 2:

```
1 Input: [1,2,3,4,5]
2 Output: 4
3 Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5),
  profit = 5-1 = 4.
4 Note that you cannot buy on day 1, buy on day 2 and sell them later, as
  you are engaging multiple transactions at the same time. You must sell
  before buying again.
```

Example 3:

```
1 Input: [7,6,4,3,1]
2 Output: 0
3 Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

122.买卖股票的最佳时机 II

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

```
1 输入: [7,1,5,3,6,4]
2 输出: 7
3 解释: 在第 2 天 (股票价格 = 1) 的时候买入, 在第 3 天 (股票价格 = 5) 的时候卖出, 这笔交易
  所能获得利润 = 5-1 = 4 。
4 随后, 在第 4 天 (股票价格 = 3) 的时候买入, 在第 5 天 (股票价格 = 6) 的时候卖出, 这
  笔交易所能获得利润 = 6-3 = 3 。
```

示例 2:

```
1 输入: [1,2,3,4,5]
2 输出: 4
3 解释: 在第 1 天 (股票价格 = 1) 的时候买入, 在第 5 天 (股票价格 = 5) 的时候卖出, 这笔交
  易所能获得利润 = 5-1 = 4 。
4 注意你不能在第 1 天和第 2 天接连购买股票, 之后再将它们卖出。
5 因为这样属于同时参与了多笔交易, 你必须在再次购买前出售掉之前的股票。
```

示例 3:

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

提示：

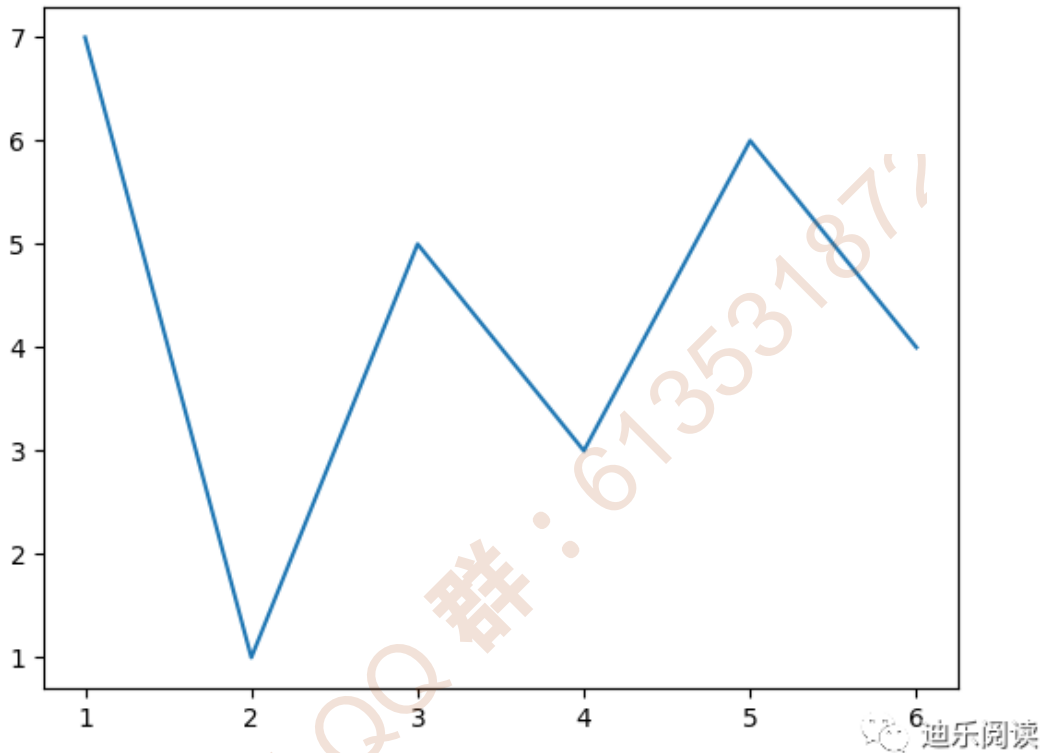
```
1 <= prices.length <= 3 * 10 ^ 4
```

```
0 <= prices[i] <= 10 ^ 4
```

强烈建议大家先别急着下手，写写画画，此题很有意思，千万不要绕进去了~

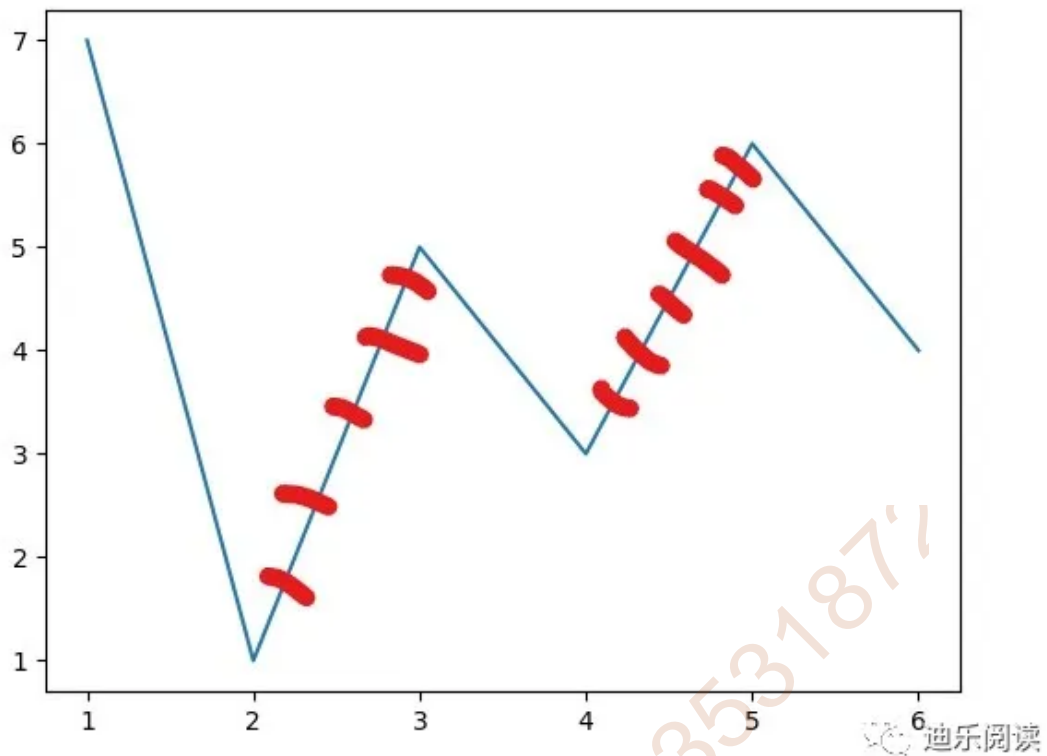
解题思路

我们来画个简图观察一下Example 1中价格变动的形态：



由于不能有多笔交易同时发生 (**Note:** You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again))，那么，什么情况下盈利最大？

其实就是所有的上涨部分获利全部拿到！如下图：



是不是很简单!

python题解:

```

1 class Solution:
2     def maxProfit(self, prices: List[int]) -> int:
3         # 初始化短期获利列表
4         get = []
5         for i in range(len(prices)-1):
6             # 如果后一项比前一项的值大
7             # 则说明股价回升
8             # 此时直接卖出以获利
9             if prices[i+1] > prices[i]:
10                # 在列表中加入短期获利数值
11                get.append(prices[i+1]-prices[i])
12            #返回所有获利
13            return sum(get)

```

Day 3 Group Anagrams

Given an array of strings, group anagrams together.

Example:

```

1 Input: ["eat", "tea", "tan", "ate", "nat", "bat"],
2 Output:
3 [
4   ["ate","eat","tea"],
5   ["nat","tan"],
6   ["bat"]
7 ]

```

Note:

- 1, All inputs will be in lowercase.
- 2, The order of your output does not matter.

49.字母异位词分组

给定一个字符串数组，将字母异位词组合在一起。字母异位词指字母相同，但排列不同的字符串。

示例:

```

1 输入: ["eat", "tea", "tan", "ate", "nat", "bat"]
2 输出:
3 [
4   ["ate","eat","tea"],
5   ["nat","tan"],
6   ["bat"]
7 ]

```

说明:

所有输入均为小写字母。

不考虑答案输出的顺序。

【解题思路】 此题主要考察对字典操作的应用。思考如下过程:

初始单词: ['eat', 'tea', 'tan', 'ate', 'nat', 'bat']

组合过程:

当前单词: eat

所含字母: aet

{'aet': ['eat']}

当前单词: tea

所含字母: aet

{'aet': ['eat', 'tea']}

当前单词: tan

所含字母: ant

{'aet': ['eat', 'tea'], 'ant': ['tan']}

当前单词: ate

所含字母: aet

{'aet': ['eat', 'tea', 'ate'], 'ant': ['tan']}

当前单词: nat

所含字母: ant

{'aet': ['eat', 'tea', 'ate'], 'ant': ['tan', 'nat']}

当前单词: bat

所含字母: abt

{'aet': ['eat', 'tea', 'ate'], 'ant': ['tan', 'nat'], 'abt': ['bat']}

【算法流程】确定单词所包含字母组合——搜索是否已在字典中——是，放入对应键下；否，创造新的键，并赋值。

python 题解

```
1 class Solution:
2     def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
3         temp = {}
4         for i in strs:
5             # 确定当前单词所含字母顺序
6             key_word = ''.join(sorted(i))
7             # 判断该字母顺序键是否存在
8             if key_word in temp:
9                 # 是，追加
10                temp[key_word] += [i]
11            else:
12                # 否，创建
13                temp[key_word] = [i]
14
15        return list(temp.values())
```

另，今天看到一个有趣的概念【Levenshtein Distance】：

若要由一个单词 w_1 转换为另一个单词 w_2 所需要的最少单字符编辑操作次数是多少呢？

注：单字符编辑操作有且仅有三种：

- 插入 (Insertion)
- 删除 (Deletion)
- 替换 (Substitution)

例如："kitten" 和 "sitting" 这两个单词，由 "kitten" 转换为 "sitting" 需要的最少单字符编辑操作有：

- 1.kitten → sitten (substitution of "s" for "k")
- 2.sitten → sittin (substitution of "i" for "e")
- 3.sittin → sitting (insertion of "g" at the end)

因此，"kitten" 和 "sitting" 这两个单词之间的编辑距离为 3。

Day 4 Counting Elements

Given an integer array `arr`, count element `x` such that `x + 1` is also in `arr`.

If there're duplicates in `arr`, count them separately.

Example 1:

```
1 Input: arr = [1,2,3]
2 Output: 2
3 Explanation: 1 and 2 are counted cause 2 and 3 are in arr.
```

Example 2:


```
1 Input: arr = [1,1,3,3,5,5,7,7]
2 Output: 0
3 Explanation: No numbers are counted, cause there's no 2, 4, 6, or 8 in arr.
```

Example 3:

```
1 Input: arr = [1,3,2,3,5,0]
2 Output: 3
3 Explanation: 0, 1 and 2 are counted cause 1, 2 and 3 are in arr.
```

Example 4:

```
1 Input: arr = [1,1,2,2]
2 Output: 2
3 Explanation: Two 1s are counted cause 2 is in arr.
```

1426.元素计数

给定一个整数数组arr，计数元素x，使x+1也在arr。

如果在arr中有重复的，分开计数。

解题思路

【字典，哈希表的使用】

仔细观察题目后发现今天的题目也是字典的使用（看来字典的使用频率真的高）。

题目要求给定一个数组后，统计数组中含有比自己大1的数的个数，感觉像绕口令。

我们来把它游戏化一下。一个屋里有N个小朋友，每个小朋友分配一个数字。

对于每一个小朋友，如果屋里有另一个小朋友手中的数字刚好比他大1的话，就给他一块巧克力。

问：屋里一共有多少小朋友拿到巧克力了啊？

```
1 class Solution:
2     def countElements(self, arr: List[int]) -> int:
3         # 创建字典
4         hash_table = defaultdict(int)
5         # 得到巧克力的小朋友数
6         num = 0
7         for i in arr:
8             # 统计拿到数字i的小朋友人数
9             hash_table[i] += 1
10        for i in hash_table:
11            # 判断是否有拿到i+1的小朋友
12            if i+1 in hash_table:
13                # 如果有，拿到i的小朋友每发巧克力一块！
14                num += hash_table[i]
15        # 总共有多少小朋友拿到巧克力呢？
16        return num
```

【编程思路】建立字典——用获得数字作为“键”；每个数字有几个人获得作为键的“值”——搜索字典中有没有比当前“键”大1的“键”——有，当前“键”下，每个人拿块巧克力；没有，很抱歉，只能看别人吃巧克力咯——统计所有拿到巧克力小朋友的人数——完成！

但是，吃饼少女觉得应该1对1才有意思。

比如，游戏规则变成：

每个拿到数字的小朋友要快速找到屋里拿到比他数字刚好大1的小朋友。

如果找到即赶紧抓牢，最后，手上抓着比自己大1的小朋友才有巧克力拿。

比如，Input=[1,1,2]，结果是1。

```
1 class Solution:
2     def countElements(self, arr: List[int]) -> int:
3         # 创建字典
4         hash_table = defaultdict(int)
5         # 得到巧克力的小朋友数
6         num = 0
7         for i in arr:
8             # 统计拿到数字i的小朋友人数
9             hash_table[i] += 1
10        for i in hash_table:
11            # 判断是否有拿到i+1的小朋友
12            if i+1 in hash_table:
13                # 找到比自己大1的i的小朋友发巧克力一块！
14                num += min(hash_table[i], hash_table[i+1])
15        # 总共有多少个小朋友拿到巧克力呢？
16        return num
```

Day 5 Middle of the Linked List

Given a non-empty, singly linked list with head node head, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

Example 1:

```
1 Input: [1,2,3,4,5]
2 Output: Node 3 from this list (Serialization: [3,4,5])
3 The returned node has value 3. (The judge's serialization of this node is
4 [3,4,5]).
5 Note that we returned a ListNode object ans, such that:
6 ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, and ans.next.next.next
7 = NULL.
```

Example 2:

```
1 Input: [1,2,3,4,5,6]
2 Output: Node 4 from this list (Serialization: [4,5,6])
3 Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Note:

The number of nodes in the given list will be between 1 and 100.

876.链表的中间节点

给定一个带有头结点 head 的非空单链表，返回链表的中间结点。

如果有两个中间结点，则返回第二个中间结点。

示例 1:

```
1 输入: [1,2,3,4,5]
2 输出: 此列表中的结点 3 (序列化形式: [3,4,5])
3 返回的结点值为 3 。 (测评系统对该结点序列化表述是 [3,4,5])。
4 注意，我们返回了一个 ListNode 类型的对象 ans，这样：
5 ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及 ans.next.next.next = NULL.
```

示例 2:

```
1 输入: [1,2,3,4,5,6]
2 输出: 此列表中的结点 4 (序列化形式: [4,5,6])
3 由于该列表有两个中间结点，值分别为 3 和 4，我们返回第二个结点。
```

提示:

给定链表的结点数介于 1 和 100 之间。

解题思路

【链表的使用】

【解法1】:

- 1, 遍历链表，得到链表中元素的个数
- 2, 计算中间节点的位置
- 3, 从头节点出发走到中间节点即可

```
1 class Solution:
2     def middleNode(self, head: ListNode) -> ListNode:
3         # 初始化链表节点数目
4         n = 1
5         # 指向链表头节点
6         node = head
7         # 统计链表中的节点数目
8         while node.next != None:
9             node = node.next
10            n += 1
11
```

```

12     # 如果节点个数为偶数
13     if n % 2 == 0:
14         # 计算靠后的节点的位置
15         n = n/2 + 1
16     # 如果节点个数为奇数
17     else:
18         # 计算中间节点的位置
19         n = n/2 + 0.5
20     # 指向链表头节点
21     node = head
22     # 从头节点开始向后走相应的步数即可
23     for i in range(int(n-1)):
24         node = node.next
25     # 返回答案
26     return node

```

复杂度 $O(n)$ 。

【解法2】：

- 1, 定义两个指针, slow/fast
- 2, fast指针走完全部时, slow刚好走一半

```

1 class Solution:
2     def middleNode(self, head: ListNode) -> ListNode:
3         slow = head
4         fast = head
5         while fast != None and fast.next != None:
6             slow = slow.next
7             fast = fast.next.next
8         return slow

```

复杂度 $O(n)$ 。

Day 6 Backspace String Compare

Given two strings **S** and **T**, return if they are equal when both are typed into empty text editors.

means a backspace character.

Example 1:

```

1 Input: S = "ab#c", T = "ad#c"
2 Output: true
3 Explanation: Both S and T become "ac".

```

Example 2:

```

1 Input: S = "ab##", T = "c#d#"
2 Output: true
3 Explanation: Both S and T become "".

```

Example 3:

```
1 Input: S = "a##c", T = "#a#c"
2 Output: true
3 Explanation: Both S and T become "c".
```

Example 4:

```
1 Input: S = "a#c", T = "b"
2 Output: false
3 Explanation: S becomes "c" while T becomes "b".
```

Note:

- `1 <= S.length <= 200`
- `1 <= T.length <= 200`
- `S` and `T` only contain lowercase letters and `'#'` characters.

Follow up:

- Can you solve it in `O(N)` time and `O(1)` space?

844.比较含退格的字符串

给定 `S` 和 `T` 两个字符串，当它们分别被输入到空白的文本编辑器后，判断二者是否相等，并返回结果。
代表退格字符。

注意：如果对空文本输入退格字符，文本继续为空。

示例 1:

```
1 输入: S = "ab#c", T = "ad#c"
2 输出: true
3 解释: S 和 T 都会变成 "ac".
```

示例 2:

```
1 输入: S = "ab##", T = "c#d#"
2 输出: true
3 解释: S 和 T 都会变成 "".
```

示例 3:

```
1 输入: S = "a##c", T = "#a#c"
2 输出: true
3 解释: S 和 T 都会变成 "c".
```

示例 4:

```
1 输入: S = "a#c", T = "b"
2 输出: false
3 解释: S 会变成 "c", 但 T 仍然是 "b".
```

提示:

```
1 <= S.length <= 200
```

```
1 <= T.length <= 200
```

S 和 T 只含有小写字母以及字符 '#'。

进阶:

你可以用 $O(N)$ 的时间复杂度和 $O(1)$ 的空间复杂度解决该问题吗?

解题思路

【链表模拟栈的操作】

【解法1】:

利用栈的思想, 遍历字符串的每一个字符, 遇到 '#' 且栈不为空则将栈顶弹出, 否则该字符入栈。

```
1 class Solution:
2     def backspaceCompare(self, S: str, T: str) -> bool:
3         # 定义执行过程函数
4         def search(s):
5             # 初始化列表
6             set = []
7             for i in s:
8                 # 如果字符不为 '#'
9                 if i != '#':
10                    set.append(i)
11                # 如果列表不为空
12                elif set:
13                    # 删除列表尾部
14                    set.pop()
15            return set
16        return search(S) == search(T)
```

时间复杂度 $O(M+N)$, 空间复杂度 $O(M+N)$, M/N 为两字符串长度。

【解法2】: 反向遍历字符串。如果看到退格字符, 则跳过下一个非退格字符。如果一个字符没有被跳过, 它就是最终答案的一部分。

```
1 class Solution(object):
2     def backspaceCompare(self, S, T):
3         def F(s):
4             skip = 0
5             # 遍历反转后的字符串
6             for x in reversed(s):
7                 if x == '#':
8                     skip += 1
9                 elif skip:
10                    skip -= 1
11                else:
12                    yield x
13            # [itertools.izip_longest] 返回一个合并了多个迭代器为一个元组的迭代器。
14            return all(x == y for x, y in itertools.izip_longest(F(S), F(T)))
```

时间复杂度 $O(M+N)$, 空间复杂度 $O(1)$, M/N 为两字符串长度。

Day 7 Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Example :

```
1 MinStack minStack = new MinStack();
2 minStack.push(-2);
3 minStack.push(0);
4 minStack.push(-3);
5 minStack.getMin(); --> Returns -3.
6 minStack.pop();
7 minStack.top(); --> Returns 0.
8 minStack.getMin(); --> Returns -2.
```

155.最小栈

设计一个支持 push , pop , top 操作，并能在常数时间内检索到最小元素的栈。

push(x) —— 将元素 x 推入栈中。

pop() —— 删除栈顶的元素。

top() —— 获取栈顶元素。

getMin() —— 检索栈中的最小元素。

示例:

```
1 输入:
2 ["MinStack","push","push","push","getMin","pop","top","getMin"]
3 [[],[ -2],[ 0],[ -3],[ ],[ ],[ ],[ ]]
4
5 输出:
6 [null,null,null,null,-3,null,0,-2]
```

提示:

- pop、top 和 getMin 操作总是在 **非空栈** 上调用。

其实对于Python3来说直接使用链表的操作可以很简单的实现这道题。如下:

```
1 class MinStack:
2     def __init__(self):
3         self.stack = []
4     def push(self, x: int) -> None:
5         self.stack.append(x)
6     def pop(self) -> None:
7         self.stack.pop()
8     def top(self) -> int:
9         return self.stack[-1]
10    def getMin(self) -> int:
11        return min(self.stack)
```

但是！直接使用list相关操纵会增加搜索时间，比如min()将遍历list中所有元素。功能的实现仅仅是最低标准！来看看目前排名最靠前的解法吧：

解题思路

【链表模拟栈的操作】

- 使用列表来代替栈
- 列表 `array` 将元素按照操作顺序“入栈”
- 列表 `stack` 中存放当前已知最小元素
- `push()` 函数执行时，元素添加在列表 `array` 尾部，若该元素为当前已知最小元素，则将之添加在列表 `stack` 尾部
- `pop()` 函数执行时，直接弹出列表 `array` 尾部元素，若该元素为列表 `stack` 尾部元素，则将之从列表 `stack` 尾部删除
- `top()` 函数执行时，直接返回列表 `array` 尾部元素
- `getMin()` 函数执行时，直接返回列表 `stack` 尾部元素

```
1 class MinStack:
2     def __init__(self):
3         """
4         initialize your data structure here.
5         """
6         # 初始化列表
7         # 使用列表来代替栈
8         self.array=[]
9         self.stack=[]
10
11     def push(self, x: int) -> None:
12         # 将 x 添加在 array 尾部
13         self.array.append(x)
14         # 如果此时 stack 为空，或 stack 尾部元素不小于 x
15         if len(self.stack)==0 or x<=self.stack[-1]:
16             # 将 x 添加在 stack 尾部
17             self.stack.append(x)
18
19     def pop(self) -> None:
20         # 获取 array 尾部元素并将之删除
21         val=self.array.pop()
22         # 如果该元素与 stack 尾部元素相同
23         if val==self.stack[-1]:
24             # 则将 stack 尾部元素删除
25             self.stack.pop()
26
27     def top(self) -> int:
28         # 返回 array 的尾部元素
29         return self.array[-1]
30
31     def getMin(self) -> int:
32         # 返回 stack 的尾部元素
33         return self.stack[-1]
34
35 # Your MinStack object will be instantiated and called as such:
36 # obj = MinStack()
37 # obj.push(x)
38 # obj.pop()
```



```

39 | # param_3 = obj.top()
40 | # param_4 = obj.getMin()

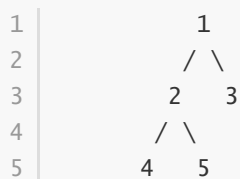
```

Day 8 Diameter of Binary Tree

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest** path between any two nodes in a tree. This path may or may not pass through the root.

Example :

Given a binary tree



Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].

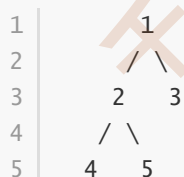
Note: The length of path between two nodes is represented by the number of edges between them.

543. 二叉树的直径

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过也可能不穿过根结点。

示例：

给定二叉树



返回 3, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

注意：两结点之间的路径长度是以它们之间边的数目表示。

解题思路

【使用递归的思想】

- 分别求取当前节点左、右子树节点到自身的最长路径，两者相加即为答案

```

1 | # Definition for a binary tree node.
2 | # class TreeNode:
3 | #     def __init__(self, x):

```

```

4      #         self.val = x
5      #         self.left = None
6      #         self.right = None
7
8      class Solution:
9          def diameterOfBinaryTree(self, root: TreeNode) -> int:
10             # 初始化最长路径距离 d = 0
11             d = 0
12             # 递归函数
13             def helper(root):
14                 # 声明 d 不是函数的局部变量
15                 nonlocal d
16                 # 当前根节点为空，则返回 0
17                 if not root:
18                     return 0
19                 # 递归求左边的最长路径
20                 l = helper(root.left)
21                 # 递归求右边的最长路径
22                 r = helper(root.right)
23                 # 左右最长路径距离之和与当前最长路径距离对比
24                 # 更新当前最长路径
25                 d = max(d, l+r)
26                 # 返回当前节点作为根节点时的最长路径距离
27                 return max(l, r)+1
28             # 执行递归函数
29             helper(root)
30             # 返回最长路径距离
31             return d

```

Day 9 Last Stone Weight

We have a collection of stones, each stone has a positive integer weight.

Each turn, we choose the two **heaviest** stones and smash them together. Suppose the stones have weights x and y with $x \leq y$. The result of this smash is:

- If $x == y$, both stones are totally destroyed;
- If $x \neq y$, the stone of weight x is totally destroyed, and the stone of weight y has new weight $y-x$.

At the end, there is at most 1 stone left. Return the weight of this stone (or 0 if there are no stones left.)

Example :

```
1 Input: [2,7,4,1,8,1]
2 Output: 1
3 Explanation:
4 We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
5 we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
6 we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
7 we combine 1 and 1 to get 0 so the array converts to [1] then that's the
  value of last stone.
```

Note:

1. `1 <= stones.length <= 30`
2. `1 <= stones[i] <= 1000`

247.最后一块石头的重量

有一堆石头，每块石头的重量都是正整数。

每一回合，从中选出两块最重的石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y ，且 $x \leq y$ 。那么粉碎的可能结果如下：

如果 $x == y$ ，那么两块石头都会被完全粉碎；

如果 $x \neq y$ ，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 $y-x$ 。

最后，最多只会剩下一块石头。返回此石头的重量。如果没有石头剩下，就返回 0。

示例：

```
1 输入: [2,7,4,1,8,1]
2 输出: 1
3 解释:
4 先选出 7 和 8，得到 1，所以数组转换为 [2,4,1,1,1]，
5 再选出 2 和 4，得到 2，所以数组转换为 [2,1,1,1]，
6 接着是 2 和 1，得到 1，所以数组转换为 [1,1,1]，
7 最后选出 1 和 1，得到 0，最终数组转换为 [1]，这就是最后剩下那块石头的重量。
```

提示：

1. `1 <= stones.length <= 30`
2. `1 <= stones[i] <= 1000`

解题思路

【队列排序法】

- 先将List从小到大排序
- 对比最后两块“石头”：一样重，销毁；不一样中，销毁小的，大的减去小的
- 返回最后剩下的石头重量，若没有剩，返回0

```
1 class Solution:
2     def lastStoneweight(self, stones: List[int]) -> int:
3         while len(stones) > 1:
4             # 将List从小到大排序
5             stones.sort()
6             L = len(stones)-1
7             if stones[L] == stones[L-1]:
8                 # 剔除最后一个
```

```

9         stones.pop()
10        # 剔除倒数第二个
11        stones.pop()
12        elif stones[L] > stones[L-1]:
13            stones[L] = stones[L] - stones[L-1]
14            stones.pop(L-1)
15        else:
16            stones[L-1] = stones[L-1] - stones[L]
17            stones.pop(L)
18
19        return stones[0] if stones else 0

```

Day 10 Contiguous Array

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Example 1:

```

1 Input: [0,1]
2 Output: 2
3 Explanation: [0, 1] is the longest contiguous subarray with equal number of 0
  and 1.

```

Example 2:

```

1 Input: [0,1,0]
2 Output: 2
3 Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal
  number of 0 and 1.

```

Note: The length of the given binary array will not exceed 50,000.

525.连续数组

给定一个二进制数组, 找到含有相同数量的 0 和 1 的最长连续子数组 (的长度)。

示例 1:

```

1 输入：[0,1]
2 输出：2
3 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:

```

1 输入：[0,1,0]
2 输出：2
3 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。

```

注意：给定的二进制数组的长度不会超过50000。

解题思路

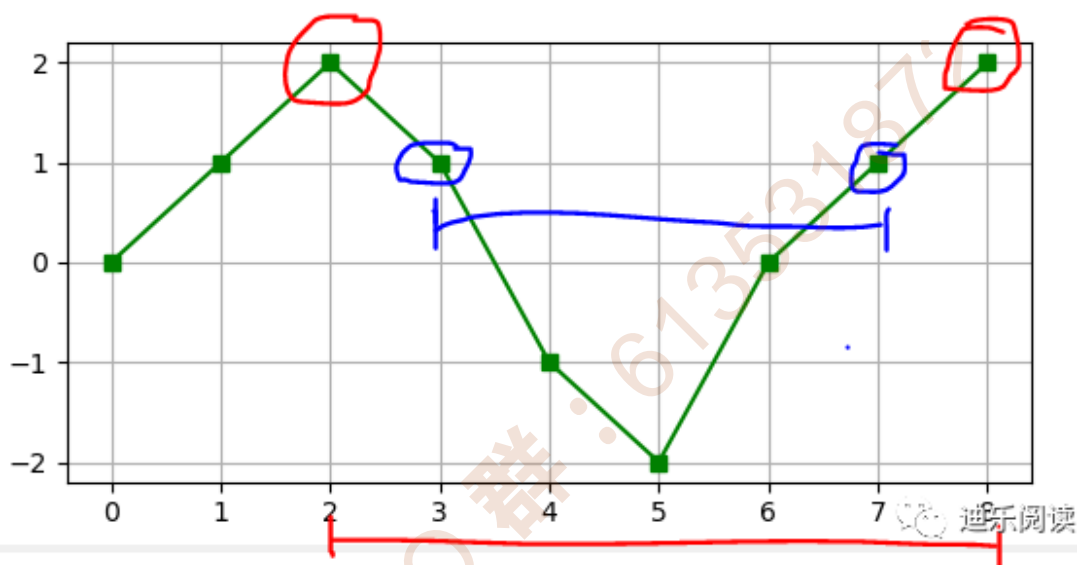
【遍历求和+字典】

- 设定sum = 0（注意这不是题目要求的答案，往后看就会知道这是干什么的了。）
- 从头遍历：如果当前值是0就sum-1，否则就sum+1

重点是理解下面这句话：

“当再次出现之前出现过的sum值时，中间所包含的0/1个数即为题目答案！”

- 以[1,1,0,0,1,1,1]为例，初始sum为0，
- 计算后续sum值为[0,1,2,1,-1,-2,0,1,2]。
- 上图：



图中绿色方块表示计及第i个数（0/1）时当前的sum值

当后续计算中sum再次出现时（如红色一对，蓝色一对），两次相同sum之间的0/1个数即为本题答案！

【字典法】

- 设定初始sum = 0
- 依此计算后续元素加入后sum值
- 若该sum值已在字典中，此时元素位置减去上一次出现相同sum时元素
- 若该sum不在字典中，以该sum为key，对应元素位置为值加入字典

```
1 class Solution:
2     def findMaxLength(self, nums):
3         """
4         :type nums: List[int]
5         :rtype: int
6         """
7         total_sum = 0
8         # 建立以sum为key的字典
9         index_map = dict()
10        index_map[0] = -1
11        res = 0
12        for i, num in enumerate(nums):
13            if num == 0:
14                total_sum -= 1
```

```

15         else:
16             total_sum += 1
17         if total_sum in index_map:
18             res = max(res, i - index_map[total_sum])
19         else:
20             index_map[total_sum] = i
21     return res

```

时间复杂度 $O(n)$ ，空间复杂度 $O(n)$ 。

Day 11 Perform String Shifts

You are given a string `s` containing lowercase English letters, and a matrix `shift`, where `shift[i] = [direction, amount]`:

- `direction` can be `0` (for left shift) or `1` (for right shift).
- `amount` is the amount by which string `s` is to be shifted.
- A left shift by 1 means remove the first character of `s` and append it to the end.
- Similarly, a right shift by 1 means remove the last character of `s` and add it to the beginning.

Return the final string after all operations.

Example 1:

```

1 Input: s = "abc", shift = [[0,1],[1,2]]
2 Output: "cab"
3 Explanation:
4 [0,1] means shift to left by 1. "abc" -> "bca"
5 [1,2] means shift to right by 2. "bca" -> "cab"

```

Example 2:

```

1 Input: s = "abcdefg", shift = [[1,1],[1,1],[0,2],[1,3]]
2 Output: "efgabcd"
3 Explanation:
4 [1,1] means shift to right by 1. "abcdefg" -> "gabcdef"
5 [1,1] means shift to right by 1. "gabcdef" -> "fgabcde"
6 [0,2] means shift to left by 2. "fgabcde" -> "abcdefg"
7 [1,3] means shift to right by 3. "abcdefg" -> "efgabcd"

```

Constraints:

- `1 <= s.length <= 100`
- `s` only contains lower case English letters.
- `1 <= shift.length <= 100`
- `shift[i].length == 2`
- `0 <= shift[i][0] <= 1`
- `0 <= shift[i][1] <= 100`

1427.Perform String Shifts

题目大意:

给定一个只包含小写字母的字符串s和一个转移(shift) 矩阵, 其中shift[i] = [direction, amount]:

direction只能是0或者1, 0表示字符串左移, 1表示右移,

amount表示字符串s的移动的数量

左移1次表示将s的第一个字符移除, 并将其插入(append)到字符串尾

同样的, 右移1次表示将s的最后一个字符移除, 并将其插入(append)到字符串首

最后返回经过所有操作后的串。

解题思路

【列表搜索】

仔细想想就会发现一次左移和一次右移抵消了。

所以, 直接统计左右移位的总次数再进行移位就可以啦。

```
1 class Solution:
2     def stringShift(self, s: str, shift: List[List[int]]) -> str:
3         s = list(s)
4         total_shift = 0
5         for i in shift:
6             if i[0] == 0:
7                 total_shift -= i[1]
8             else:
9                 total_shift += i[1]
10
11         if total_shift > 0:
12             for i in range(total_shift):
13                 s.insert(0,s.pop())
14         elif total_shift < 0:
15             for i in range(abs(total_shift)):
16                 s.append(s[0])
17                 s.remove(s[0])
18
19         return "".join(s)
```

Day 12 Product of Array Except Self

Given an array `nums` of n integers where $n > 1$, return an array `output` such that `output[i]` is equal to the product of all the elements of `nums` except `nums[i]`.

Example:

```
1 Input: [1,2,3,4]
2 Output: [24,12,8,6]
```

Constraint: It's guaranteed that the product of the elements of any prefix or suffix of the array (including the whole array) fits in a 32 bit integer.

Note: Please solve it **without division** and in $O(n)$.

Follow up:

Could you solve it with constant space complexity? (The output array **does not** count as extra space for the purpose of space complexity analysis.)

238.除自身以外数组的乘积

给你一个长度为 n 的整数数组 `nums`，其中 $n > 1$ ，返回输出数组 `output`，其中 `output[i]` 等于 `nums` 中除 `nums[i]` 之外其余各元素的乘积。

示例:

```
1 输入: [1,2,3,4]
2 输出: [24,12,8,6]
```

提示: 题目数据保证数组之中任意元素的全部前缀元素和后缀（甚至是整个数组）的乘积都在 32 位整数范围内。

说明: 请不要使用除法，且在 $O(n)$ 时间复杂度内完成此题。

进阶:

你可以在常数空间复杂度内完成这个题目吗？（出于对空间复杂度分析的目的，输出数组不被视为额外空间。）

解题思路

【注意事项】

- 不能使用除法，不然题目就太没意思了；
- 算法复杂度必须控制在 $O(n)$ ，不然外层一个 `for` 循环，内层左右遍历一下就完事了；
- 空间复杂度最好是常数。

【算法思路】

考虑这样一个包含4个元素的数组 `nums = [a, b, c, d]`，预期输出的结果为 `output = [b*c*d, a*c*d, a*b*d, a*b*c]`，为了达到目的，我们构建两个数组相乘：

- 数组1: `[1, a, a*b, a*b*c]`
- 数组2: `[b*c*d, c*d, d, 1]`

两者元素依次相乘就是结果。

【实现过程】

1. 从前向后遍历得到数组1
2. 从后向前遍历得到数组2
3. 两者相乘得到答案

```
1 class Solution:
2     def productExceptSelf(self, nums: List[int]) -> List[int]:
3         n = len(nums)
4         output = [1]*n
5         cumProductLeft = [1]*n+1
6         cumProductRight = [1]*n+1
7         for i in range(1, n):
```



```

8         cumProductLeft[i] = cumProductLeft[i-1]*nums[i-1]
9     for i in range(n-2, -1, -1):
10         cumProductRight[i] = cumProductRight[i+1]*nums[i+1]
11     for i in range(n):
12         output[i] = cumProductLeft[i] * cumProductRight[i]
13
14     return output

```

时间复杂度 $O(n)$ ，空间复杂度 $O(n)$ 。

Day 13 Valid Parenthesis String

Given a string containing only three types of characters: '(', ')' and '*', write a function to check whether this string is valid. We define the validity of a string by these rules:

1. Any left parenthesis '(' must have a corresponding right parenthesis ')'.
2. Any right parenthesis ')' must have a corresponding left parenthesis '('.
3. Left parenthesis '(' must go before the corresponding right parenthesis ')'.
4. '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string.
5. An empty string is also valid.

Example 1:

```

1 Input: "()"
2 Output: True

```

Example 2:

```

1 Input: "(*)"
2 Output: True

```

Example 3:

```

1 Input: "(*))"
2 Output: True

```

Note:

1. The string size will be in the range [1, 100].

678.有效的括号字符串

给定一个只包含三种字符的字符串：(,) 和 *，写一个函数来检验这个字符串是否为有效字符串。有效字符串具有如下规则：

- 任何左括号 (必须有相应的右括号)。
- 任何右括号) 必须有相应的左括号 (。
- 左括号 (必须在对应的右括号之前)。

- 可以被视为单个右括号)，或单个左括号(，或一个空字符串。
一个空字符串也被视为有效字符串。

示例 1:

```
1 输入: "()"
2 输出: True
```

示例 2:

```
1 输入: "(*)"
2 输出: True
```

示例 3:

```
1 输入: "(*))"
2 输出: True
```

注意:

字符串大小将在 [1, 100] 范围内。

解题思路

【算法思路】

用一个set集合来记录这个表达式中左括号 能 比右括号多的个数。

- 遇到左括号，集合里面的每个元素应该+1；
- 遇到右括号，当集合里元素>0时，-1；
- 如果遇到*，应该+1，-1或者不运算。
- 看最后这个集合 能否 包含0，即左括号的个数等于右括号的个数。

```
1 class Solution(object):
2     def checkValidString(self, s):
3         """
4         :type s: str
5         :rtype: bool
6         """
7         old_set = set([0])
8         for c in s:
9             new_set = set()
10            if c == '(':
11                for t in old_set:
12                    new_set.add(t + 1)
13            elif c == ')':
14                for t in old_set:
15                    if t > 0:
16                        new_set.add(t - 1)
17            elif c == '*':
18                for t in old_set:
19                    new_set.add(t + 1)
20                    new_set.add(t)
21                    if t > 0:
22                        new_set.add(t - 1)
23            old_set = new_set
24        return 0 in old_set
```

Day 14 Number of Islands

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
1 Input:
2 11110
3 11010
4 11000
5 00000
6
7 Output: 1
```

Example 2:

```
1 Input:
2 11000
3 11000
4 00100
5 00011
6
7 Output: 3
```

200.岛屿数量

给你一个由 '1'（陆地）和 '0'（水）组成的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1:

```
1 输入:
2 11110
3 11010
4 11000
5 00000
6 输出: 1
```

示例 2:

```
1 输入：
2 11000
3 11000
4 00100
5 00011
6 输出：3
```

解释：每座岛屿只能由水平和/或竖直方向上相邻的陆地连接而成。

解题思路

【算法思路】

"陆地清除计划"

1. 遇到"1"则清零上下左右所有是"1"的地方，且这种清除具有传播性，即对每一个被清除的"1"均执行相似的操作
2. 岛屿面积缩水，统计剩余的"1"即可

```
1  from typing import List
2
3  class Solution:
4      def numIslands(self, grid: List[List[str]]) -> int:
5          count = 0
6          for i in range(len(grid)):
7              for j in range(len(grid[0])):
8                  if grid[i][j] == '1':
9                      count += 1
10                     self.recurse(grid,i,j)
11             return count
12
13     def recurse(self,grid,l,h):
14         # 将所有 1 的上下左右的 1 全部清零
15         # 并且对与这个 1 相邻的所有 1 执行同样的操作
16         grid[l][h] = '0'
17         if l - 1 >= 0 and grid[l-1][h] == '1':
18             self.recurse(grid,l-1,h)
19         if l + 1 < len(grid) and grid[l+1][h] == '1':
20             self.recurse(grid,l+1,h)
21         if h - 1 >= 0 and grid[l][h-1] == '1':
22             self.recurse(grid,l,h-1)
23
24         if h + 1 < len(grid[0]) and grid[l][h+1] == '1':
25             self.recurse(grid,l,h+1)
26
27     grid = [["1","1","1"],["0","1","0"],["1","1","1"]]
28     model = Solution()
29     print(model.numIslands(grid))
```

Day 15 Minimum Path Sum

Given a $m \times n$ grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example:

```
1 Input:
2 [
3   [1,3,1],
4   [1,5,1],
5   [4,2,1]
6 ]
7 Output: 7
8 Explanation: Because the path 1→3→1→1→1 minimizes the sum.
```

64.最小路径和

给定一个包含非负整数的 $m \times n$ 网格，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：每次只能向下或者向右移动一步。

示例:

```
1 输入:
2 [
3   [1,3,1],
4   [1,5,1],
5   [4,2,1]
6 ]
7 输出: 7
8 解释: 因为路径 1→3→1→1→1 的总和最小。
```

解题思路

【动态规划】

这是一道非常典型的动态规划题目。

题目要求只能向右或者向下走，也就是说第 (i, j) 个位置只能从左或者上走到。

因而确定从左和从上过来的两条路径哪个最小就可以啦。

不过需要注意，矩阵的上边界和左边界比较特殊。

上边界的每个位置只能从左侧达到；左边界的每个位置只能从上边达到，需要特殊处理。

```
1 class solution:
2     def minPathSum(self, grid: List[List[int]]) -> int:
3
4         for i in range(len(grid)):
5             for j in range(len(grid[0])):
6                 # 矩阵头一个元素不需要处理
7                 if i == 0 and j == 0:
8                     continue
9                 # 上边界元素只能从左侧到达
```

```

10         elif i == 0:
11             grid[i][j] = grid[i][j-1] + grid[i][j]
12             # 左边界元素只能从上侧到达
13         elif j == 0:
14             grid[i][j] = grid[i-1][j] + grid[i][j]
15             # 其他元素取左和上侧路径中较小的那个
16         else:
17             grid[i][j] = min(grid[i][j-1], grid[i-1][j]) + grid[i]
18     [j]
19
20     return grid[-1][-1]
21
22 gird = [[1,3,1],[1,5,1],[4,2,1]]
23 model = Solution()
24 print(model.minPathSum(gird))

```

Day 16 Search in Rotated Sorted Array

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

You are given a target value to search. If found in the array return its index, otherwise return `-1`.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

Example 1:

```

1 Input: nums = [4,5,6,7,0,1,2], target = 0
2 Output: 4

```

Example 2:

```

1 Input: nums = [4,5,6,7,0,1,2], target = 3
2 Output: -1

```

33. 搜索旋转排序数组

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0,1,2,4,5,6,7]` 可能变为 `[4,5,6,7,0,1,2]`)。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 `-1`。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 $O(\log n)$ 级别。

示例 1:

```
1 输入: nums = [4,5,6,7,0,1,2], target = 0
2 输出: 4
```

示例 2:

```
1 输入: nums = [4,5,6,7,0,1,2], target = 3
2 输出: -1
```

解题思路

【变式二分法】

- 如果 $\text{nums}[\text{mid}] < \text{nums}[\text{right}]$ ，即中间的数小于最右边的数，那么 mid （包括 mid ）右边的数是有序的，
- 如果中间的数大于最右边的数，那么 mid （包括 mid ）左边的数是有序的。
- 每次都可以折半进行查找。

```
1 class Solution:
2     def search(self, nums: List[int], target: int) -> int:
3         if not nums:
4             return -1
5
6         i=0
7         j=len(nums)-1
8         while i<=j:
9             mid=(i+j)//2
10            if nums[mid]==target:
11                return mid
12            if nums[i]<=nums[mid]:
13                if nums[i]<=target<nums[mid]:
14                    j=mid-1
15            else:
16                i=mid+1
17        else:
18            if nums[mid]<target<=nums[j]:
19                i=mid+1
20            else:
21                j=mid-1
22        return -1
```

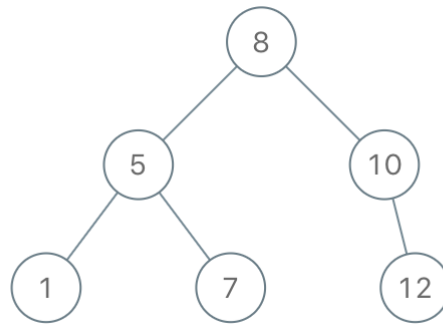
Day 17 Construct Binary Search Tree from Preorder Traversal

Return the root node of a binary **search** tree that matches the given `preorder` traversal.

(Recall that a binary search tree is a binary tree where for every node, any descendant of `node.left` has a value `< node.val`, and any descendant of `node.right` has a value `> node.val`. Also recall that a preorder traversal displays the value of the `node` first, then traverses `node.left`, then traverses `node.right`.)

Example 1:

```
1 Input: [8,5,1,7,10,12]
2 Output: [8,5,10,1,7,null,12]
```



Note:

1. `1 <= preorder.length <= 100`
2. The values of `preorder` are distinct.

1008. 先序遍历构造二叉树

返回与给定先序遍历 `preorder` 相匹配的二叉搜索树 (binary search tree) 的根结点。

(回想一下，二叉搜索树是二叉树的一种，其每个节点都满足以下规则，对于 `node.left` 的任何后代，值总 $< \text{node.val}$ ，而 `node.right` 的任何后代，值总 $> \text{node.val}$ 。此外，先序遍历首先显示节点的值，然后遍历 `node.left`，接着遍历 `node.right`。)

示例:

```
1 输入: [8,5,1,7,10,12]
2 输出: [8,5,10,1,7,null,12]
```

提示:

1. `1 <= preorder.length <= 100`
- 先序 `preorder` 中的值是不同的。

解题思路

- 以数列首位作为根，小于根的数列为左子树，大于根的数列为右子树
- 如果数列为空，则得到最小的子树 `None`

```
1 class Solution:
2     def bstFromPreorder(self, preorder: List[int]) -> TreeNode:
3         if len(preorder) == 0:
4             return None
5         root = TreeNode(preorder[0])
6         preorder_left = [val for val in preorder if val < root.val]
7         preorder_right = [val for val in preorder if val > root.val]
8         root.left = self.bstFromPreorder(preorder_left)
9         root.right = self.bstFromPreorder(preorder_right)
10        return root
```


Day 18 Leftmost Column with at Least a One

(This problem is an *interactive problem*.)

A binary matrix means that all elements are `0` or `1`. For each **individual** row of the matrix, this row is sorted in non-decreasing order.

Given a row-sorted binary matrix `binaryMatrix`, return leftmost column index(0-indexed) with at least a `1` in it. If such index doesn't exist, return `-1`.

You can't access the Binary Matrix directly. You may only access the matrix using a `BinaryMatrix` interface:

- `BinaryMatrix.get(x, y)` returns the element of the matrix at index `(x, y)` (0-indexed).
- `BinaryMatrix.dimensions()` returns a list of 2 elements `[n, m]`, which means the matrix is `n * m`.

Submissions making more than `1000` calls to `BinaryMatrix.get` will be judged *Wrong Answer*. Also, any solutions that attempt to circumvent the judge will result in disqualification.

For custom testing purposes you're given the binary matrix `mat` as input in the following four examples. You will not have access the binary matrix directly.

Example 1:

0	0
1	1

```
1 Input: mat = [[0,0],[1,1]]
2 Output: 0
```

Example 2:

0	0
0	1

```
1 Input: mat = [[0,0],[0,1]]
2 Output: 1
```

Example 3:

0	0
0	0

```
1 Input: mat = [[0,0],[0,0]]
2 Output: -1
```

Example 4:

0	0	0	1
0	0	1	1
0	1	1	1

```
1 Input: mat = [[0,0,0,1],[0,0,1,1],[0,1,1,1]]
2 Output: 1
```

Constraints:

- `1 <= mat.length, mat[i].length <= 100`
- `mat[i][j]` is either `0` or `1`.
- `mat[i]` is sorted in a non-decreasing way.

1428.至少有一个1的最左端列

有道翻译:

(这个问题是一个交互问题。)

一个二进制矩阵意味着所有的元素都是0或1。对于矩阵的每一行，这一行都是非降的顺序。

给定一个行排序的二进制矩阵，返回最左边的列索引(0索引)，其中至少有一个1。如果不存在这样的索引，则返回-1。

你不能直接访问二进制矩阵。您只能使用二进制矩阵接口访问矩阵:

`BinaryMatrix`。 `get(x, y)`返回索引(x, y)(0索引)处的矩阵元素。

`dimensions()`返回一个包含2个元素的列表[n, m]，这意味着该矩阵为n * m。

提交产生超过1000次对`BinaryMatrix`的调用。得到的答案会被判定为错误。此外，任何试图绕过法官的解决办法都将导致取消资格。

为了进行自定义测试，在以下四个示例中为您提供了二进制矩阵`mat`作为输入。您不能直接访问二进制矩阵。

解题思路

【解法1】指针

- 假设有一个指针p从矩阵的右上角元素开始。p只能向左或向下移动。
- 如果p的值为0，则下移。
- 如果p处的值为1，则向左移动。

```
1 # This is BinaryMatrix's API interface.
2 # You should not implement it, or speculate about its implementation
3 # ""
4 #class BinaryMatrix(object):
5 #     def get(self, x: int, y: int) -> int:
```

```

6 # def dimensions(self) -> list[]:
7 class Solution:
8     def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') -> int:
9         Dim = binaryMatrix.dimensions()
10        i = 0
11        j = Dim[1]-1
12        l = 0
13        while i < Dim[0]:
14            if binaryMatrix.get(i,j)==0:
15                i += 1
16                l = j + 1
17            else:
18                if j == 0:
19                    return 0
20                j -= 1
21
22        return -1 if j == Dim[1]-1 else l

```

【解法2】二分查找

- 对每一行执行二分搜索以找到leftmost并更新答案。

```

1 class Solution:
2     def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') -> int:
3
4         x, y = binaryMatrix.dimensions()
5         def seems_legit(column):
6             return any(binaryMatrix.get(i, column) for i in range(x))
7
8         lo = 0
9         hi = y
10
11        while lo < hi:
12            mid = (lo + hi) // 2
13            if seems_legit(mid):
14                hi = mid
15            else:
16                lo = mid + 1
17        return lo if lo < y else -1

```

Day 19 Subarray Sum Equals K

Given an array of integers and an integer **k**, you need to find the total number of continuous subarrays whose sum equals to **k**.

Example 1:

```

1 Input:nums = [1,1,1], k = 2
2 Output: 2

```

Note:

1. The length of the array is in range [1, 20,000].
2. The range of numbers in the array is [-1000, 1000] and the range of the integer **k** is [-1e7, 1e7].

560. 和为K的子数组

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

示例 1：

```
1 输入:nums = [1,1,1], k = 2
2 输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。
```

说明：

数组的长度为 [1, 20,000]。

数组中元素的范围是 [-1000, 1000]，且整数 **k** 的范围是 [-1e7, 1e7]

解题思路

【解法】哈希表

一句话描述就是：

在索引 **i** 和 **j** 处序列的累计和相差 **k**，即 $\text{sum}[i] - \text{sum}[j] = k$ ，则位于索引 **i** 和 **j** 之间的元素之和是 **k**。所有这些中间元素就是要找的子序列啦。

- 另 $\text{sum}[i]$ 表示遍历至元素 **i** 处时的总和
- 每当遇到一个新的 sum 时，在哈希表中创建一个与该 sum 相对应的新条目
- 如果再次出现相同的 sum ，增加与 map 哈希表中的 sum 相对应的计数。
- 当当前 sum 减去目标值 **k** ($\text{cur_sum} - k$) 已经出现在表中时，其对应的值则为满足条件子序列个数，累加之。

```
1 class Solution:
2     def subarraySum(self, nums: List[int], k: int) -> int:
3
4         nums_times = collections.defaultdict(int)
5         nums_times[0] = 1
6         cur_sum = 0
7         res = 0
8
9         for i in nums:
10             cur_sum += i
11             if cur_sum - k in nums_times:
12                 res += nums_times[cur_sum - k]
13
14             nums_times[cur_sum] += 1
15
16         return res
```

Day 20 Bitwise AND of Numbers Range

Given a range $[m, n]$ where $0 \leq m \leq n \leq 2147483647$, return the bitwise AND of all numbers in this range, inclusive.

Example 1:

```
1 | Input: [5,7]
2 | Output: 4
```

Example 2:

```
1 | Input: [0,1]
2 | Output: 0
```

201.数字范围按位与

给定范围 $[m, n]$ ，其中 $0 \leq m \leq n \leq 2147483647$ ，返回此范围内所有数字的按位与（包含 m, n 两端点）。

示例 1:

```
1 | 输入: [5,7]
2 | 输出: 4
```

示例 2:

```
1 | 输入: [0,1]
2 | 输出: 0
```

解题思路

原文链接: <https://leetcode-cn.com/problems/bitwise-and-of-numbers-range/solution/wu-xu-xun-huan-li-yong-shu-xue-si-wei-qiao-miao-po/>

第一：不管多少位相与，只要有一个0结果必为0。

那么9(1001)与5(101)在题中相与的结果是多少呢？

答案是0，因为9相比5多了一位，那么必定[5,9]之间有个8(1000)，所以8与5相与结果为0，根本不需要和其他数进行相与操作。

规律一：if len(bin(n))>len(bin(m))，结果必定为0！

当且仅当len(bin(n))==len(bin(m))的时候，res不为0且bin(res)的长度等于bin(n)。

第二：现在只需要考虑n,m的二进制位数相同的情况了

虽然规律一排除了绝大多数可能导致超时的例题，但直接使用循环还是太慢了。如果不用循环，显然我们应该从n,m上找规律。自然我们想到了两数相减，令sub=n-m。

例如

n=23(10111),m=19(10011),sub=4(100),

也就是m要加4次1.让我们来看看整个过程

第一次: $19+1=10011+1=10100=20$

第二次: $19+2=10011+10=10101=21$

第三次: $19+3=10011+11=10110=22$

第三次: $19+4=10011+100=10111=23$

我们发现 $19+4$ 的过程中必定会产生两次进位(20与22的时候), 所以 $[19,23]$ 的过程中, 最后一位与倒数第二位必定有0位出现。由规律一知结果二进制位数必定为5位, 现在我们可以确定最后两位必定是0。我们不知道倒数第三位会不会产生进位, 但不管是否会产生进位, 倒数第三位+1之后只有两种情况: 原本是0, 变成1, 原本是1, 变成0产生进位, 无论哪种, 必定相与为0。所以, $\text{bin}(\text{res})$ 的后 $\text{bin}(\text{sub})$ 位必定为0。

规律二: $\text{bin}(\text{res})$ 的后 $\text{bin}(\text{sub})$ 位必定为0

第三: 我们已经知道了结果res的位数, 并且可以根据相减数得到某几位的准确数字。那么res的前几位呢?

至于 $\text{bin}(\text{res})$ 的前几个高位, 仍然以19, 23为例子, 其相减为4, 4为三位, 那么在19-23的变化过程中, 前二位至多只发生一次变化, 产生一个进位, 例如19(10011)加4,5,6,7, 加4不变, 加5, 6, 7只变一次。我们不必在乎 $[19,23]$ 的过程中是哪一次发生了进位, 结果的前两位必定等于19, 23两个数字前两位相与。

规律三: $\text{bin}(\text{res})$ 的除开 $\text{bin}(\text{sub})$ 的前几位就是n与m两个数字前几位相与的结果, 与 (n,m) 区间里的数字无关。

【算法】

- 1: 如果 $\text{bin}(n) > \text{bin}(m)$, return 0
- 2: 计算 $\text{bin}(n-m)$ 的位数 len_sub
- 3: 计算 $n \& m$ 的结果
- 4: 将 $n \& m$ 前 $\text{len}(n) - \text{len_sub}$ 位和 len_sub 个0相加, 就是结果

```
1 class Solution:
2     def rangeBitwiseAnd(self, m: int, n: int) -> int:
3         if len(bin(n)) > len(bin(m)):
4             return 0
5         if m == n:
6             return m
7         sub = len(bin(n-m))[2:]
8         return int(bin(m&n)[2:-sub]+'0'*sub,2)
```

Day 21 LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: `get` and `put`.

`get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

`put(key, value)` - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

The cache is initialized with a **positive** capacity.

Follow up:

Could you do both operations in **O(1)** time complexity?

Example:

```
1 LRUCache cache = new LRUCache( 2 /* capacity */ );
2
3 cache.put(1, 1);
4 cache.put(2, 2);
5 cache.get(1);      // returns 1
6 cache.put(3, 3);    // evicts key 2
7 cache.get(2);      // returns -1 (not found)
8 cache.put(4, 4);    // evicts key 1
9 cache.get(1);      // returns -1 (not found)
10 cache.get(3);      // returns 3
11 cache.get(4);      // returns 4
```

146. LRU缓存机制

运用你所掌握的数据结构，设计和实现一个 LRU (最近最少使用) 缓存机制。它应该支持以下操作：获取数据 `get` 和 写入数据 `put`。

获取数据 `get(key)` - 如果关键字 (key) 存在于缓存中，则获取关键字的值（总是正数），否则返回 -1。

写入数据 `put(key, 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。

进阶:

你是否可以在 **O(1)** 时间复杂度内完成这两种操作？

示例:

```
1 LRUCache cache = new LRUCache( 2 /* 缓存容量 */ );
2
3 cache.put(1, 1);
4 cache.put(2, 2);
5 cache.get(1);      // 返回 1
6 cache.put(3, 3);    // 该操作会使得关键字 2 作废
7 cache.get(2);      // 返回 -1 (未找到)
8 cache.put(4, 4);    // 该操作会使得关键字 1 作废
9 cache.get(1);      // 返回 -1 (未找到)
10 cache.get(3);      // 返回 3
11 cache.get(4);      // 返回 4
```

解题思路

【解法1】

```

1  from collections import OrderedDict
2
3  class LRUCache:
4
5      def __init__(self, capacity: int):
6          self.maxsize = capacity
7          self.lrucache = OrderedDict()
8
9      def get(self, key: int) -> int:
10         # 说明在缓存中,重新移动字典的尾部
11         if key in self.lrucache:
12             self.lrucache.move_to_end(key)
13         return self.lrucache.get(key, -1)
14
15
16     def put(self, key: int, value: int) -> None:
17         # 如果存在,删除,重新赋值
18         if key in self.lrucache:
19             del self.lrucache[key]
20         self.lrucache[key] = value
21         if len(self.lrucache) > self.maxsize:
22             # 因为存储空间不够,弹出字典的头部
23             self.lrucache.popitem(last = False)

```

【解法2】

哈希+双向链表

```

1  # 创建双向链表
2  class Node:
3      def __init__(self, key, val):
4          self.key = key
5          self.val = val
6          self.prev = None
7          self.next = None
8
9
10 class LRUCache:
11
12     def __init__(self, capacity: int):
13         # 构建首尾节点,使之相连
14         self.head = Node(0, 0)
15         self.tail = Node(0, 0)
16         self.head.next = self.tail
17         self.tail.prev = self.head
18
19         self.lookup = dict()
20         self.max_len = capacity
21
22     def get(self, key: int) -> int:
23         if key in self.lookup:
24             node = self.lookup[key]
25             self.remove(node)
26             self.add(node)
27         return node.val

```



```

28         else:
29             return -1
30
31     def put(self, key: int, value: int) -> None:
32         if key in self.lookup:
33             self.remove(self.lookup[key])
34         if len(self.lookup) == self.max_len:
35             # 把表头位置节点删除(说明最近的数据值)
36             self.remove(self.head.next)
37         self.add(Node(key, value))
38     # 删除链表节点
39     def remove(self, node):
40         del self.lookup[node.key]
41         node.prev.next = node.next
42         node.next.prev = node.prev
43     # 加在链表尾
44     def add(self, node):
45         self.lookup[node.key] = node
46         pre_tail = self.tail.prev
47         node.next = self.tail
48         self.tail.prev = node
49         pre_tail.next = node
50         node.prev = pre_tail

```

Day 22 Jump Game

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

```

1 | Input: [2,3,1,1,4]
2 | Output: true
3 | Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

```

Example 2:

```

1 | Input: [3,2,1,0,4]
2 | Output: false
3 | Explanation: You will always arrive at index 3 no matter what. Its maximum
4 |               jump length is 0, which makes it impossible to reach the last
   |               index.

```

55. 跳跃游戏

给定一个非负整数数组，你最初位于数组的第一个位置。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个位置。

示例 1:

```
1 输入: [2,3,1,1,4]
2 输出: true
3 解释: 我们可以先跳 1 步, 从位置 0 到达 位置 1, 然后再从位置 1 跳 3 步到达最后一个位置。
```

示例 2:

```
1 输入: [3,2,1,0,4]
2 输出: false
3 解释: 无论如何, 你总会到达索引为 3 的位置。但该位置的最大跳跃长度是 0, 所以你永远不可能到达最后一个位置。
```

解题思路

【贪心】

首先思考一下, 如果判断数组中任意一个位置y是否可以到达呢?

只需要在前序数组中有一个位置x可以到达, 同时从位置x跳跃的最大长度 $x+nums[x] \geq y$, 那么位置y一定可以到达!

换句话说, 对于每一个可以到达的位置 x, 它使得 $x+1, x+2, \dots, x+nums[x]$ 这些连续的位置都可以到达。

因而, 依次遍历数组中的每一个位置, 并实时维护最远可以到达的位置。在遍历的过程中, 如果最远可以到达的位置大于等于数组中的最后一个位置, 则说明最后一个位置可达, 返回 True 作为答案。反之, 如果在遍历结束后, 最后一个位置仍然不可达, 返回 False 作为答案。

```
1 class Solution:
2     def canJump(self, nums: List[int]) -> bool:
3         max_i = 0
4         for i in range(len(nums)):
5             if i <= max_i:
6                 max_i = max(i+nums[i], max_i)
7             if max_i >= len(nums) - 1:
8                 return True
9
10        return False
```

【反向思路】思路同上, 反向寻找更快

```
1 class Solution:
2     def canJump(self, nums: List[int]) -> bool:
3         N = len(nums)
4         target = N-1
5         for i in range(N-1, -1, -1):
6             if i + nums[i] >= target:
7                 target = i
8
9         return target == 0
```

Day 23 Longest Common Subsequence

Given two strings `text1` and `text2`, return the length of their longest common subsequence.

A *subsequence* of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters. (eg, "ace" is a subsequence of "abcde" while "aec" is not). A *common subsequence* of two strings is a subsequence that is common to both strings.

If there is no common subsequence, return 0.

Example 1:

```
1 Input: text1 = "abcde", text2 = "ace"
2 Output: 3
3 Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
1 Input: text1 = "abc", text2 = "abc"
2 Output: 3
3 Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
1 Input: text1 = "abc", text2 = "def"
2 Output: 0
3 Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- `1 <= text1.length <= 1000`
- `1 <= text2.length <= 1000`
- The input strings consist of lowercase English characters only.

1143. 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。

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。

若这两个字符串没有公共子序列，则返回 0。

示例 1:

```
1 输入: text1 = "abcde", text2 = "ace"
2 输出: 3
3 解释: 最长公共子序列是 "ace"，它的长度为 3。
```

示例 2:

```
1 输入: text1 = "abc", text2 = "abc"
2 输出: 3
3 解释: 最长公共子序列是 "abc", 它的长度为 3。
```

示例 3:

```
1 输入: text1 = "abc", text2 = "def"
2 输出: 0
3 解释: 两个字符串没有公共子序列, 返回 0。
```

提示:

```
1 <= text1.length <= 1000
```

```
1 <= text2.length <= 1000
```

输入的字符串只含有小写英文字符。

解题思路

【动态规划】

- 考虑dp table, 对于0位置未添加空字符串
- $dp[i][j]$ 表示 $s1[0...i-1]$ 与 $s2[0...j-1]$ 的最长公共子序列的长度
- 当 $s1[i]==s2[j]$,说明这两个字符是公共字符, 只要考察其子问题, $dp[i-1][j-1]$ 的长度即可, 在此基础上+1,
- 当 $s1[i]!=s2[j]$,说明这两个字符不是公共的字符, 只要考察其两个子问题, $dp[i-1][j]$, $dp[i][j-1]$ 取max
- 结果为 $dp[m-1][n-1]$ (或 $dp[-1][-1]$)

```
1 class Solution:
2     def longestCommonSubsequence(self, text1: str, text2: str) -> int:
3         m, n = len(text1), len(text2)
4         # 构建dp table, base case 为 0
5         dp_table = [[0] * (n + 1) for _ in range(m + 1)]
6
7         for i in range(1, m+1):
8             for j in range(1, n+1):
9                 if text1[i-1] == text2[j-1]:
10                     dp_table[i][j] = dp_table[i-1][j-1] + 1
11                 else:
12                     dp_table[i][j] = max(dp_table[i-1][j], dp_table[i][j-1])
13
14         return dp_table[-1][-1]
```

Day 24 Maximal Square

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

Example:

```
1 Input:
2
3 1 0 1 0 0
4 1 0 1 1 1
5 1 1 1 1 1
6 1 0 0 1 0
7
8 Output: 4
```

221.最大正方形

在一个由 0 和 1 组成的二维矩阵内，找到只包含 1 的最大正方形，并返回其面积。

示例:

```
1 输入:
2
3 1 0 1 0 0
4 1 0 1 1 1
5 1 1 1 1 1
6 1 0 0 1 0
7
8 输出: 4
```

解题思路

【动态规划】

今天又是一道典型的动态规划应用。

- 初始化一个0矩阵 dp_table，维数和原始矩阵维数相同；
- dp(i,j) 表示的是由 1 组成的最大正方形的边长；
- base case 初始化与原matrix一致（第一行/第一列）
- 对原始矩阵中的每一个 1，将当前元素的值更新为

$dp(i, j) = \min(dp(i-1, j), dp(i-1, j-1),$

$dp(i, j-1)) + 1$

- 用一个另变量记录当前出现的最大边长max_l，这样遍历一次，找到最大的正方形边长max_l^2

```
1 class Solution:
2     def maximalSquare(self, matrix: List[List[str]]) -> int:
3         if not matrix:
4             return 0
5
6         r = len(matrix)
7         h = len(matrix[0])
8         dp_table = [[0]*h for i in range(r)]
9         max_l = 0
10
11        for i in range(r):
12            dp_table[i][0] = int(matrix[i][0])
13            max_l = max(max_l, dp_table[i][0])
```

```

14         for j in range(h):
15             dp_table[0][j] = int(matrix[0][j])
16             max_l = max(max_l, dp_table[0][j])
17
18         for i in range(1, r):
19             for j in range(1, h):
20                 if int(matrix[i][j]) == 1:
21                     dp_table[i][j] = min(dp_table[i-1][j], dp_table[i][j-1], dp_table[i-1][j-1]) + 1
22                     max_area = max(l_area, dp_table[i][j])
23                 else:
24                     dp_table[i][j] = 0
25
26         return max_l**2

```

- 时间复杂度：O(mn)。
- 空间复杂度：O(mn)。

Day 25 First Unique Number

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the `FirstUnique` class:

- `FirstUnique(int[] nums)` Initializes the object with the numbers in the queue.
- `int showFirstUnique()` returns the value of **the first unique** integer of the queue, and returns `-1` if there is no such integer.
- `void add(int value)` insert value to the queue.

Example 1:

```

1  Input:
2  ["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","add","showFirstUnique"]
3  [[2,3,5],[],[5],[2],[3],[]]
4  Output:
5  [null,2,null,2,null,3,null,-1]
6
7  Explanation:
8  FirstUnique firstUnique = new FirstUnique([2,3,5]);
9  firstUnique.showFirstUnique(); // return 2
10 firstUnique.add(5);             // the queue is now [2,3,5,5]
11 firstUnique.showFirstUnique(); // return 2
12 firstUnique.add(2);             // the queue is now [2,3,5,5,2]
13 firstUnique.showFirstUnique(); // return 3
14 firstUnique.add(3);             // the queue is now [2,3,5,5,2,3]
15 firstUnique.showFirstUnique(); // return -1

```

Example 2:

```

1  Input:

```

```

2 ["FirstUnique","showFirstUnique","add","add","add","add","add","showFirstUn
  ique"]
3 [[7,7,7,7,7,7]],[],[7],[3],[3],[7],[17],[]
4 Output:
5 [null,-1,null,null,null,null,null,17]
6
7 Explanation:
8 FirstUnique firstUnique = new FirstUnique([7,7,7,7,7,7]);
9 firstUnique.showFirstUnique(); // return -1
10 firstUnique.add(7);           // the queue is now [7,7,7,7,7,7]
11 firstUnique.add(3);           // the queue is now [7,7,7,7,7,7,3]
12 firstUnique.add(3);           // the queue is now [7,7,7,7,7,7,3,3]
13 firstUnique.add(7);           // the queue is now [7,7,7,7,7,7,3,3,7]
14 firstUnique.add(17);          // the queue is now [7,7,7,7,7,7,3,3,7,17]
15 firstUnique.showFirstUnique(); // return 17

```

Example 3:

```

1 Input:
2 ["FirstUnique","showFirstUnique","add","showFirstUnique"]
3 [[[809]],[],[809],[]]
4 Output:
5 [null,809,null,-1]
6
7 Explanation:
8 FirstUnique firstUnique = new FirstUnique([809]);
9 firstUnique.showFirstUnique(); // return 809
10 firstUnique.add(809);         // the queue is now [809,809]
11 firstUnique.showFirstUnique(); // return -1

```

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 108`
- `1 <= value <= 108`
- At most `50000` calls will be made to `showFirstUnique` and `add`.

1429. 首个唯一的数字

有道翻译:

您有一个整数队列，您需要检索队列中的第一个唯一整数。

实现“FirstUnique”类:

- 'FirstUnique(int[] nums)' 用队列中的数字初始化对象。
- 'int showFirstUnique()' 返回队列的第一个唯一的整数的值，如果没有这样的整数，返回-1。
- 'void add(int value)' 向队列插入值。

解题思路

【HashMap + queue】

- 字典存储数字出现的次数
- 队列首位存储首个仅出现一次的数字

```

1 from collections import deque, defaultdict
2 class FirstUnique:
3
4     def __init__(self, nums: List[int]):
5         self.queue = deque()
6         self.hashmap = defaultdict(int)
7         for num in nums:
8             self.queue.append(num)
9             if num in self.hashmap:
10                 self.hashmap[num] += 1
11             else:
12                 self.hashmap[num] = 1
13
14     def showFirstUnique(self) -> int:
15         while self.queue and self.hashmap[self.queue[0]] > 1:
16             self.queue.popleft()
17
18         if len(self.queue) == 0:
19             return -1
20         else:
21             return self.queue[0]
22
23     def add(self, value: int) -> None:
24         self.hashmap[value] += 1
25         if self.hashmap[value] == 1:
26             self.queue.append(value)
27
28 # Your FirstUnique object will be instantiated and called as such:
29 # obj = FirstUnique(nums)
30 # param_1 = obj.showFirstUnique()
31 # obj.add(value)

```

Day 26 Binary Tree Maximum Path Sum

Given a **non-empty** binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

Example 1:

```

1 Input: [1,2,3]
2
3     1
4    / \
5   2   3
6
7 Output: 6

```

Example 2:


```

1 | Input: [-10,9,20,null,null,15,7]
2 |
3 |     -10
4 |    /  \
5 |   9    20
6 |  /  \
7 | 15   7
8 |
9 | Output: 42

```

124. 二叉树中的最大路径和

给定一个非空二叉树，返回其最大路径和。

本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径至少包含一个节点，且不一定经过根节点。

示例 1:

输入: [1,2,3]

```

1 |     1
2 |    / \
3 |   2   3

```

输出: 6

示例 2:

输入: [-10,9,20,null,null,15,7]

```

-10
 / \
9  20
 / \
15 7

```

输出: 42

解题思路

【递归】

```

1 | # Definition for a binary tree node.
2 | # class TreeNode:
3 | #     def __init__(self, val=0, left=None, right=None):
4 | #         self.val = val
5 | #         self.left = left
6 | #         self.right = right
7 |
8 | class Solution:
9 |     def dfs_sum(self, root: TreeNode):
10 |         if root == None: return 0
11 |         val = root.val
12 |         sum_l = max(0, self.dfs_sum(root.left))
13 |         sum_r = max(0, self.dfs_sum(root.right))
14 |         self.ans = max(self.ans, sum_l + sum_r + val)

```

```

15         return max(sum_l , sum_r) + val
16
17     def maxPathSum(self, root: TreeNode) -> int:
18         self.ans = - 1e9
19         self.dfs_sum(root)
20         return self.ans

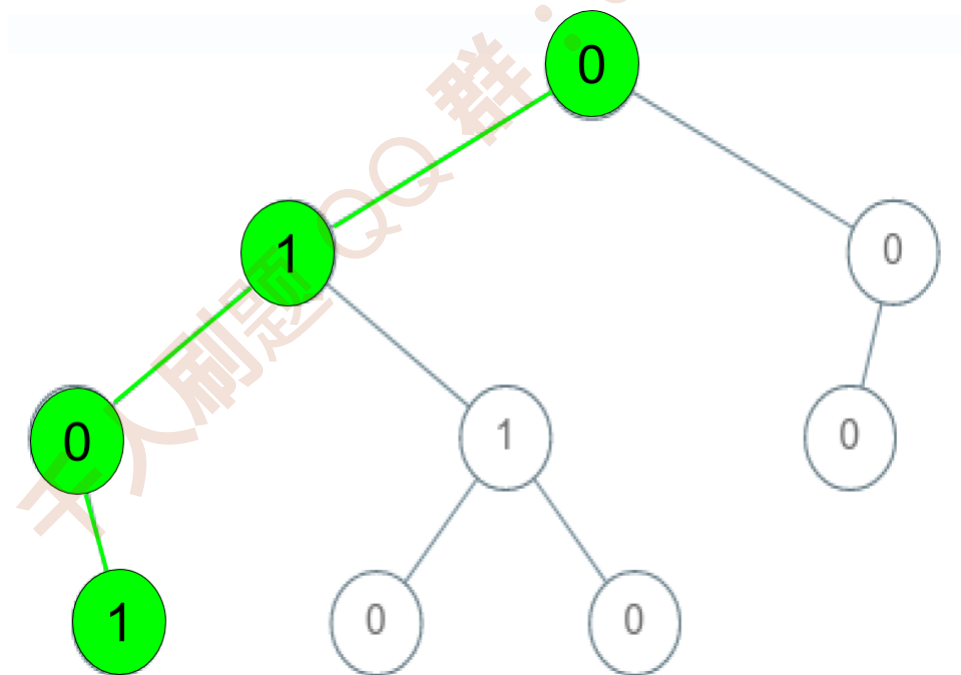
```

Day 27 Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

Given a binary tree where each path going from the root to any leaf form a **valid sequence**, check if a given string is a **valid sequence** in such binary tree.

We get the given string from the concatenation of an array of integers `arr` and the concatenation of all values of the nodes along a path results in a **sequence** in the given binary tree.

Example 1:

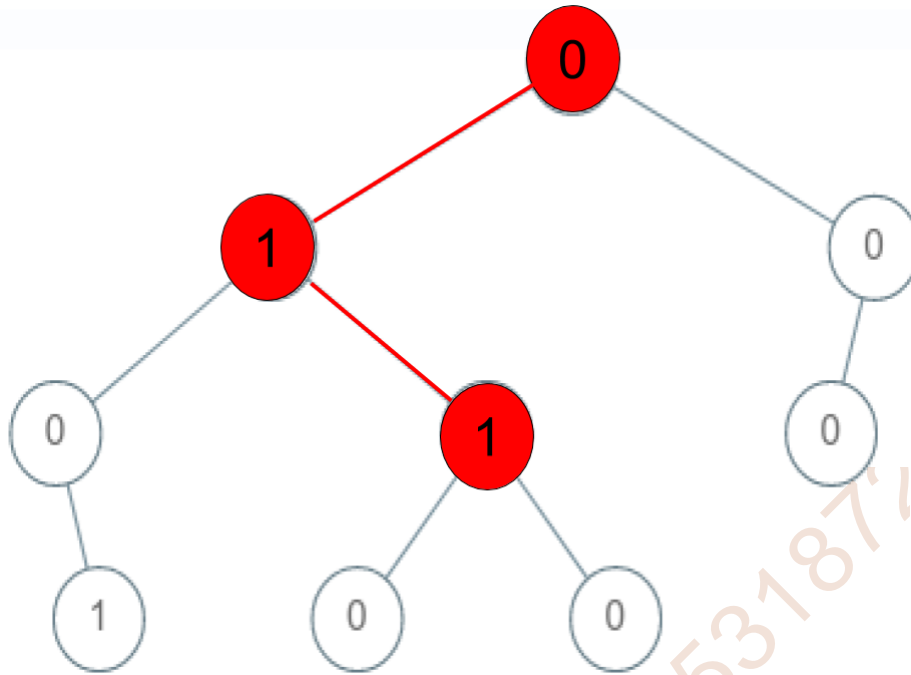


```

1 Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1]
2 Output: true
3 Explanation:
4 The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure).
5 Other valid sequences are:
6 0 -> 1 -> 1 -> 0
7 0 -> 0 -> 0

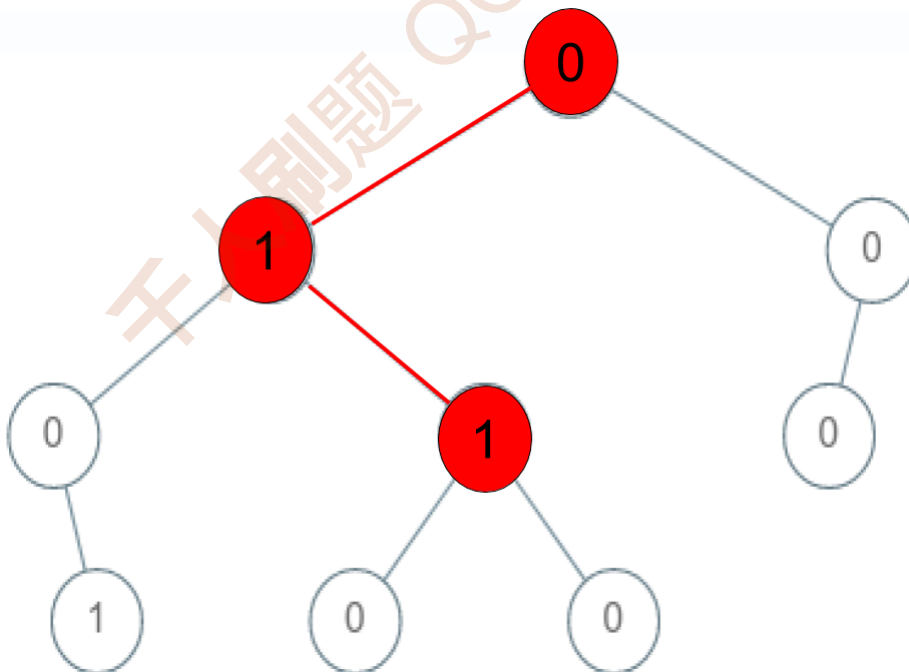
```

Example 2:



- 1 Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,0,1]
- 2 Output: false
- 3 Explanation: The path 0 -> 0 -> 1 does not exist, therefore it is not even a sequence.

Example 3:



- 1 Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,1]
- 2 Output: false
- 3 Explanation: The path 0 -> 1 -> 1 is a sequence, but it is not a valid sequence.

Constraints:

- `1 <= arr.length <= 5000`
- `0 <= arr[i] <= 9`
- Each node's value is between [0 - 9].

1430.检查字符串是否是二叉树中从根到叶路径的有效序列

有道翻译:

给定一个二叉树，其中从根到任何叶的每条路径都构成一个有效序列，检查给定的字符串在这个二叉树中是否是一个有效序列。

我们从一个整数数组arr的串联中得到给定的字符串，而沿着一条路径的所有节点值的串联则得到给定二叉树中的一个序列。

解题思路

【暴力】

- 寻找二叉树的所有路径，然后判断

```
1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def isValidSequence(self, root: TreeNode, arr: List[int]) -> bool:
9
10         def construct_paths(root, path):
11             if root:
12                 path += str(root.val)
13                 if not root.left and not root.right: # 当前节点是叶子节点
14                     paths.append(path) # 把路径加入到答案中
15                 else:
16                     # 当前节点不是叶子节点，继续递归遍历
17                     construct_paths(root.left, path)
18                     construct_paths(root.right, path)
19
20         paths = []
21         construct_paths(root, '')
22         s = ""
23         for item in arr:
24             s = s + str(item)
25         return s in paths
```

千人刷题QQ群: 613531871