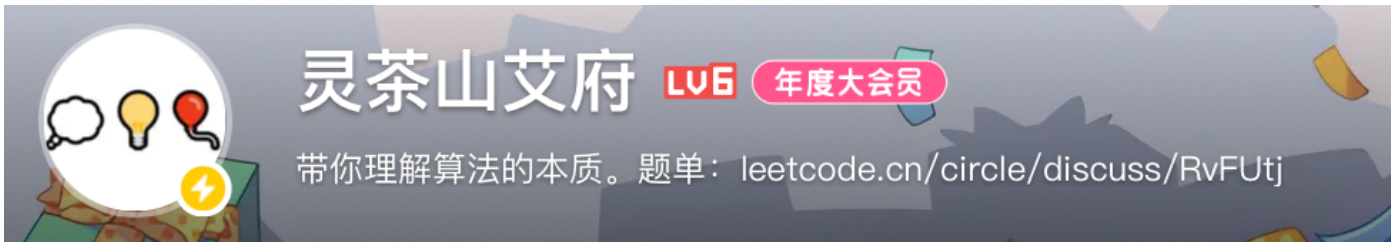
计算概论第一次学业辅导

一些题库和学习资源

http://cs101.openjudge.cn

https://leetcode.cn



基本语法与常用数据结构

基本语法

```
#四则运算：+、-、*、/、**、//
#基础的循环：if和while，range函数的使用
#常用的数据结构：字符串、列表、集合、字典、队列、堆
#常用的外置库：math/defaultdict/heapq等（后续会更细致的讲）
```

例1.(用列表记录状态)

http://cs101.openjudge.cn/practice/02808/

某校大门外长度为L的马路上有一排树，每两棵相邻的树之间的间隔都是1米。我们可以把马路看成一个数轴，马路的一端在数轴0的位置，另一端在L的位置；数轴上的每个整数点，即0，1，2，……，L，都种有一棵树。
马路上有一些区域要用来建地铁，这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标都是整数，区域之间可能有重合的部分。现在要把这些区域中的树（包括区域端点处的两棵树）移走。你的任务是计算将这些树都移走后，马路上还有多少棵树。

输入

输入的第一行有两个整数L（1 <= L <= 10000）和 M（1 <= M <= 100），L代表马路的长度，M代表区域的数目，L和M之间用一个空格隔开。接下来的M行每行包含两个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。

输出

输出包括一行，这一行只包含一个整数，表示马路上剩余的树的数目。

样例输入

```
500 3
150 300
100 200
470 471
```

样例输出

```
298
```

代码参考

```
L, M = map(int, input().split())
list=[1]*(L+1)
sum=0
for _ in range(M):
    start, end= map(int,input().split())
    for i in range (start,end+1):
        list[i]=0
for j in range(L+1):
    if list[j]==1:
        sum+=1
print(sum)
```

Q: 如何优化此代码?

（差分数组）

```
L, M = map(int, input().split())
diff = [0] * (L + 2)  # 差分数组

# 标记区间变化：O(M)
for _ in range(M):
    start, end = map(int, input().split())
    diff[start] += 1
    diff[end + 1] -= 1

# 计算前缀和：O(L)
current = 0
remaining_trees = 0
for i in range(L + 1):
    current += diff[i]
    if current == 0:  # 没有被任何区间覆盖
        remaining_trees += 1

print(remaining_trees)
```

例2.（质数筛法）

http://cs101.openjudge.cn/2024fallroutine/03143/

描述

验证“歌德巴赫猜想”，即：任意一个大于等于6的偶数均可表示成两个素数之和。

输入

输入只有一个正整数x。(x<=2000)

输出

如果x不是“**大于等于**6的偶数”，则输出一行：
Error!
否则输出这个数的**所有**分解形式，形式为：
x=y+z
其中x为待验证的数，y和z满足y+z=x，而且y<=z，y和z均是素数。
如果存在多组分解形式，则按照**y的升序**输出所有的分解，每行一个分解表达式。

注意输出**不要**有多余的空格。

样例输入

```
输入样例1：
7
输入样例2：
10
输入样例3：
100
```

样例输出

```
输出样例1：
Error!
输出样例2：
10=3+7
10=5+5
输出样例3：
100=3+97
100=11+89
100=17+83
100=29+71
100=41+59
100=47+53
```

思路：我们直接做一个质数表，然后用质数表中的数去筛看另一个分量是不是也在质数表中

质数的筛法：

1.朴素算法

```python
def is_prime_optimized(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n ** 0.5) + 1, 2):  # 只检查奇数
```

```
        if n % i == 0:
            return False
    return True

primes = [n for n in range(1, 1000) if is_prime_optimized(n)]
print(primes)
```

2.埃氏筛

```
s=[True for i in range(n)]
z=[]
for i in range(2,n):
  if s[i]:
    z.append(i)
    for j in range(i+i,n,i):
      s[j]=False
print(z)
```

3.欧式筛

```
#定义欧拉筛函数,判断n是否为质数
def euler_sieve(n):
    # 初始化质数列表(primes) prime:质数
    primes = []
    #初始化标记列表(is_prime)  假设所有的数都是质数
    is_prime = [True] * (n + 1)
    #因为质数是除了1和它本身外再不能被其他数整除的自然数. 0既不是质数也不是合数
    is_prime[0] = is_prime[1] = False
    #遍历(2, n+1) 包左不包右
    for i in range(2, n + 1):
        if is_prime[i]:            #如果i是质数
            primes.append(i)       #将i添加到质数列表
        #设置变量prime遍历primes中的每一个元素,用已经找到的质数去筛选掉i的倍数
        #核心细节: 用已经找到的质数去筛选掉i的倍数
        for prime in primes:
            #核心细节:如果i与当前质数的乘积大于视为越界,不做处理,结束本轮遍历
            if i * prime > n:
                break
            #并将其标注为非质数
            is_prime[i * prime] = False
            #核心细节:如果i是prime的倍数,说明它的最小质因子是prime, 即: 非质数会被最小因子筛去, 结束本轮遍历,防止重复标记非质数,提高了效率
            if i % prime == 0:
                break
    #返回质数列表
    return primes
```

例3.（字典与defaultdict）

http://cs101.openjudge.cn/20241107mockexam/M18182/

描述

Q神无聊的时候经常打怪兽。现在有一只怪兽血量是b，Q神在一些时刻可以选择一些技能打怪兽，每次释放技能都会让怪兽掉血。
现在给出一些技能ti,xi，代表这个技能可以在ti时刻使用，并且使得怪兽的血量下降xi。这个打怪兽游戏有个限制，每一时刻最多可以使用m个技能（一个技能只能用一次）。如果技能使用得当，那么怪兽会在哪一时刻死掉呢?

输入

第一行是数据组数nCases, nCases<=100
对于每组数据，第一行是三个整数n,m,b，n代表技能的个数，m代表每一时刻可以使用最多m个技能，b代表怪兽初始的血量。
1<=n<=1000，1<=m<=1000，1<=b<=10^9
接下来n行，每一行一个技能ti,xi，1<=ti<=10^9，1<=xi<=10^9

输出

对于每组数据，输出怪兽在哪一时刻死掉，血量小于等于0就算挂，如果不能杀死怪兽，输出alive

样例输入

```
2
1 1 10
1 5
2 2 10
1 5
1 5
```

样例输出

```
alive
1
```

参考代码

```
from collections import defaultdict
nCases=int(input())
for _ in range(nCases):
    n,m,b=map(int,input().split())
    skills=[tuple(map(int,input().split())) for _ in range(n)]
    skills.sort() #此处稍微讲一下各种sort类型的greedy
    skill_t=defaultdict(list)
    for i in skills:
        skill_t[i[0]].append(i[1])
    for i in skill_t:
        skill_t[i]=sorted(skill_t[i],reverse=True)
        b-=sum(skill_t[i][:m])
        if b<=0:
            print(i)
            break
```

```
    if b>0:
        print("alive")
```

常用数据结构

1. 队列

```
#queue：左边进右边出，先进先出（FIFO）
#deque：双端队列，此时没有要求只能哪边进哪边出

from queue import Queue
queue=Queue()
queue.put("A")
queue.put("B")
print(queue.get())
print(queue.get())

from collections import deque
queue=deque()
queue.append("A")
queue.appendleft("B")
queue.append("C")
queue.popleft()
queue.pop()
```

例1（约瑟夫问题）

问题描述

有 n 个人围坐成一个圆圈，从第 1 个人开始报数，数到第 k 个人时，该人出列。然后从下一个人重新开始报数，数到第 k 个人再出列。如此反复，直到所有人都出列为止。

要求编写程序，按出列顺序输出每个人的编号。

输入格式

- 第一行包含两个整数 n 和 k
- n 表示总人数（1 ≤ n ≤ 1000）
- k 表示报数的间隔（1 ≤ k ≤ 100）

输出格式

- 输出一行，包含 n 个整数，表示出列的顺序
- 数字之间用空格分隔

样例输入

```
7 3
```

样例输出

```
3 6 2 7 5 1 4
```

```
from collections import deque

def josephus(n, k):
    """
    约瑟夫环问题
    n: 总人数
    k: 报数到k的人出列
    """
    queue = deque(range(1, n+1))
    result = []

    while queue:
        # 前k-1个人从队头移到队尾
        for _ in range(k-1):
            queue.append(queue.popleft())
        # 第k个人出列
        result.append(queue.popleft())

    return result

# 示例：n=7, k=3
# 输出：[3, 6, 2, 7, 5, 1, 4]
```

2.堆

```
#今天先粗略讲一下堆，后面贪心&搜索会更细致地讲具体的用法
#我们暂时不用纠结堆的本质，只需要知道他是一个能够自己维护的最小的永远在堆顶的数据结构

import heapq

# 堆的基本操作演示
nums = [3, 1, 4, 1, 5, 9, 2, 6]

# 建堆（最小堆）
heapq.heapify(nums)
print(f"建堆后: {nums}")  # [1, 1, 2, 3, 5, 9, 4, 6]

# 弹出最小元素
min_val = heapq.heappop(nums)
print(f"弹出最小值: {min_val}, 剩余堆: {nums}")

# 插入新元素
heapq.heappush(nums, 0)
print(f"插入0后: {nums}")
```

双指针

双指针是一种使用**两个指针**（索引）在数据结构中协同工作的算法技巧。

为什么需要双指针

- 优化时间复杂度：将O(n²)优化为O(n)
- 简化复杂问题：将复杂问题分解为指针移动
- 空间效率高：通常只需要O(1)额外空间

双指针的三种类型

1. **相向指针**：从两端向中间移动
2. **同向指针**：从同一端同向移动（快慢指针主要应用地点还是链表）
3. **滑动窗口**：同向指针的特殊形式

例1.两数之和

https://leetcode.cn/problems/kLl5u1/description/

题目描述

给定一个已按**非递减顺序**排列的整数数组 `numbers`，从数组中找出两个数满足相加之和等于目标数 `target`。

假设每个输入只对应唯一的答案，而且不可以重复使用相同的元素。

```
输入：numbers = [2,7,11,15], target = 9
输出：[1,2]
解释：2 与 7 之和等于目标数 9。因此 index1 = 1, index2 = 2。
```

参考代码

```python
def twoSum(numbers, target):
    left, right = 0, len(numbers) - 1

    while left < right:
        current_sum = numbers[left] + numbers[right]

        if current_sum == target:
            return [left + 1, right + 1]  # 题目要求索引从1开始
        elif current_sum < target:
            left += 1  # 和太小，左指针右移
        else:
            right -= 1  # 和太大，右指针左移

    return []  # 无解
```

例2. 接雨水

https://leetcode.cn/problems/trapping-rain-water/description/

给定 `n` 个非负整数表示每个宽度为 `1` 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：


```
输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]
输出：6
解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。
```

示例 2：

```
输入：height = [4,2,0,3,2,5]
输出：9
```

参考代码（双指针解法）

```python
class Solution:
    def trap(self, height: List[int]) -> int:
        if not height:
            return 0

        left, right = 0, len(height) - 1
        left_max, right_max = height[left], height[right]
        water = 0

        while left < right:
            # 关键：总是处理较矮的那一边
            if left_max < right_max:
                left += 1
                # 更新左边最大值
                left_max = max(left_max, height[left])
                # 计算当前位置能接的雨水
                water += left_max - height[left]
            else:
```

```
                right -= 1
                # 更新右边最大值
                right_max = max(right_max, height[right])
                # 计算当前位置能接的雨水
                water += right_max - height[right]

        return water
```

补充一些

str

```
s.isdigit 判断是否都为数字
s.islower/isupper 大小写
s.startswith() s.endswith() 判断前后缀是否存在
s.find() s.index() 返回该字符第一个索引
s.count() 计数
```

后两个在list里也能用

list

```
del l[0] # 删掉指定位置元素
l.remove(1)    # 删掉第一个匹配值
print(l.pop())  # 删除并返回最后一个
```

dict

```
d['a'] = 1, d['a'] = {} # 增加键值对
del d['a'] # 删除键值对
'a' in d, 'a' in d.keys() # 查键
```

math

```
math.ceil(x)：返回大于或等于 x 的最小整数。
math.floor(x)：返回小于或等于 x 的最大整数。
math.trunc(x)：返回 x 的整数部分（截断小数部分）。
math.sqrt(x)：返回 x 的平方根。
math.pow(x, y)：返回 x 的 y 次幂。
math.exp(x)：返回 e 的 x 次幂。
math.log(x[, base])：返回 x 的自然对数（默认以 e 为底），或以指定 base 为底的对数。
math.log2(x)：返回 x 的以 2 为底的对数。
math.log10(x)：返回 x 的以 10 为底的对数。
math.fabs(x)：返回 x 的绝对值。
math.factorial(x)：返回 x 的阶乘（x 必须是非负整数）。
```

小数位数保留问题

```
number = 12345.6789
formatted_number = "Number: {:.2f}".format(number)
print(formatted_number)  # 输出: Number: 12345.68
```