五月算法精选合集

整理人：千紫

题目出处： leetcode 五月挑战计划

题解发表公众号：迪乐阅读

刷题组织公众号：闪电内推

加入成员 5000+ 的字节内推群和成员 2000+ 的刷题群方法

- 搜索字节技术小哥 Dean 的微信 1014247150， 进入其朋友圈并跳入微信群
- QQ 刷题群：613531872

这里有你学习、求职的一系列资料、内推福利，全部无条件赠送！

Day 1 —— First Bad Version

首个坏“蛋”

问题描述 You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which will return whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example:

```
Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true

Then 4 is the first bad version.
```

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。

假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。

你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。

示例:

```
给定 n = 5，并且 version = 4 是第一个错误的版本。

调用 isBadVersion(3) -> false
调用 isBadVersion(5) -> true
```

```
调用 isBadVersion(4) -> true

所以，4 是第一个错误的版本。
```

解题思路

【典型二分法】

- 没啥好说的，典型二分法~

```python
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution:
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        hi = n
        lo = 1

        while lo <= hi:
            crr = (hi + lo) // 2
            if isBadVersion(crr):
                hi = crr - 1
            else:
                lo = crr + 1

        return lo
```

时间复杂度O(logn)，空间复杂度O(1)。

二分查找模板

```python
def binary_search(list,item):
    low = 0
    high = len(list) - 1
    while low <= high:
        mid = int((low + high) / 2)
        guess = list[mid]
        if guess == item:
            return mid
        if guess > item:
            high = mid - 1
        else:
            low = mid + 1
```

Day 2 —— Jewels and Stones

石头和宝石

问题描述

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"` .

Example 1:

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note:

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

给定字符串J 代表石头中宝石的类型，和字符串 S代表你拥有的石头。 S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

J 中的字母不重复，J 和 S中的所有字符都是字母。字母区分大小写，因此"a"和"A"是不同类型的石头。

示例 1:

```
输入: J = "aA", S = "aAAbbbb"
输出: 3
```

示例 2:

```
输入: J = "z", S = "ZZ"
输出: 0
```

注意:

S 和 J 最多含有50个字母。 J 中的字符不重复。 **解题思路**

时间/空间复杂度的平衡

【哈希集合】

今天题目比较简单，使用哈希表降低时间复杂度的同时增加了空间复杂度：

```python
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        fortune = 0
        # 将宝石集合转换为字典，查找复杂度为O(1)
        set_J = set(J)
        for s in S:
            if s in set_J:
                fortune += 1

        return fortune
```

时间复杂度O(J.length + S.length)，空间复杂度O(J.length)。

【注】时间复杂度中O(J.length）来自于创建哈希表

Day 3 —— Ransom Note

赎金券

问题描述

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note: You may assume that both strings contain only lowercase letters.

```
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

给定一个赎金信 (ransom) 字符串和一个杂志(magazine)字符串，判断第一个字符串 ransom 能不能由第二个字符串 magazines 里面的字符构成。如果可以构成，返回 true ； 否则返回 false。

(题目说明：为了不暴露赎金信字迹，要从杂志上搜索各个需要的字母，组成单词来表达意思。杂志字符串中的每个字符只能在赎金信字符串中使用一次。)

注意：

你可以假设两个字符串均只含有小写字母。

```
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

解题思路

【字典】

Counter（计数器）：用于追踪值的出现次数

Counter类继承dict类，能使用dict类里面的方法

```
from collections import Counter
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        dic = Counter(magazine)
        for i in ransomNote:
            if i not in dic or dic[i] == 0:
                return False
            else:
                dic[i] -= 1
        return True
```

【字符串替换】

```
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        for item in ransomNote:
            if not item in magazine:
                return False
            magazine = magazine.replace(item, "", 1)
        return True
```

Day 4 —— Number Complement

数字补码

问题描述

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Example 1:

```
Input: 5
Output: 2
Explanation: The binary representation of 5 is 101 (no leading zero bits), and its
complement is 010. So you need to output 2.
```

Example 2:

```
Input: 1
Output: 0
Explanation: The binary representation of 1 is 1 (no leading zero bits), and its
complement is 0. So you need to output 0.
```

Note:

1. The given integer is guaranteed to fit within the range of a 32-bit signed integer.
2. You could assume no leading zero bit in the integer’s binary representation.
3. This question is the same as 1009: https://leetcode.com/problems/complement-of-base-10-integer/

给定一个正整数，输出它的补数。补数是对该数的二进制表示取反。

示例 1:

```
输入：5
输出：2
解释：5 的二进制表示为 101（没有前导零位），其补数为 010。所以你需要输出 2 。
```

示例 2:

```
输入：1
输出：0
解释：1 的二进制表示为 1（没有前导零位），其补数为 0。所以你需要输出 0 。
```

注意:

给定的整数保证在 32 位带符号整数的范围内。 你可以假定二进制数不包含前导零位。

解题思路

【异或】

按照题目要求，最直观的思路就是与1异或。

```
class Solution:
    def findComplement(self, num: int) -> int:
        if num == 0: return 1
        t = num
        res = 0
        while t > 0:
            t = t >> 1
            res = res << 1
```

```
            res += 1
        return res ^ num
```

Day 5 —— First Unique Character in a String

字符串中首个唯一字符

问题描述

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

Note: You may assume the string contain only lowercase letters.

给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

示例:

```
s = "leetcode"
返回 0

s = "loveleetcode"
返回 2
```

提示：你可以假定该字符串只包含小写字母。

解题思路

【字典——计数】

Counter（计数器）：用于追踪值的出现次数。

Counter类继承dict类，能使用dict类里面的方法

```
class Solution:
    def firstUniqChar(self, s: str) -> int:
        if not s:
            return -1

        set_s = Counter(s)

        for i in range(len(s)):
            if set_s[s[i]] == 1:
                return i

        return -1
```

时间复杂度O(N)，线性遍历两次。

空间复杂度O(N)。

Day 6 —— Majority Element

多数元素

问题描述

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

Example 1:

```
Input: [3,2,3]
Output: 3
```

Example 2:

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

给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数大于 ⌊ n/2 ⌋ 的元素。

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: [3,2,3]
输出: 3
```

示例 2:

```
输入: [2,2,1,1,1,2,2]
输出: 2
```

解题思路

【哈希】

跟昨天一样使用Counter计数器。

```
class Solution:
      def majorityElement(self, nums):
          counts = collections.Counter(nums)
          return max(counts.keys(), key=counts.get)
```

时间复杂度O(N^2)。

空间复杂度O(N)。

Day 7 —— Cousins in Binary Tree

二叉树的堂兄弟

问题描述

In a binary tree, the root node is at depth `0`, and children of each depth `k` node are at depth `k+1`.

Two nodes of a binary tree are *cousins* if they have the same depth, but have **different parents**.

We are given the `root` of a binary tree with unique values, and the values `x` and `y` of two different nodes in the tree.

Return `true` if and only if the nodes corresponding to the values `x` and `y` are cousins.

Example 1:

```
Input: root = [1,2,3,4], x = 4, y = 3
Output: false
```

Example 2

```
Input: root = [1,2,3,null,4,null,5], x = 5, y = 4
Output: true
```

Example 3:

```
Input: root = [1,2,3,null,4], x = 2, y = 3
Output: false
```

Note:

1. The number of nodes in the tree will be between `2` and `100`.
2. Each node has a unique integer value from `1` to `100`.

在二叉树中，根节点位于深度 0 处，每个深度为 k 的节点的子节点位于深度 k+1 处。

如果二叉树的两个节点深度相同，但父节点不同，则它们是一对堂兄弟节点。

我们给出了具有唯一值的二叉树的根节点 root，以及树中两个不同节点的值 x 和 y。

只有与值 x 和 y 对应的节点是堂兄弟节点时，才返回 true。否则，返回 false。

示例 1：

```
输入：root = [1,2,3,4], x = 4, y = 3
输出：false
```

示例 2：

```
输入：root = [1,2,3,null,4,null,5], x = 5, y = 4
输出：true
```

示例 3：

```
输入：root = [1,2,3,null,4], x = 2, y = 3
输出：false
```

提示：

二叉树的节点数介于 2 到 100 之间。 每个节点的值都是唯一的、范围为 1 到 100 的整数。

解题思路

【哈希表】

根据题目要求“堂兄弟”同深度不同父母，考虑深度遍历确定各Node父母和深度即可：

```
# 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 isCousins(self, root: TreeNode, x: int, y: int) -> bool:
        parent = {}
        depth = {}

        def dfs(node,par=None):
            if node:
                parent[node.val] = par
                depth[node.val] = 1 + depth[par.val] if par else 0
                dfs(node.left,node)
                dfs(node.right,node)

        dfs(root)
        return depth[x] == depth[y] and parent[x] != parent[y]
```

时间复杂度O(N)。

空间复杂度O(N)。

【简化】上述代码中需要把所有节点都遍历一遍，但实际上只需要找到对应x和y的父母和深度即可：

```
class Solution(object):
    def isCousins(self, root, x, y):
        lookup = {}
        def dfs(root, depth=0, par=None):
            if root:
                if root.val in (x, y):
                    lookup[root.val] = (depth, par)
                dfs(root.left, depth=depth+1, par=root.val)
                dfs(root.right, depth=depth+1, par=root.val)
        dfs(root)
        return lookup[x][0] == lookup[y][0] and lookup[x][1] != lookup[y][1]
```

Day 8 —— Check If It Is a Straight Line

这是一条直线吗

问题描述

You are given an array `coordinates`, `coordinates[i] = [x, y]`, where `[x, y]` represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:

```
Input: coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]
Output: true
```

Example 2:

```
Input: coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]
Output: false
```

Constraints:

- `2 <= coordinates.length <= 1000`
- `coordinates[i].length == 2`
- `-10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4`
- `coordinates` contains no duplicate point.

在一个 XY 坐标系中有一些点，我们用数组 coordinates 来分别记录它们的坐标，其中 coordinates[i] = [x, y] 表示横坐标为 x、纵坐标为 y 的点。

请你来判断，这些点是否在该坐标系中属于同一条直线上，是则返回 true，否则请返回 false。

提示：

`2 <= coordinates.length <= 1000` `coordinates[i].length == 2`
`-10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4` `coordinates` 中不含重复的点

解题思路

【一次函数】

简单来说就是计算任意两点之间斜率是否相同，需要注意以下平行于坐标轴（x/y轴）的特殊情况。

```
class Solution:
    def checkStraightLine(self, coordinates: List[List[int]]) -> bool:
        if coordinates[0][0]-coordinates[1][0] == 0:
            m = 0
        else:
            m = (coordinates[0][1]-coordinates[1][1])/(coordinates[0][0]-
coordinates[1][0])
        c = coordinates[0][1] - m*coordinates[0][0]

        for i in range(2,len(coordinates)):
            if coordinates[i][1] != m*coordinates[i][0] + c:
                return False
        return True
```

Day 9 —— Valid Perfect Square

有效完全平方数

问题描述

Given a positive integer *num*, write a function which returns True if *num* is a perfect square else False.

Note: Do not use any built-in library function such as `sqrt` .

Example 1:

```
Input: 16
Output: true
```

Example 2:

```
Input: 14
Output: false
```

给定一个正整数 num，编写一个函数，如果 num 是一个完全平方数，则返回 True，否则返回 False。

说明：不要使用任何内置的库函数，如 sqrt。

示例 1：

```
输入：16
输出：True
```

示例 2：

```
输入：14
输出：False
```

解题思路

【二分查找】

二分查找的下界为 0，上界可以粗略地设定为 *num*。

```
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        lo = 1
        hi = num

        while num != 0 and lo <= hi:
            mid = (lo + hi) // 2
            if mid ** 2 == num:
                return True
            elif mid ** 2 > num:
                hi = mid - 1
            else:
                lo = mid + 1

        return False
```

Day 10 —— Find the Town Judge

寻找法官

问题描述

In a town, there are `N` people labelled from `1` to `N`. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given `trust`, an array of pairs `trust[i] = [a, b]` representing that the person labelled `a` trusts the person labelled `b`.

If the town judge exists and can be identified, return the label of the town judge. Otherwise, return `-1`.

Example 1:

```
Input: N = 2, trust = [[1,2]]
Output: 2
```

Example 2:

```
Input: N = 3, trust = [[1,3],[2,3]]
Output: 3
```

Example 3:

```
Input: N = 3, trust = [[1,3],[2,3],[3,1]]
Output: -1
```

Example 4:

```
Input: N = 3, trust = [[1,2],[2,3]]
Output: -1
```

Example 5:

```
Input: N = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]]
Output: 3
```

Note:

1. `1 <= N <= 1000`
2. `trust.length <= 10000`
3. `trust[i]` are all different
4. `trust[i][0] != trust[i][1]`
5. `1 <= trust[i][0], trust[i][1] <= N`

解题思路

*【*哈希表*】今天是一道阅读理解题~

小镇中可能有一位法官：

1. 法官不相信任何人
2. 除了法官以外每个人都相信法官

从算法的角度来考虑，只要统计两点：

1. 小镇中是否有一个人A不相信任何人（在List[0]中不存在的人）
2. 小镇中N-1个人都相信A吗？

```python
import collections
class Solution:
    def findJudge(self, N: int, trust: List[List[int]]) -> int:
        if N == 1 and not trust:
            return 1
        # 需要注意defaultdict的用法
        dic = collections.defaultdict(list)
        peo = [x[0] for x in trust]

        for i in range(len(trust)):
            if dic[trust[i][1]]:
                dic[trust[i][1]] += 1
```

```
            else:
                dic[trust[i][1]] = 1
            if dic[trust[i][1]] == N - 1 and trust[i][1] not in peo:
                return trust[i][1]

        return -1
```

Day 11 —— Flood Fill

图像渲染

问题描述

An `image` is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

有一幅以二维整数数组表示的图画，每一个整数表示该图画的像素值大小，数值在 0 到 65535 之间。

Given a coordinate `(sr, sc)` representing the starting pixel (row and column) of the flood fill, and a pixel value `newColor`, "flood fill" the image.

给你一个坐标 (sr, sc) 表示图像渲染开始的像素值（行，列）和一个新的颜色值 newColor，让你重新上色这幅图像。

To perform a "flood fill", consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the newColor.

为了完成上色工作，从初始坐标开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重复该过程。将所有有记录的像素点的颜色值改为新的颜色值。

At the end, return the modified image.

最后返回经过上色渲染后的图像。

Example 1:

```
Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation:
From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected
by a path of the same color as the starting pixel are colored with the new color.
Note the bottom corner is not colored 2, because it is not 4-directionally connected
to the starting pixel.在图像的正中间，(坐标(sr,sc)=(1,1)),在路径上所有符合条件的像素点的颜色都被更改成2。注意，右下角的像素没有更改为2，因为它不是在上下左右四个方向上与初始点相连的像素点。
```

Note:

- The length of `image` and `image[0]` will be in the range `[1, 50]`.
- The given starting pixel will satisfy `0 <= sr < image.length` and `0 <= sc < image[0].length`.
- The value of each color in `image[i][j]` and `newColor` will be an integer in `[0, 65535]`.

解题思路

【深度优先搜索+递归】

```
class Solution(object):
    def floodFill(self, image, sr, sc, newColor):
```

```
        R, C = len(image), len(image[0])
        color = image[sr][sc]
        if color == newColor: return image
        def dfs(r, c):
            if image[r][c] == color:
                image[r][c] = newColor
                if r >= 1: dfs(r-1, c)
                if r+1 < R: dfs(r+1, c)
                if c >= 1: dfs(r, c-1)
                if c+1 < C: dfs(r, c+1)

        dfs(sr, sc)
        return image
```

Day 12 —— Single Element in a Sorted Array

有序数组中的单一元素

问题描述

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once. Find this single element that appears only once.

Example 1:

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

Example 2:

```
Input: [3,3,7,7,10,11,11]
Output: 10
```

Note: Your solution should run in O(log n) time and O(1) space.

解题思路

【变体二分法】

每次需要判断mid左边和右边哪个元素于mid相同。

然后判断消除mid相同元素后剩余左/右序列哪个长度为奇数，唯一的元素一定出现在奇数序列中。

```
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        lo = 0
        hi = len(nums) - 1

        while lo < hi:
            mid = (hi + lo) // 2
            if nums[mid-1] == nums[mid]:
                if (mid - 1 - lo) % 2 == 1:
                    hi = mid - 2
                else:
                    lo = mid + 1
            elif nums[mid + 1] ==nums[mid]:
                if (hi - mid -1) % 2 == 1:
                    lo = mid + 2
```

```
                else:
                    hi = mid -1
            else:
                return nums[mid]

        return nums[lo]
```

- 时间复杂度：O(logN)。
- 空间复杂度：O(1)。

【数学计算法使用set()求和做差。

set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等

```
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        return sum(set(nums))*2-sum(nums)
```

Day 14 —— Remove K Digits

移掉K位数字

问题描述

Given a non-negative integer *num* represented as a string, remove *k* digits from the number so that the new number is the smallest possible.

Note:

- The length of *num* is less than 10002 and will be ≥ *k*.
- The given *num* does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is
the smallest.
```

Example 2:

```
Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output must
not contain leading zeroes.
```

Example 3:

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing which
is 0.
```

解题思路

*【贪心】*利用栈来实现：

对于每个数字，如果该数字小于栈顶部，即该数字的左邻居，则弹出堆栈，即删除左邻居。否则，把数字推到栈上。

```python
class Solution:
    def removeKdigits(self, num: str, k: int) -> str:
        numstack = []

        for digit in num:
            while k and numstack and numstack[-1] > digit:
                numstack.pop()
                k -= 1

            numstack.append(digit)
        finalnum = numstack[:-k] if k else numstack
        # 可能还有一些前导零。要格式化最后的数字，我们需要去掉前导零。
        # 如果从序列中删除所有的数字，返回“0”
        return "".join(finalnum).lstrip('0') or "0"
```

Day 14 —— Implement Trie (Prefix Tree)

前缀树

问题描述

Implement a trie with `insert`, `search`, and `startsWith` methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // returns true
trie.search("app");     // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");     // returns true
```

Note:

- You may assume that all inputs are consist of lowercase letters `a-z`.
- All inputs are guaranteed to be non-empty strings.

解题思路

【字典迭代】

```python
class Trie:
    def __init__(self):
        self.d = {}

    def insert(self, word: str) -> None:
        t = self.d

        for c in word:
            if c not in t:
                t[c] = {}
            t = t[c]        #字典迭代
        t['end'] = True     #标记单词终点

    def search(self, word: str) -> bool:
        t = self.d
```

```
            for c in word:
                if c not in t:
                    return False
                t = t[c]
            return 'end' in t

        def startsWith(self, prefix: str) -> bool:
            t = self.d
            for c in prefix:
                if c not in t:
                    return False
                t = t[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)
```

Day 15 —— Maximum Sum Circular Subarray

环形子数组最大的和

问题描述

Given a **circular array C** of integers represented by `A`, find the maximum possible sum of a non-empty subarray of **C**.

Here, a *circular array* means the end of the array connects to the beginning of the array. (Formally, `C[i] = A[i]` when `0 <= i < A.length`, and `C[i+A.length] = C[i]` when `i >= 0`.)

Also, a subarray may only include each element of the fixed buffer `A` at most once. (Formally, for a subarray `C[i], C[i+1], ..., C[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % A.length = k2 % A.length`.)

Example 1:

```
Input: [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3
```

Example 2:

```
Input: [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10
```

Example 3:

```
Input: [3,-1,2,-1]
Output: 4
Explanation: Subarray [2,-1,3] has maximum sum 2 + (-1) + 3 = 4
```

Example 4:

```
Input: [3,-2,2,-3]
Output: 3
Explanation: Subarray [3] and [3,-2,2] both have maximum sum 3
```

Example 5:

```
Input: [-2,-3,-1]
Output: -1
Explanation: Subarray [-1] has maximum sum -1
```

Note:

1. -30000 <= A[i] <= 30000
2. 1 <= A.length <= 30000

解题思路

【Kadane算法】

给定数组 A，Kadane 算法可以用来找到 A 的最大子段和。

可以把问题的解分为两种情况。（1）解没有跨过A[n-1]到A[0]（原问题）。（2）解跨过A[n-1]到A[0]。

```python
class Solution:
    def maxSubarraySumCircular(self, A: List[int]) -> int:
        res1 = 0
        res2 = 0
        res = 0
        local_max = 0

        for i in A:
            if i < local_max + i:
                local_max = local_max + i
            else:
                local_max = i
            res1 = max(res1, local_max)
        if res1 == 0:
            res = max(A)
            return res

        local_min = 0
        for i in A:
            if local_min + i >= 0:
                local_min = 0
            else:
                local_min += i

            res2 = min(res2, local_min)

        res = max(res1, sum(A) - res2)
        return res
```

Day 16 —— Odd Even Linked List

奇偶链表

问题描述

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example 1:

```
Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
```

Example 2:

```
Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
```

Note:

- The relative order inside both the even and odd groups should remain as it was in the input.
- The first node is considered odd, the second node even and so on ...

解题思路

【绘图大法】

官方解答的图很好理解，需要注意每次指针的移动。

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def oddEvenList(self, head: ListNode) -> ListNode:
        if not head:
            return head
        odd = head
        even = head.next
        odd_head = ListNode(0)
        odd_head.next = odd
        even_head = ListNode(0)
        even_head.next = even

        while odd.next and even.next:
            odd.next = even.next
            even.next = odd.next.next
            odd = odd.next
            even = even.next
        odd.next = even_head.next

        return odd_head.next
```

Day 17 —— Find All Anagrams in a String

查找回文字符串

问题描述

Given a string **s** and a **non-empty** string **p**, find all the start indices of **p**'s anagrams in **s**.

Strings consists of lowercase English letters only and the length of both strings **s** and **p** will not be larger than 20,100.

The order of output does not matter.

Example 1:

```
Input:
s: "cbaebabacd" p: "abc"

Output:
[0, 6]

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
```

Example 2:

```
Input:
s: "abab" p: "ab"

Output:
[0, 1, 2]

Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```

解题思路

【滑动窗口】

```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        res = []
        k = len(p)
        dic_p = Counter(p)
        dic_s = {}
        left, right = 0, 0

        while right < len(s):
            c = s[right]
            if c not in dic_p:
                dic_s.clear()
                left = right = right + 1
            else:
                dic_s[c] = dic_s.get(c, 0) + 1
                if right - left + 1 == k:
                    if dic_s == dic_p:
                        res.append(left)

                    dic_s[s[left]] -= 1
                    left += 1
                right += 1

        return res
```

Day 18 —— Permutation in String

查找回文字符串

问题描述

Given two strings **s1** and **s2**, write a function to return true if **s2** contains the permutation of **s1**. In other words, one of the first string's permutations is the **substring** of the second string.

Example 1:

```
Input: s1 = "ab" s2 = "eidbaooo"
Output: True
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

```
Input:s1= "ab" s2 = "eidboaoo"
Output: False
```

Note:

1. The input strings only contain lower case letters.
2. The length of both given strings is in range [1, 10,000].

解题思路

【滑动窗口*】*

维护窗口大小为len(s1)的两个指针i和j，在字符串s2上滑动，判断窗口内的hash表是否和s1的hash表一致。

```
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        import collections
        count_dict = collections.Counter(s1)
        m = len(s1)
        i = 0
        j = m - 1
        while j < len(s2):
            if collections.Counter(s2[i:j+1]) == count_dict:
                return True
            i += 1
            j += 1
        return False
```

Day 19 —— Online Stock Span

股票价格跨度

问题描述

Write a class `StockSpanner` which collects daily price quotes for some stock, and returns the *span* of that stock's price for the current day.

The span of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backwards) for which the price of the stock was less than or equal to today's price.

For example, if the price of a stock over the next 7 days were `[100, 80, 60, 70, 60, 75, 85]`, then the stock spans would be `[1, 1, 1, 2, 1, 4, 6]`.

Example 1:

```
Input: ["StockSpanner","next","next","next","next","next","next","next"], [[],[100],
[80],[60],[70],[60],[75],[85]]
Output: [null,1,1,1,2,1,4,6]
Explanation:
First, S = StockSpanner() is initialized.  Then:
S.next(100) is called and returns 1,
S.next(80) is called and returns 1,
S.next(60) is called and returns 1,
S.next(70) is called and returns 2,
S.next(60) is called and returns 1,
S.next(75) is called and returns 4,
S.next(85) is called and returns 6.

Note that (for example) S.next(75) returned 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's price.
```

Note:

1. Calls to `StockSpanner.next(int price)` will have `1 <= price <= 10^5`.
2. There will be at most `10000` calls to `StockSpanner.next` per test case.
3. There will be at most `150000` calls to `StockSpanner.next` across all test cases.
4. The total time limit for this problem has been reduced by 75% for C++, and 50% for all other languages.

解题思路

【单调栈】

求出小于或等于今天价格的最大连续日数实际上就等价于求出最近的一个大于今日价格的日子。

```python
class StockSpanner:
    def __init__(self):
        self.stock = []

    def next(self, price: int) -> int:
        weight = 1
        while self.stock and self.stock[-1][0] <= price:
            weight += self.stock.pop()[1]
        self.stock.append((price, weight))

        return weight


# Your StockSpanner object will be instantiated and called as such:
# obj = StockSpanner()
# param_1 = obj.next(price)
```

Day 20 —— Kth Smallest Element in a BST

二叉搜索树中的k小元素

问题描述

Given a binary search tree, write a function `kthSmallest` to find the **k**th smallest element in it.

Note: You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

Follow up: What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

解题思路

【二叉搜索树】**

首先要理解二叉搜索树（Binary Search Tree）。

- 特点：每个节点的值大于其任意左侧子节点的值，小于其任意右节点的值。
- 目的：为了提高查找的性能，其查找在平均和最坏的情况下都是logn级别，接近二分查找。

【怎么遍历树】**

基本的几种遍历方法要熟悉：

【迭代】

结合BST特点，其中序遍历的结果即为升序序列。其中，第 `k-1` 个元素就是第 `k` 小的元素。

```
# 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 kthSmallest(self, root, k):
        def inorder(root):
            return inorder(root.left) + [root.val] + inorder(root.right) if root else
[]

        return inorder(root)[k-1]
```

Day 21 —— Count Square Submatrices with All Ones

统计全为1的正方形子矩阵

问题描述

Given a `m * n` matrix of ones and zeros, return how many **square** submatrices have all ones.

Example 1:

```
Input: matrix =
[
  [0,1,1,1],
  [1,1,1,1],
  [0,1,1,1]
]
Output: 15
Explanation:
There are 10 squares of side 1.
There are 4 squares of side 2.
There is  1 square of side 3.
Total number of squares = 10 + 4 + 1 = 15.
```

Example 2:

```
Input: matrix =
[
  [1,0,1],
  [1,1,0],
  [1,1,0]
]
Output: 7
Explanation:
There are 6 squares of side 1.
There is 1 square of side 2.
Total number of squares = 6 + 1 = 7.
```

Constraints:

- `1 <= arr.length <= 300`
- `1 <= arr[0].length <= 300`
- `0 <= arr[i][j] <= 1`

解题思路

【动态规划】**

动态规划问题最重要是3步：

1. 明确创建怎样的dp数组
2. 初始化dp数组
3. 明确动态转移方程

```
class Solution:
    def countSquares(self, matrix: List[List[int]]) -> int:
        m, n = len(matrix), len(matrix[0])
        dp_table = [[0] * n for _ in range(m)]
        res = 0

        for i in range(m):
            for j in range(n):
```

```
                if i == 0 or j == 0:
                    dp_table[i][j] = matrix[i][j]
                elif matrix[i][j] == 0:
                    dp_table[i][j] = 0
                else:
                    dp_table[i][j] = min(dp_table[i-1][j], dp_table[i][j-1],
dp_table[i-1][j-1]) + 1

                res += dp_table[i][j]

        return res
```

Day 22 —— Sort Characters By Frequency

依照频率进行字符排序

问题描述

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

```
Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
```

Example 2:

```
Input:
"cccaaa"

Output:
"cccaaa"

Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.
Note that "cacaca" is incorrect, as the same characters must be together.
```

Example 3:

```
Input:
"Aabb"

Output:
"bbAa"

Explanation:
"bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

解题思路

【桶排序】**

桶排序基本原理可参考：

https://www.cnblogs.com/bqwzx/p/11029264.html

```
class Solution:
    def frequencySort(self, s: str) -> str:
        # 桶排序
        res = []
        dict_s = collections.defaultdict(int)
        bucket = [[] for _ in range(len(s) + 1)]
        for i in s:
            dict_s[i] += 1

        for i in dict_s:
            bucket[dict_s[i]].append(i * dict_s[i])
        # 桶排序中按从小到大存储 ， 若要按照从大到小输出需反向
        for i in bucket[::-1]:
            if i:
                res.extend(i)

        return "".join(res)
```

Day 23 —— Interval List Intersections

区间列表的交集

问题描述

Given two lists of **closed** intervals, each list of intervals is pairwise disjoint and in sorted order.

Return the intersection of these two interval lists.

(Formally, a closed interval `[a, b]` (with `a <= b`) denotes the set of real numbers `x` with `a <= x <= b`. The intersection of two closed intervals is a set of real numbers that is either empty, or can be represented as a closed interval. For example, the intersection of [1, 3] and [2, 4] is [2, 3].)

Example 1:

```
Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],[15,24],[25,26]]
Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
Reminder: The inputs and the desired output are lists of Interval objects, and not
arrays or lists.
```

Note:

1. `0 <= A.length < 1000`
2. `0 <= B.length < 1000`
3. `0 <= A[i].start, A[i].end, B[i].start, B[i].end < 10^9`

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

解题思路

【双指针】

由于给定的两个区间列表都是排好序的，我们只需要用两个指针i 和 j，分别考察A、B数组的每个子区间的左右界，求出交集区间。

然后比较A、B两个数组中谁含有最小的末端，拥有最小末端的指针+1即可。

```
class Solution:
    def intervalIntersection(self, A: List[List[int]], B: List[List[int]]) ->
List[List[int]]:
        res = []
        i, j = 0, 0

        while i < len(A) and j < len(B):
            lo = max(A[i][0],B[j][0])
            hi = min(A[i][1],B[j][1])
            if lo <= hi:
                res.append([lo,hi])

            if A[i][1] < B[j][1]:
                i += 1
            else:
                j += 1


        return res
```

Day 24 —— Construct Binary Search Tree from Preorder Traversal

利用前序遍历构建二叉搜索树

问题描述

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:

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

Note:

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

解题思路

【二叉搜索树构建】**

同样先回顾前/中/后序搜索

求解思路

- 如果数列为空，则得到最小的子树 None

- 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

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

class Solution:
    def bstFromPreorder(self, preorder: List[int]) -> TreeNode:
        if not preorder:
            return None

        root = TreeNode(preorder[0])
        preorder_left = []
        preorder_right = []
        for i in range(1, len(preorder)):
            if preorder[i] < root.val:
                preorder_left.append(preorder[i])
            else:
                preorder_right.append(preorder[i])
        root.left = self.bstFromPreorder(preorder_left)
        root.right = self.bstFromPreorder(preorder_right)

        return root
```

Day 25 —— Uncrossed Lines

不交叉的线

问题描述

We write the integers of `A` and `B` (in the order they are given) on two separate horizontal lines.

Now, we may draw *connecting lines*: a straight line connecting two numbers `A[i]` and `B[j]` such that:

- `A[i] == B[j]`;
- The line we draw does not intersect any other connecting (non-horizontal) line.

Note that a connecting lines cannot intersect even at the endpoints: each number can only belong to one connecting line.

Return the maximum number of connecting lines we can draw in this way.

Example 1:

```
Input: A = [1,4,2], B = [1,2,4]
Output: 2
Explanation: We can draw 2 uncrossed lines as in the diagram.
We cannot draw 3 uncrossed lines, because the line from A[1]=4 to B[2]=4 will
intersect the line from A[2]=2 to B[1]=2.
```

Example 2:

```
Input: A = [2,5,1,2,5], B = [10,5,2,1,5,2]
Output: 3
```

Example 3:

```
Input: A = [1,3,7,1,7,5], B = [1,9,2,5,1]
Output: 2
```

Note:

1. 1 <= A.length <= 500
2. 1 <= B.length <= 500
3. 1 <= A[i], B[i] <= 2000

解题思路

【动态规划】**

递推公式：

设 A[0] ~ A[x] 与 B[0] ~ B[y] 的最大连线数为 f(x, y)，那么对于任意位置的 f(i, j) 而言：

从后往前遍历，可以得到**递推公式**：

- 如果A[i] == B[j] -> dp_table[i][j] = dp_table[i+1][j+1] + 1
- 否则 -> dp_table[i][j] = max(dp_table[i+1][j], dp_table[i][j+1])

最后dp[0][0]就是最大的划线数量

```
class Solution:
    def maxUncrossedLines(self, A: List[int], B: List[int]) -> int:
        M, N = len(A), len(B)
        dp_table = [[0] * (N + 1) for _ in range(M + 1)]

        for i in range(M)[::-1]:
            for j in range(N)[::-1]:
                if A[i] == B[j]:
                    dp_table[i][j] = dp_table[i+1][j+1] + 1
                else:
                    dp_table[i][j] = max(dp_table[i+1][j], dp_table[i][j+1])

        return dp_table[0][0]
```

Day 26 —— Contiguous Array

连续数组搜索

问题描述

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

Example 1:

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

Example 2:

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

```
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.

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**[1,1,0,0,0,1,1,1]**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

```
class Solution:
    def findMaxLength(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        total_sum = 0
        # 建立以sum为key的字典
        index_map = dict()
        index_map[0] = -1
        res = 0
        for i, num in enumerate(nums):
            if num == 0:
                total_sum -= 1
            else:
                total_sum += 1
            if total_sum in index_map:
                res = max(res, i - index_map[total_sum])
            else:
                index_map[total_sum] = i
        return res
```

Day 27 —— Possible Bipartition

可能的二分

问题描述

Given a set of `N` people (numbered `1, 2, ..., N`), we would like to split everyone into two groups of **any** size.

Each person may dislike some other people, and they should not go into the same group.

Formally, if `dislikes[i] = [a, b]`, it means it is not allowed to put the people numbered `a` and `b` into the same group.

Return `true` if and only if it is possible to split everyone into two groups in this way.

Example 1:

```
Input: N = 4, dislikes = [[1,2],[1,3],[2,4]]
Output: true
Explanation: group1 [1,4], group2 [2,3]
```

Example 2:

```
Input: N = 3, dislikes = [[1,2],[1,3],[2,3]]
Output: false
```

Example 3:

```
Input: N = 5, dislikes = [[1,2],[2,3],[3,4],[4,5],[1,5]]
Output: false
```

Note:

1. `1 <= N <= 2000`
2. `0 <= dislikes.length <= 10000`
3. `1 <= dislikes[i][j] <= N`
4. `dislikes[i][0] < dislikes[i][1]`
5. There does not exist `i != j` for which `dislikes[i] == dislikes[j]`.

解题思路

【dfs+着色】

- 假设第一组中的人是红色，第二组中的人是蓝色。
- 如果第一个人是红色的，那么这个人不喜欢的人必须是蓝色的。然后，任何不喜欢蓝色人的人都是红色的，那么任何不喜欢红色人的人都是蓝色的，依此类推。
- 如果在任何时候存在冲突，那么这个任务是不可能的完成的，因为从第一步开始每一步都符合逻辑。如果没有冲突，那么着色是有效的，所以答案是 true。
- 如何做到这一点：将任一结点涂成红色，然后将它的所有邻居都涂成蓝色，然后将所有的邻居的邻居都涂成红色，以此类推。如果我们将一个红色结点涂成蓝色（或蓝色结点涂成红色），那么就会产生冲突。

```
class Solution(object):
    def possibleBipartition(self, N, dislikes):
        graph = collections.defaultdict(list)
        for u, v in dislikes:
            graph[u].append(v)
            graph[v].append(u)

        color = {}
        def dfs(node, c = 0):
            if node in color:
                return color[node] == c
```

```
            color[node] = c
            return all(dfs(nei, c ^ 1) for nei in graph[node])

        return all(dfs(node)
                   for node in range(1, N+1)
                   if node not in color)
```

Day 28—— Counting Bits

比特位计数

问题描述

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

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

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

解题思路

【dp】

- 编码会按照0，1，10，11，100，101，110，111进行
- 规律为，每次进位之后，后面的数即为之前的所有内容的一个排序
- 即最开始为1位编码，当进位为两位之后，首位固定为1，而剩余的一位则是又走了一遍一位时的排序
- 所以就可以用此来构建动态规划方式了

```
class Solution:
    def countBits(self, num: int) -> List[int]:
        dp=[0,1]
        if num==0:
            return [0]
        if num==1:
            return [0,1]
        g=2
        i=2
        q=0
        while i <num+1:
            if q==g:
                q=0
                g=len(dp)
            h=dp[i-g]+1
```

```
            q+=1
            dp.append(h)
            i+=1
        return dp
```

Day 29 —— Course Schedule

课程表

问题描述

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses-1`.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0, and to take course 0
you should
             also have finished course 1. So it is impossible.
```

Constraints:

- The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about how a graph is represented.
- You may assume that there are no duplicate edges in the input prerequisites.
- `1 <= numCourses <= 10^5`

解题思路

【拓扑排序】

核心思想就是找寻找图中是否有环。统计课程安排图中每个节点的入度，生成**入度表**：

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        from collections import defaultdict, deque
        graph = defaultdict(list)
        degree = [0] * numCourses
        # 建图
        for x, y in prerequisites:
            graph[y].append(x)
            degree[x] += 1
        queue = deque([i for i in range(numCourses) if degree[i] == 0])

        cnt = 0
```

```
        while queue:
            i = queue.pop()
            cnt += 1
            for j in graph[i]:
                degree[j] -= 1
                if degree[j] == 0:
                    queue.appendleft(j)
        return cnt == numCourses
```

Day 30 —— K Closest Points to Origin

最接近原点的K个点

问题描述

We have a list of `points` on the plane. Find the `K` closest points to the origin `(0, 0)`.

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

Example 1:

```
Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just
[[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)
```

Note:

1. `1 <= K <= points.length <= 10000`
2. `-10000 < points[i][0] < 10000`
3. `-10000 < points[i][1] < 10000`

解题思路

【sort排序】

```
class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:
        points.sort(key = lambda p: p[0] ** 2 + p[1] ** 2)
        return points[:K]
```

Day 31 —— Edit Distance

编辑距离

问题描述

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

解题思路

【动态规划】

- 边界条件：

  一个空串和一个非空串的编辑距离为 dp_table[i][0] = i 和 dp_table[0][j] = j，dp_table[i][0] 相当于对 word1 执行 i 次删除操作，dp_table[0][j] 相当于对 word1执行 j 次插入操作。

- 状态转移方程：

  若 A 和 B 的最后一个字母相同：

  dp_table[i][j]=1+min(dp_table[i][j−1],dp_table[i−1][j],dp_table[i−1][j−1]−1)

  若 A 和 B 的最后一个字母不同：

  dp_table[i][j]=1+min(dp_table[i][j−1],dp_table[i−1][j],dp_table[i−1][j−1])

```
class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        m = len(word1)
        n = len(word2)

        dp_table = [[0] * (n + 1) for _ in range(m + 1)]

        if m * n == 0:
            return m + n

        for i in range(m + 1):
            dp_table[i][0] = i
        for j in range(n + 1):
```

```
            dp_table[0][j] = j

        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if word1[i - 1] != word2[j - 1]:
                    dp_table[i][j] = min(dp_table[i-1][j], dp_table[i][j-1],
dp_table[i-1][j-1]) + 1
                else:
                    dp_table[i][j] = min(dp_table[i-1][j], dp_table[i][j-1],
dp_table[i-1][j-1] - 1) + 1

        return dp_table[-1][-1]
```

时间复杂度*O(mn)*

空间复杂度*O(mn)*