# Assignment #8: 田忌赛马来了

Updated 1021 GMT+8 Nov 12, 2024

2024 fall, Complied by <mark>同学的姓名、院系</mark>

**姓名：李彦臻**
**学号：2300010821**
**学院：数学科学学院**

**说明：**

1）请把每个题目解题思路（可选），源码 Python, 或者 C++（已经在 Codeforces/Openjudge 上 AC），截图（包含 Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用 word）。AC 或者没有 AC，都请标上每个题目大致花费时间。

2）提交时候先提交 pdf 文件，再把 md 或者 doc 文件上传到右侧“作业评论”。Canvas 需要有同学清晰头像、提交文件有 pdf、"作业评论"区有上传的 md 或者 doc 附件。

3）如果不能在截止前提交作业，请写明原因。

## 1. 题目

### 12558: 岛屿周长

matices, http://cs101.openjudge.cn/practice/12558/

思路：先建立一个 nxm 的矩阵，用于记录哪里是陆地哪里是水；然后暴力遍历（n 乘 m 复杂度）得出陆地的面积，乘以四得到初步周长；
接着再遍历每一行，记录横向的边的重叠数；再遍历每一列，记录纵向的边的重叠数；
最后再将将初步周长减去两倍的重叠边的个数即可~

代码：

```python
```

```
data=list(map(int,input().split()))
n,m=data[0],data[1]
area=[]#一个 nxm 的表格，用于记录哪里是陆地哪里是水
```

```
for i in range(n):
    area.append(list(map(int,input().split())))

x=0#记录陆地的面积，乘以四得到初步周长
y=0#记录横向的陆地重叠数
z=0#记录纵向的陆地重叠数
for i in range(n):
    x += area[i].count(1)
    for j in range(m-1):
        if area[i][j] + area[i][j+1] == 2:
            y += 1

for i in range(n-1):
    for j in range(m):
        if area[i][j] + area[i+1][j] == 2:
            z += 1

print(4*x - 2*y - 2*z)#将初步周长减去两倍的重叠边的个数即可
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>


### LeetCode54.螺旋矩阵

matrice, https://leetcode.cn/problems/spiral-matrix/

与OJ这个题目一样的 18106: 螺旋矩阵，http://cs101.openjudge.cn/practice/18106

思路：可以把这个螺旋的过程视为若干个周期：每循环一圈视为一个周期，进入新的一圈视为一个新的周期的开始；对于第 i 行 j 列的数，先找出其在第几个周期；
如果其在第 t 个周期，则说明此前已经排序过 n^2-(n-2t+2)^2 个数了！
接着，把外圈全部挖掉，剩下一个 n-2t+2 乘 n-2t+2 的矩阵；看他此时在第几行第几列，从而计算他是这个周期里第几个被排到的；最后再把这个周期之前就已经排序了的数的个数加上，就能得到它是第几个数了！

代码：

```python

```
```
n=int(input())
matrix=[[0]*n for _ in range(n)]

for i in range(n):
    for j in range(n):
        t=min(i+1,n-i,j+1,n-j)#找出其在第几个周期
        #如果在第 t 个周期，则说明此前已经排序过 n^2-(n-2t+2)^2 个数了！
        #把外圈全部挖掉，剩下一个 n-2t+2 乘 n-2t+2 的矩阵；
        #此时，不难算得他在第 i-t+1 行、第 j-t+1 列（从 0 开始），
        #从而就可以计算他是这个周期里第几个被排到的了~
        if i-t+1 == 0:#先验证是否在第一行！
            matrix[i][j]=(n**2-(n-2*t+2)**2) + (j-t+2)
        elif j-t+1 == n-2*t+1:#再验证是否在最后一列
            matrix[i][j]=(n**2-(n-2*t+2)**2) + (n-2*t+1) + (i-t+2)
        elif i-t+1 == n-2*t+1:#再验证是否在最后一行
            matrix[i][j]=(n**2-(n-2*t+2)**2) + 2*(n-2*t+1) + ((n-2*t+2) - (j-t+1))
        else:#接下来就只有可能在第一列了~
            matrix[i][j]=(n**2-(n-2*t+2)**2) + 3*(n-2*t+1) + ((n-2*t+2) - (i-t+1))

for i in range(n):
    print(" ".join(map(str,matrix[i])))
```

代码运行截图 ==（至少包含有"Accepted"）==


### 04133:垃圾炸弹

matrices, http://cs101.openjudge.cn/practice/04133/

思路：这道题我的方法比较暴力：对每个路口放置的炸弹，直接考察这个炸弹能炸到哪些垃圾，并把垃圾数量全部加起来放进一个 list 中，最后直接找这个 list 的最大值即可

代码：

```python

```
import array#不改变空间复杂度，但运行效率更高

d=int(input())
n=int(input())
data=[]#记录每个垃圾的信息即可
for i in range(n):
    data.append(list(map(int,input().split())))

bomb=array.array("i",[0]*(1025**2))#"i"表示 array 内全是整数
for i in range(1025):
    for j in range(1025):
        for trash in data:

```
                if abs(trash[0]-i) <= d and abs(trash[1]-j) <= d:
                    bomb[i*1025 + j] += trash[2]
                    #记录将炸弹放在第 j 列时，第 i 行能清理的垃圾数量

k=max(bomb)
i=bomb.count(k)
print(f"{i} {k}")
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>


---

### LeetCode376.摆动序列

greedy, dp, https://leetcode.cn/problems/wiggle-subsequence/

与 OJ 这个题目一样的，26976:摆动序列, http://cs101.openjudge.cn/routine/26976/

思路：这道题暂时没有想到很好的方法（o(n)），只想到了 o(n^2)的常规 dp 方法
大致思路就是，因为每时每刻（考察第 i 个数的时候）长度为 j 的序列有两种，一种是最后一个差为正，另一种是最后一个差为负，因此要建立两个 dp 表格：
dp_plus 里的第 i 个数记录每个时刻的“最后一个差为正且长度为 i 的序列”的最大尾数
dp_minus 里的第 i 个数记录每个时刻的“最后一个差为负且长度为 i 的序列”的最小尾数
然后接下来每一步都按部就班的分类讨论、比较大小即可~

代码：

```python

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

dp_plus=[[] for _ in range(n)]#第 i 个列表的数记录每个时刻的“最后一个差为正且长度为 i 的序列”的最大尾数
dp_minus=[[] for _ in range(n)]#第 i 个列表的数记录每个时刻的“最后一个差为负且长度为 i 的序列”的最小尾数
dp_plus[0].append(num[0])
dp_minus[0].append(num[0])

for i in range(1,n):
    for j in range(n):
        if len(dp_plus[j]) > 0 and num[i] < dp_plus[j][0]:#如果存在长度为 j 的 plus 序列且可以往里面加数
            if len(dp_minus[j+1]) == 0:
                dp_minus[j+1]=[num[i]]
            elif dp_minus[j+1][0] > num[i]:#则只需验证 num[i]是否比“原来的长度为 i+1 的 minus 序列的最小尾数”小即可
                dp_minus[j+1]=[num[i]]
    for j in range(n):
       if len(dp_minus[j]) > 0 and num[i] > dp_minus[j][0]:#如果存在长度为 j 的 minus 序列且可以往里面加数
            if len(dp_plus[j+1]) == 0:
                dp_plus[j+1]=[num[i]]
           elif dp_plus[j+1][0] < num[i]:#则只需验证 num[i]是否比“原来的长度为 i+1 的 plus 序列的最大尾数”大即可
                dp_plus[j+1]=[num[i]]

for i in range(n):
    if len(dp_plus[i]) > 0:
        x=i+1
    if len(dp_minus[i]) > 0:
        y=i+1
print(max(x,y))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>


### CF455A: Boredom

dp, 1500, https://codeforces.com/contest/455/problem/A

思路：注意到所有 i 和所有 i+1 能、且仅能选一个，因此可以把所有 i 之和记为 ai，并存入列表的第 i 个数，最后，再用 dp 方法，找出最大可能的分数！
具体而言，我们可以建立一个 dp 表格：表格的第 i 个数表示前 i 个数中选了第 i 个数的最大总分数（ai）；则基于一个事实（至多连续不选两个数）可推得：ai=max(ai-2+xi,ai-3+xi)
到了最后，最大值就是 a100000 和 a99999 中的较大者~

代码：

```python
```

```
n=int(input())
num=list(map(int,input().split()))
freq=[0]*(10**5 + 1)#记录每一个数在 num 里出现的次数
for i in range(n):
    freq[num[i]] += 1
total=[0]*(10**5 + 1)
for i in range(10**5 + 1):
    total[i]=i*freq[i]#不要直接 total[i]=i*(num.count(i))，这样复杂度就是 10**5 乘 n!
```

```
dp_one=[0]*(10**5 + 1)
dp_one[1]=total[1]
for i in range(2,10**5 + 1):
    dp_one[i]=max(dp_one[i-2] + total[i],dp_one[i-3] + total[i])
print(max(dp_one[10**5],dp_one[10**5 - 1]))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>


### 02287: Tian Ji -- The Horse Racing

greedy, dfs http://cs101.openjudge.cn/practice/02287

思路：在两边没有相同的数的时候，直接让田忌的每个马赢得尽量少即可；
而如果两边有相同的马，有两种可能的决策：用这个马制造平局，或者用这个马来赢一局；这里，在具体判断每个马应该使用哪种情况的时候，我用的是计算机方法，即通过对平局的所有可能情况进行枚举，一一算出他们此时的最大值，最后再取最大值的最大值即可。
（这里枚举的前提是，注意到，可以不妨假设只有一个值存在平局！（原理见后面的总结）这样就可以把需要枚举的情况从阶乘量级直接降到一次方量级！）

代码：

```python
```

```
def no_tie(one,two,result):#建立一个函数，专门计算不刻意安排平局情况的最大值
    #p.s.不刻意安排平局指的是，只有在田忌当下的最大的一匹马恰好等于国王的马的最小值时，才不得不安排平局
    m=len(one)
    x=0#能赢的局数
    y=0#打平的局数
    i,j=0,m-1#田忌和国王的指针
    while True:
        if i >= m or j < 0:
            break
        elif one[i] < two[0]:#如果他当下最大的马都无法击败国王最小的马，则 break
            break
        elif one[i] < two[j]:#如果小于，则把国王的指针往下移
            j -= 1
        elif one[i] == two[j]:#如果恰好相等，则除非这已经是国王最小的马了（就不得不平局），否则跳过找更小的
            if two[j] != two[0]:
                j -= 1
            else:#否则，如果已经是最小的了，则找到最多能同归于尽多少匹马（制造尽量多的平局）并 break
                a=one[i:].count(one[i])#田忌剩下的马中等于当下这匹马的马的数目
                y=min(a,j+1)
                break
        else:#如果大于，则说明找到了田忌当下的这匹马能赢的最大的国王的马
            i += 1
            j -= 1
            x += 1
    result.append(200*(x - (m-x-y)))

def race(n):
    tian=list(map(int,input().split()))
    king=list(map(int,input().split()))
    tian.sort(reverse=True)#把田忌的马从大到小排序
    king.sort()
    result=[]

    #先对原班人马计算出不刻意安排平局情况的最大值
    no_tie(tian[:],king[:],result)

    #再挨个枚举刻意安排安排平局的情况（可以不妨假设只有一个值存在平局）
    for i in range(n):
        if i == 0 or tian[i] != tian[i-1]:#前一个数与他不同，则新开一个副本
            tian_dup=tian[:]
            king_dup=king[:]
            if tian[i] in king_dup:
                tian_dup.remove(tian[i])
```

```
                king_dup.remove(tian[i])
                no_tie(tian_dup,king_dup,result)
        else:#前一个数与他相同，则接着之前的列表继续
            if tian[i] in king_dup:
                tian_dup.remove(tian[i])
                king_dup.remove(tian[i])
                no_tie(tian_dup,king_dup,result)

    print(max(result))

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

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

## 2. 学习总结和收获

<mark>如果作业题目简单，有否额外练习题目，比如：OJ“计概 2024fall 每日选做”、CF、LeetCode、洛谷等网站题目。</mark>

做第三题垃圾炸弹的时候，我一开始的思路比较简单粗暴：先把每个路口的垃圾数量用一个大表格记录下来，然后再计算出在每个路口放炸弹时能炸掉的垃圾数量；不过如果直接计算每个路口附近的（2d+1）^2 个路口的垃圾数量之和的话，会导致时间复杂度最高能够达到 1025*1025*100*100，这样显然会 TLE，因此我稍微改进了一下：

```
d=int(input())
n=int(input())
trash=[[0]*1025 for _ in range(1025)]
for i in range(n):
    data=list(map(int,input().split()))
    i,j,k=data[0],data[1],data[2]
    trash[i][j]=k

line=[[0]*1025 for _ in range(1025)]
for i in range(1025):
    for j in range(1025):
        line[i][j]=sum(trash[i][k] for k in range(1025) if abs(k-j) <= d)
        #记录将炸弹放在第j列时，第i行能清理的垃圾数量

bomb=array.array("i",[0]*(1025**2))#"i"表示array内全是整数
for i in range(1025):
    for j in range(1025):
        bomb[i*1025 + j]=sum(line[k][j] for k in range(1025) if abs(k-i) <= d)
        #记录将炸弹放在每一个路口能清理的垃圾数量

k=max(bomb)
i=bomb.count(k)
print(f"{i} {k}")
```

第一步先用 line 表格记录将炸弹放在第 j 列时，第 i 行能清理的垃圾数量（复杂度为 100），第二步再把 line[i-d][j]一直到 line[i+d][j]的数加起来（复杂度为 100），这样就能得到把炸弹放在第 i 行第 j 列能够找到的垃圾数量了；这样改进的话就相当于把一个 100*100 的步骤拆为了两个 100 的步骤，而 1024*1024*100 的复杂度勉强可以接受，于是我试着提交，但无论怎么改进，oj 一直在报 MLE；后来我没辙了，就咨询了一下 gpt，学会了 array 方法（之前一直有所耳闻，但没学过），使用了上面图片中的方法，终于不 MLE 了（虽然变成 TLE 了，但也算是有所进步了~）

后来，我想了半天自己实在是没想出什么好方法，于是看了看群里同学的交流，发现有一个同学的方法十分简单粗暴（代码如下图）：

#47185003提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
d=int(input())
n=int(input())
rubbishs=[]
for j in range(0,n):
    x,y,i=map(int,input().split())
    rubbishs.append((x,y,i))
points=[]
for ax in range(0,1025):
    for ay in range(0,1025):
        uuu=0
        for (x,y,i) in rubbishs:
            if abs(x-ax)<=d and abs(y-ay)<=d:
                uuu+=i
        points.append(uuu)
boom=max(points)
print(points.count(boom),boom)
```

基本信息

#: 47185003
题目: 04133
提交人: 24n2400010975贾镕旭
内存: 11964kB
时间: 1211ms
语言: Python3
提交时间: 2024-11-15 19:04:49



English 帮助 关于

他是直接对每个路口计算附近的所有垃圾之和；我一开始不太理解为什么他这样能 ac，因为我以为这个思路和我第一个思路一样，复杂度有 1024*1024*100*100；后来我才突然意识到，我是把周围每个没有垃圾的路口的垃圾量假设为 0 了，因此这样确实需要对每个路口求 100*100 个数的和；但是根本没必要这样，其实只需要像这个同学一样，对附近的每个垃圾求和就好了！而垃圾数量是很有限的（至多 20 个），所以无论如何，每个路口都最多只用对 20 个数求和！意识到这一点之后我很快就 ac 了~

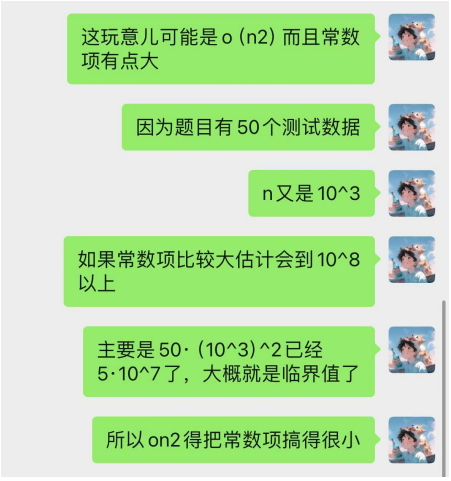
通过这道题，我不仅学会了 array 方法，也得出了一个教训：做题的时候要尽可能忽略次要因素，只考虑我们要考虑的东西就好了；这样复杂度会低不少，而且也更接近问题本质！总之，做这道题让我蛮有收获的~

此外，第六题田忌赛马这道题我也思考了非常久，最终非常有收获~我一个初步的想法是让每个田忌的马都赢的尽量的少，这样可以物有所值。这个思路在两边没有相同的数的时候是很方便的，但是在有相同的数的时候，情况会较复杂：究竟是让田忌的那个的马与国王的那个与他相同的马同归于尽（制造平局），还是用它来赢下一局呢？在考察完 123 和 123 的田忌赛马的最基础的情况之后，我一开始的想法是，对于一个马，如果比他弱的所有的国王的马都可以被比他弱的队友一一吃掉，则他就牺牲一下自己，拿来与他相同的马同归于尽（制造平局），除此以外的相同的马都拿来制造胜局。但是，后来我看了老师在群里的那个数据：
4 15 15 20 20
13 17 20 20 21
之后，我意识到，可能判断每个马究竟属于哪种情况并没有那么简单，因为这里的两个与对面相同的 20，一个是拿来制造平局了，另一个是拿来制造胜局了，可能并没有很简单的判断方法。

后来，我又花了很长时间试图寻找一个可以说的清楚的最优策略，但都以失败告终。于是，我开始改变策略，开始思考能不能使用计算机方法对平局情况进行枚举，来用计算机的运算找到最优策略；但是因为两边可能有非常多相同的数，而每一个相同的数都有可能拿来赢或者拿来平这两种情况，因此要枚举的情况可能有幂次甚至阶乘量级，这显然是不能接受的。于是我又花了比较长的时间跟另外一个同样是数院的同学讨论，我们俩最终意识到，完全可以假设两边的马只在同一个值（不妨设为 a）上相同！

这是因为，如果两边的马在两个不同的值上相同，比如两边在 a 上平局一次，在 b 上平局了一次（田忌的 a 和齐王的 a 平局了，田忌的 b 和齐王的 b 上平局了），那么可以对他们俩进行交换（即 a 对 b，b 对 a），这样就变成了赢一局输一局，拿到的钱是一样的，但是减少了平局的数量！因此，经过有限次这样的操作之后，出现平局的大小必然只会有一个值 a。换句话说，不可能同时存在两个值 a 和 b，使得齐王和田忌的马在 a 和 b 上都平了局（不然可以再次交换来减少平局）！这样一来，既然所有平局的马都是同一个大小，那么枚举的次数显然就是 o（n）量级了，接着按部就班的操作就可以了～

到这里，我以为一切都结束了，毕竟这个算法我估算了一下，是一个 o（n^2）量级，应该没问题，但是后来我没想到，写完了一个代码之后我在本地测试的数据都没问题，但是居然 TLE 了。于是，我又和那个同学交流了一下，意识到，即使是 on2 的算法也得尽量减少常数项，还得优化代码的细节~



（我和那个同学的交流记录）

于是，我又多次优化了代码，在优化的过程中，我学习了 binary search，学习了 deque，还回忆了一下双指针，学到了很多、也回忆了很多之前学过的但是没怎么用过的方法，很有收获！最终，在多个方法的不断优化探索之后，我终于把这个代码给 AC 了，真是不容易啊哈哈哈~

后来，我看了看标准答案，也很有收获；不过，虽然我的方法远不如它好，但是我沿着自己的思路一直坚持了下去，并在一次次优化中从 WA 逐渐变成了 TLE 并最终变成了 AC，还是蛮有成就感的~感觉在做这次作业的过程中，我也意识到了自己的很多薄弱点和没怎么练过的算法，看来还得针对性的练习一下，希望下次作业的时候能更熟练的运用更多算法~