The_KEY_to_Ultra_Tough_Problems

1.Tricky_Problems

T25353:排队

总时间限制: 60000ms　单个测试点时间限制: 3000ms　内存限制: 524288kB

描述

有 N 名同学从左到右排成一排，第 i 名同学的身高为 hi。现在张老师想改变排队的顺序，他能进行任意多次（包括0次）如下操作：

- 如果两名同学相邻，并且他们的身高之差不超过 D，那么老师就能交换他俩的顺序。

请你帮张老师算一算，通过以上操作，字典序最小的所有同学（从左到右）身高序列是什么？

输入

第一行包含两个正整数 N, D ($1<=N<=10^5$, $1<=D<=10^9$)。
接下去 N 行，每行一个正整数 h_i ($1<=h_i<=10^9$) 表示从左到右每名同学的身高。

输出

输出 N 行，第 i 行表示答案中第 i 名同学的身高。

```
# 25353: 排队
# 题意：每次只能交换相邻且身高差不超过 D 的人，求字典序最小的最终排列
# 思路：将允许交换关系看作有向无环图（DAG），每次取“入度为 0”的人中身高最小者输出。
# 实现：多轮扫描——每轮找出当前所有入度为 0 的节点集合 S，
#       对 S 内按身高升序输出并从序列中删除，重复直到序列为空。
# 注意：最坏情况下复杂度为 O(N^2)。

import sys
input = sys.stdin.readline

def solve():
    line = input().split()
    if not line:
        return
    n, D = map(int, line)
    arr = [int(input()) for _ in range(n)]

    result = []
    cur = arr[:]  # 当前剩余队列

    while cur:
        m = len(cur)
        S_idx = []  # 当前入度为 0 的位置下标
        left_min = None
        left_max = None

        for i in range(m):
            h = cur[i]
            if i == 0:
                # 第一个人必然没有左侧约束，入度为 0
                S_idx.append(i)
                left_min = h
                left_max = h
                continue

            # 判断是否满足：与左侧所有人身高差均 ≤ D
            # 即 h 必须位于 [left_max - D, left_min + D] 区间内
            if left_max - D <= h <= left_min + D:
                S_idx.append(i)

            # 更新左侧区间最小/最大值
            if h < left_min:
                left_min = h
            if h > left_max:
                left_max = h

        # 收集并按身高升序排序
        S = [cur[i] for i in S_idx]
        S.sort()

        # 输出这些人
        result.extend(S)

        # 删除已输出的元素（保持原相对顺序）
        to_remove = set(S_idx)
        cur = [cur[i] for i in range(m) if i not in to_remove]

    # 输出结果
    print('\n'.join(map(str, result)))


if __name__ == "__main__":
    solve()
```

T26971:分发糖果

总时间限制: 1000ms　内存限制: 65536kB

描述

n 个孩子站成一排。给你一个整数数组 ratings 表示每个孩子的评分。

你需要按照以下要求，给这些孩子分发糖果：

每个孩子至少分配到 1 个糖果。
相邻两个孩子评分更高的孩子会获得更多的糖果。

请你给每个孩子分发糖果，计算并返回需要准备的 **最少糖果数目** 。

输入

第一行包含一个整数n。1 <= n <= 2 * 10^4
第二行包含n个整数，相邻整数间以空格隔开。0 <= ratings[i] <= 2 * 10^4

输出

一个整数

```
def hillgravedown(listy):
    hillgravedown_n = len(listy)
    hilldown_left = [0] * hillgravedown_n
    for i in range(hillgravedown_n):
        if i > 0 and listy[i] > listy[i - 1]:
            hilldown_left[i] = hilldown_left[i-1] + 1
        else:
            hilldown_left[i] = 1

    hilldown_right = hillgravedown_dust = 0
    for i in range(hillgravedown_n - 1, -1, -1):
        if i < hillgravedown_n - 1 and listy[i] > listy[i + 1]:
            hilldown_right += 1
        else:
            hilldown_right = 1
        hillgravedown_dust += max(hilldown_left[i], hilldown_right)

    return hillgravedown_dust

n = int(input())
js = list(map(int, input().split()))
print(hillgravedown(js))
```

M04147:汉诺塔问题(Tower of Hanoi)

总时间限制: 1000ms　内存限制: 65535kB

描述

一、汉诺塔问题

有三根杆子A，B，C。A杆上有N个(N>1)穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至C杆： 每次只能移动一个圆盘； 大盘不能叠在小盘上面。 提示：可将圆盘临时置于B杆，也可将从A杆移出的圆盘重新移回A杆，但都必须遵循上述两条规则。

问：如何移？最少要移动多少次?

```
def mov_hanoi(n0, a0, b0):
    return f'{n0}:{a0}->{b0}'


def solv_hanoi(n0, a0, b0, c0):
    if n0 == 1:
        return [mov_hanoi(n0, a0, c0)]
    else:
        jit_sp_solv_hanoi = []
        for i in solv_hanoi(n0 - 1, a0, c0, b0):
            jit_sp_solv_hanoi.append(str(i))
        jit_sp_solv_hanoi.append(mov_hanoi(n0, a0, c0))
        for i in solv_hanoi(n0 - 1, b0, a0, c0):
            jit_sp_solv_hanoi.append(str(i) )
        return jit_sp_solv_hanoi


n, a, b, c = list(map(str,input().split()))
n_int = int(n)
js = solv_hanoi(n_int, a, b, c)
for j in js:
    print(j)
```

T28701:炸鸡排

总时间限制: 1000ms 内存限制: 65536kB

描述

小P买了n块鸡排，想将它们做成美味的炸鸡排，其中第i块鸡排需要t[i]秒炸熟。小P只有一个炸锅，炸锅内可以放置k(k≤n)块鸡排。小P是个完美主义者，他要求任意时刻炸锅内必须恰有k块鸡排。他可以在任意时刻改变锅内正在炸的鸡排，只需保证已经熟了的鸡排不能继续留在锅中。小P希望知道炸鸡排最多可以持续多少时间。

例如，小P的三块鸡排需要分别需要1,1,1的时间炸熟，炸锅内需要放置2块鸡排，那么他决定在第一个0.5秒炸1和2两块鸡排，第二个0.5秒炸2和3两块鸡排，第三个0.5秒炸1和3两块鸡排，共持续了1.5秒。

输入

第一行输入两个正整数n和k，k≤n
第二行输入n个正整数，代表n块鸡排分别需要炸熟的时间t[1],t[2]...t[n]
输入数据保证，n≤1000，0<t[i]≤1000000

输出

输出一个双精度浮点数，代表炸鸡排最多可以持续的时间，结果保留三位小数。

```
def zhajipai(js,k):
    js.sort(reverse=True)
    res = sum(js) / k
    if js[0] <= res:
        return res
    else:
        new_js = js[1:]
        nk = k-1
        return zhajipai(new_js,nk)

n,k0 = map(int,input().split())
js0 = list(map(int,input().split()))
print(f"{zhajipai(js0,k0):.3f}")
```

T04117:简单的整数划分问题

总时间限制: 100ms 内存限制: 65536kB

描述

将正整数n 表示成一系列正整数之和，n=n1+n2+…+nk, 其中n1>=n2>=…>=nk>=1 ， k>=1 。
正整数n 的这种表示称为正整数n 的划分。正整数n 的不同的划分个数称为正整数n 的划分数。

输入

标准的输入包含若干组测试数据。每组测试数据是一个整数N(0 < N <= 50)。

输出

对于每组测试数据，输出N的划分数。

样例输入

```
5
```

样例输出

```
7
```

提示

5, 4+1, 3+2, 3+1+1, 2+2+1, 2+1+1+1, 1+1+1+1+1

```
def div(n0, k0):
    if k0 == 1:
        return 1
    elif k0 == 2:
        return n0 // 2
    else:
        res = 0
        i = 0
        while n0 - i * k0 - 1 >= 2:
            res += div(n0 - i * k0 - 1, k0 - 1)
            i += 1
        return res


def solve():
    n = int(input())
    sgm = 0
    for i in range(1, n + 1):
        sgm += div(n, i)
    print(sgm)


while True:
    try:
        if __name__ == '__main__':
            solve()
    except:
        break
```

T29947:校门外的树又来了

总时间限制: 1000ms 内存限制: 65536kB

描述

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 <= 10^9）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
```

```
n,line = list(map(int, input().split()))
tree = n + 1
cut = []
for _ in range(line):
    a,b = list(map(int, input().split()))
    cut.append((a,b))
cut.sort(key=lambda x:x[0])

init = 0
kil = 0
while init <= line-1:
    at = cut[init][0]
    bt = cut[init][1]
    while init < line-1 and cut[init+1][0] <= bt:
        init += 1
        bt = max(cut[init][1], cut[init-1][1],bt)
    init += 1
    kil += bt - at + 1

print(tree - kil)
```

T02754:八皇后

总时间限制: 1000ms 内存限制: 65536kB

描述

会下国际象棋的人都很清楚：皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将8个皇后放在棋盘上（有8 * 8个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。
对于某个满足要求的8皇后的摆放方法，定义一个皇后串a与之对应，即$a=b_1b_2...b_8$，其中b_i为相应摆法中第i行皇后所处的列数。已经知道8皇后问题一共有92组解（即92个不同的皇后串）。
给出一个数b，要求输出第b个串。串的比较是这样的：皇后串x置于皇后串y之前，当且仅当将x视为整数时比y小。

输入

第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数b(1 <= b <= 92)

输出

输出有n行，每行输出对应一个输入。输出应是一个正整数，是对应于b的皇后串。

```
def kOj(listy):
    return ''.join(listy)


def _2sqr(x0, y0, x1, y1):
    if abs(x0 - x1) == abs(y0 - y1):
        return True
    else:
```

```
        return False


def permute0(listy):
    if len(listy) == 1:
        return [[listy[0]]]
    else:
        al_listy = []
        for j_in_listy in listy:
            n_listy = [j_in_listy]
            listy0 = listy[:]
            listy0.remove(j_in_listy)
            for k_in_listyf in permute0(listy0):
                for k_in_listy in k_in_listyf:
                    n_listy.append(k_in_listy)
                al_listy.append(n_listy)
                n_listy = [j_in_listy]
        return al_listy


al = permute0([1, 2, 3, 4, 5, 6, 7, 8])
bl = []
for a in al:
    blacklist = 0
    for i in range(0, 7):
        if blacklist == 0:
            for j in range(i + 1, 8):
                if _2sqr(i + 1, int(a[i]), j + 1, int(a[j])):
                    blacklist = 1
                    break

    if blacklist == 0:
        bl.append(a)


res = []
for a in bl:
    aa = []
    for b in a:
        aa.append(str(b))
    res.append(aa)
res.sort()

Lne = int(input())
for _ in range(Lne):
    n = int(input())
    print(kOj(res[n - 1]))
```

T02287:Tian Ji -- The Horse Racing

查看

总时间限制: 5000ms 内存限制: 65536kB

全局题号
添加于
提交次数
尝试人数
通过人数

你的提交

#	结果
5	Acce
4	Acce
3	Wron
2	Wron
1	Wron

描述

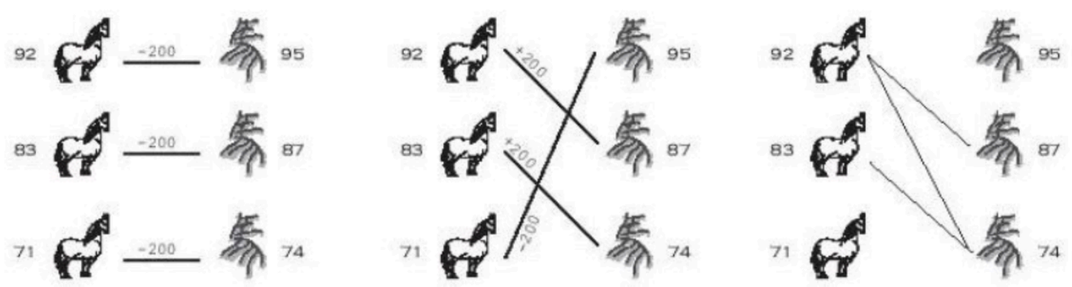
Here is a famous story in Chinese history.
That was about 2300 years ago. General Tian Ji was a high official in the country Qi. He likes to play horse racing with the king and others.

Both of Tian and the king have three horses in different classes, namely, regular, plus, and super. The rule is to have three rounds in a match; each of the horses must be used in one round. The winner of a single round takes two hundred silver dollars from the loser.

Being the most powerful man in the country, the king has so nice horses that in each class his horse is better than Tian's. As a result, each time the king takes six hundred silver dollars from Tian.

Tian Ji was not happy about that, until he met Sun Bin, one of the most famous generals in Chinese history. Using a little trick due to Sun, Tian Ji brought home two hundred silver dollars and such a grace in the next match.

It was a rather simple trick. Using his regular class horse race against the super class from the king, they will certainly lose that round. But then his plus beat the king's regular, and his super beat the king's plus. What a simple trick. And how do you think of Tian Ji, the high ranked official in China?



Were Tian Ji lives in nowadays, he will certainly laugh at himself. Even more, were he sitting in the ACM contest right now, he may discover that the horse racing problem can be simply viewed as finding the maximum matching in a bipartite graph. Draw Tian's horses on one side, and the king's horses on the other. Whenever one of Tian's horses can beat one from the king, we draw an edge between them, meaning we wish to establish this pair. Then, the problem of winning as many rounds as possible is just to find the maximum matching in this graph. If there are ties, the problem becomes more complicated, he needs to assign weights 0, 1, or -1 to all the possible edges, and find a maximum weighted perfect matching...

However, the horse racing problem is a very special case of bipartite matching. The graph is decided by the speed of the horses -- a vertex of higher speed always beat a vertex of lower speed. In this case, the weighted bipartite matching algorithm is a too advanced tool to deal with the problem.

In this problem, you are asked to write a program to solve this special case of matching problem.

输入

The input consists of up to 50 test cases. Each case starts with a positive integer n (n<=1000) on the first line, which is the number of horses on each side. The next n integers on the second line are the speeds of Tian's horses. Then the next n integers on the third line are the speeds of the king's horses. The input ends with a line that has a single `0' after the last test case.

输出

For each input case, output a line containing a single number, which is the maximum money Tian Ji will get, in silver dollars.

```
while True:
    n = int(input())
    if n == 0:
        break

    tj = list(map(int, input().split()))
    tj.sort()
    gw = list(map(int, input().split()))
    gw.sort()

    lt = 0
    rt = n - 1
    lg = 0
    rg = n - 1
    cnt = 0
    while lt <= rt:
        if tj[lt] > gw[lg]:
            cnt += 1
            lt += 1
            lg += 1
        elif tj[rt] > gw[rg]:
            cnt += 1
            rt -= 1
            rg -= 1
        else:
            if tj[lt] < gw[rg]:
                cnt -= 1

            lt += 1
            rg -= 1

    print(200 * cnt)
```

2.Special_Problems

E02689:大小写字母互换

总时间限制: 1000ms 内存限制: 65536kB

描述

把一个字符串中所有出现的大写字母都替换成小写字母，同时把小写字母替换成大写字母。

输入

输入一行：待互换的字符串。

输出

输出一行：完成互换的字符串（字符串长度小于80）。

样例输入

```
If so, you already have a Google Account. You can sign in on the right.
```

样例输出

```
iF SO, YOU ALREADY HAVE A gOOGLE aCCOUNT. yOU CAN SIGN IN ON THE RIGHT.
```

```
text=input()
def swap_case(s):
    return ''.join(
        c.lower() if c.isupper() else
        c.upper() if c.islower() else
        c
        for c in s
    )
swapped=swap_case(text)
print(swapped)
```

E02712:细菌繁殖

总时间限制: 1000ms　内存限制: 65536kB

描述

一种细菌的繁殖速度是每天成倍增长。例如：第一天有10个，第二天就变成20个，第三天变成40个，第四天变成80个，……。现在给出第一天的日期和细菌数目，要你写程序求出到某一天的时候，细菌的数目。

输入

第一行有一个整数n，表示测试数据的数目。其后n行每行有5个整数，整数之间用一个空格隔开。第一个数表示第一天的月份，第二个数表示第一天的日期，第三个数表示第一天细菌的数目，第四个数表示要求的那一天的月份，第五个数表示要求的那一天的日期。已知第一天和要求的一天在同一年并且该年不是闰年，要求的一天一定在第一天之后。数据保证要求的一天的细菌数目在长整数（long）范围内。

输出

对于每一组测试数据，输出一行，该行包含一个整数，为要求的一天的细菌数。

样例输入

```
2
1 1 1 1 2
2 28 10 3 2
```

样例输出

```
2
40
```

来源

2005~2006医学部计算概论期末考试

```
l = int(input())
months = [0,31,28,31,30,31,30,31,31,30,31,30,31]
for x in range(1, l + 1):
    i, j, k, m, n = list(map(int,input().split()))
    sm = 0
    for t in range(i, m):
        sm += months[t]
    sd = n - j
    r = sm + sd
    num = k * 2 ** r
    print(num)
```

E02724:生日相同

总时间限制: 1000ms　内存限制: 65536kB

描述

在一个有180人的大班级中，存在两个人生日相同的概率非常大，现给出每个学生的学号，出生月日。试找出所有生日相同的学生。

输入

第一行为整数n，表示有n个学生，n<100。
此后每行包含一个字符串和两个整数，分别表示学生的学号（字符串长度小于10）和出生月(1<=m<=12)日(1<=d<=31)。
学号、月、日之间用一个空格分隔。

输出

对每组生日相同的学生，输出一行，
其中前两个数字表示月和日，后面跟着所有在当天出生的学生的学号，数字、学号之间都用一个空格分隔。
对所有的输出，要求按日期从前到后的顺序输出。
对生日相同的学号，按输入的顺序输出。

样例输入

```
5
00508192 3 2
00508153 4 5
00508172 3 2
00508023 4 5
00509122 4 5
```

样例输出

```
3 2 00508192 00508172
4 5 00508153 00508023 00509122
```

```
n = int(input())
btc = {}
for _ in range(n):
    card,m,d = input().split()
    m = int(m)
    d = int(d)
    b = (m,d)
    if b not in btc:
        btc[b] = []
    btc[b].append(card)
results = []
for b in sorted(btc.keys()):
    ss = btc[b]
    if len(ss) >= 2:
        result = f"{b[0]} {b[1]} " + " ".join(ss)
        results.append(result)
for r in results:
    print(r)
```

E02734:十进制到八进制

总时间限制: 1000ms　内存限制: 65536kB

描述

把一个十进制正整数转化成八进制。

输入

一行，仅含一个十进制表示的整数a(0 < a < 65536)。

输出

一行，a的八进制表示。

样例输入

```
9
```

样例输出

```
11
```

```
n=int(input())
o=oct(n)
print(o[2:])
```

E02753:菲波那契数列

总时间限制: 1000ms　内存限制: 65536kB

描述

菲波那契数列是指这样的数列: 数列的第一个和第二个数都为1，接下来每个数都等于前面2个数之和。
给出一个正整数a，要求菲波那契数列中第a个数是多少。

输入

第1行是测试数据的组数n，后面跟着n行输入。每组测试数据占1行，包括一个正整数a(1 <= a <= 20)

输出

输出有n行，每行输出对应一个输入。输出应是一个正整数，为菲波那契数列中第a个数的大小

```
l=int(input())
for t in range(l):
    n=int(input())
    i=0
    j=1
    for t in range(n//2):
        i+=j
        j+=i
    if n%2==1:
        print(j)
    else:
        print(i)
```

E04067:回文数字（Palindrome Number）

总时间限制: 1000ms　内存限制: 65536kB

描述

给出一系列非负整数，判断是否是一个回文数。回文数指的是正着写和倒着写相等的数。

输入

若干行，每行是一个非负整数（不超过99999999）

输出

对每行输入，如果其是一个回文数，输出YES。否则输出NO。

样例输入

```
11
123
0
14277241
67945497
```

样例输出

```
YES
NO
YES
YES
NO
```

```
while True:
    try:
        n = input()
        if n == n[::-1]:
            print('YES')
        else:
            print('NO')
    except:
        break
```

M26976:摆动序列

总时间限制: 1000ms　内存限制: 65536kB

描述

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为 **摆动序列** 。第一个差（如果存在的话）可能是正数或负数。仅有一个元素或者含两个不等元素的序列也视作摆动序列。

例如，[1, 7, 4, 9, 2, 5] 是一个 **摆动序列** ，因为差值 (6, -3, 5, -7, 3) 是正负交替出现的。

相反，[1, 4, 7, 2, 5]，[1, 7, 4, 5, 5]，不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。

子序列 可以通过从原始序列中删除一些（也可以不删除）元素来获得，剩下的元素保持其原始顺序。

给你一个整数数组 nums ，返回 nums 中作为 **摆动序列** 的 **最长子序列的长度** 。

输入

第一行包含一个整数n。1 <= n <= 1000

第二行包含n个整数，相邻整数间以空格隔开。0 <= nums[i] <= 1000

输出

一个整数

```
n = int(input())
js = list(map(int, input().split()))

if n == 1 or max(js) - min(js) == 0:
    print(1)
elif n == 2 and js[0]!=js[1]:
    print(2)
else:
    k = 0
    stt = 1
    v = 0
    ks = [js[i] - js[i - 1] for i in range(1, n)]
    if 0 in ks:
        ks.remove(0)
    if ks[0] < 0:
        stt = -1

    for i in range(0, len(ks)):
        if ks[i] // stt > 0:
            stt *= (-1)
            k += 1
    print(k + 1)
```

M27217:有多少种合法的出栈顺序

总时间限制: 1000ms　内存限制: 131072kB

描述

栈是计算机中经典的数据结构，简单的说，栈就是限制在一端进行插入删除操作的线性表。

栈有两种最重要的操作，即 pop（从栈顶弹出一个元素）和 push（将一个元素进栈）。

```
n = int(input())
if n == 0:
    print(1)
else:
    c = 1
    for i in range (n):
        c = c * (4 * i + 2) // (i + 2)
print(c)
```

02:输出绝对值

总时间限制: 1000ms　内存限制: 65536kB

描述

输入一个浮点数，输出这个浮点数的绝对值。

输入

输入一个浮点数，其绝对值不超过10000。

输出

输出这个浮点数的绝对值，保留到小数点后两位。

```
print("%.2f" % abs(float(input())))
```

M19944:这一天星期几

总时间限制: 1000ms　内存限制: 65536kB

描述

在日常生活中，计算某一个具体的日期是星期几，往往需要去翻阅日历。请你帮助更快地计算出每个日期在一星期是第几天。

参考:

蔡勒公式（Zeller's congruence），是一种计算任何一日属一星期中哪一日的算法，由德国数学家克里斯提安·蔡勒推算出来，可以计算1582年10月15日之后的情况。

$$w=\left(y+\left[\frac{y}{4}\right]+\left[\frac{c}{4}\right]-2c+\left[\frac{26(m+1)}{10}\right]+d-1\right) \bmod 7$$

```
week = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
line = int(input())
for i in range(line):
    dt = input()
    c = int(dt[0:2])
    y = int(dt[2:4])
    m = int(dt[4:6])
    d = int(dt[6:8])
    if m <= 2:
        m += 12
        y -= 1
    if y == -1:
        y = 99
        c -= 1
    tw = y + y//4 + c//4 - 2*c + 26*(m+1)//10 + d - 1
    w = tw % 7
    print(week[w])
```