Assignment #8: 田忌赛马来了

Updated 1021 GMT+8 Nov 12, 2024

2024 fall, Complied by 胡杨 元培学院

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：奇妙的想法之岛屿描边法（？）先给矩阵加一个保护圈，然后遍历矩阵，如果点m[i][j]是1，则看它上下左右四周有几个0，有几个0周长就增加几

大概耗时：15min

启示

一行代码加保护圈

```
l = [[0]*(m+2)] +[[0] +list(map(int,input().split()))+[0] for _ in range(n)]+
[[0]*(m+2)]
```

代码：

```
n,m=map(int,input().split()) #读取行数，列数
a=[[] for _ in range(n+2)] #搞n+2个空列表是为了方便加一圈保护圈

a[0].extend([0]*(m+2)) #加第一行的保护圈
a[-1].extend([0]*(m+2)) #加最后一行的保护圈
for i in range(1,n+1):
    a[i].extend([0]+list(map(int,input().split()))+[0]) #读取输入，顺便加上保护圈

l=0 #初始化周长
for i in range(1,n+1):
    for j in range(1,m+1):
        if a[i][j]==1:
```

```
                l+=4-(a[i][j-1]+a[i][j+1]+a[i-1][j]+a[i+1][j]) #如果遇到岛屿，计算它周围有几个1，4减去其就是周围海水的数量，即周围的边长

print(l) #输出周长
```

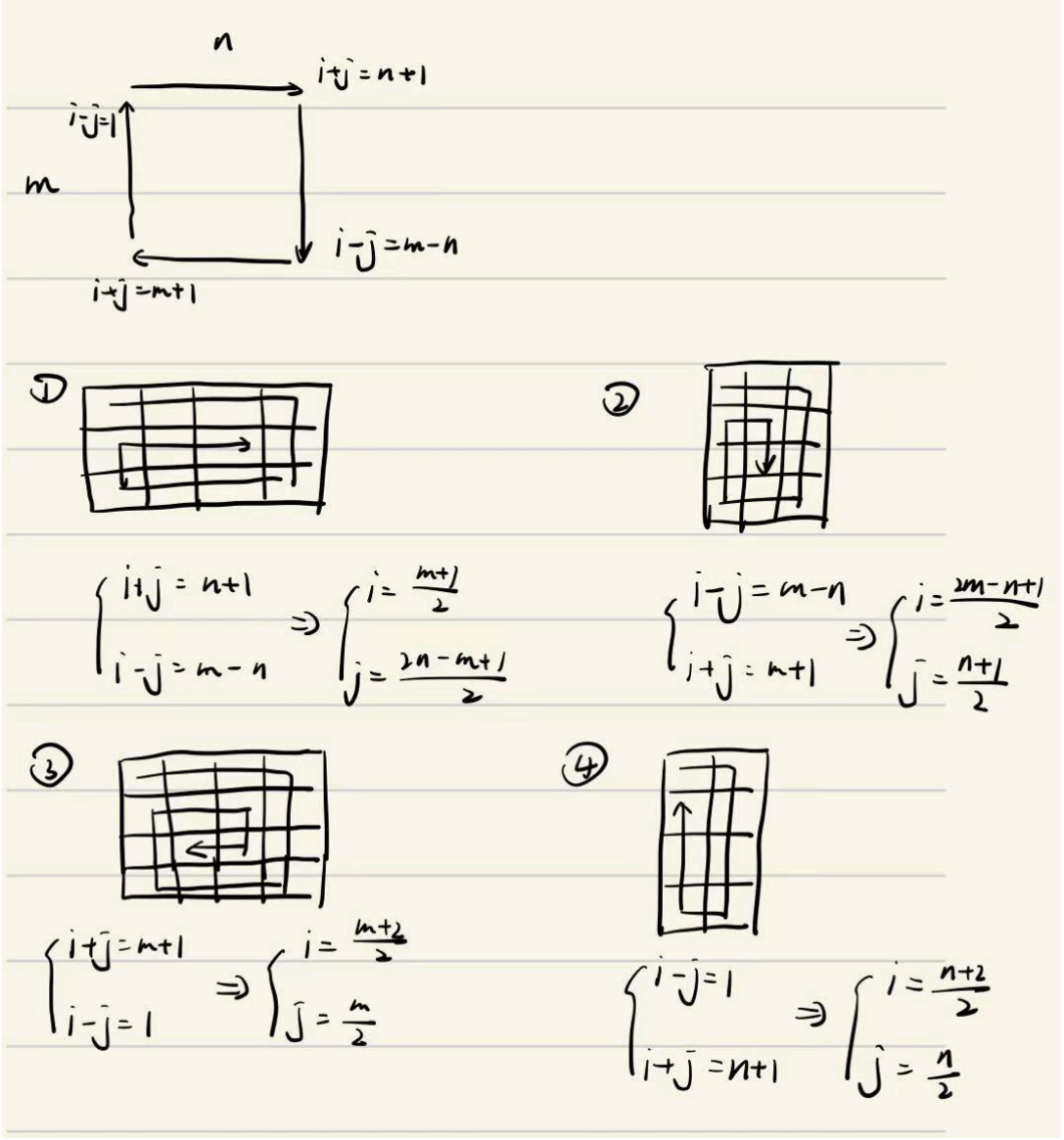
代码运行截图 （至少包含有"Accepted"）

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：一共m行n列（统一从1开始计，为了方便加一圈保护圈），主要的任务是模拟题干的描述，先读第一行，然后最后一列，然后最后一行，然后第一列。注意到从左往右读行的时候，如果i+j==n+1（i行j列，从1开始），需要转弯；从上往下时，i-j==m-n时转弯；从右往左时，i+j==m+1转弯；从下往上时，i-j==1时转弯。停止时间：如果连续两个转弯条件重合时停止（相当于要调头了），如图所示



耗时：50min（写代码不熟练写了好久）

改leetcode的格式又改了一个多小时X(，leetcode我恨你:(

启示

1.ast.literal_eval

leetcode上的输入令人膈应，使用`ast.literal_eval`将输入的字符串转化成理想的形式（如果要在本地跑的话）

`ast.literal_eval` 是 Python 标准库 `ast` 模块中的一个函数，用于安全地评估一个字符串形式的 Python 字面量结构，比如**列表、字典、元组、数字、字符串、布尔值、** `None` 等。与 `eval` 不同的是，`ast.literal_eval` 只能解析上述提到的安全的数据类型，不能执行任意的表达式或代码，因此更加安全。

这里是 `ast.literal_eval` 的基本用法示例：

导入模块

首先，你需要从 `ast` 模块中导入 `literal_eval` 函数。

```
from ast import literal_eval
```

使用 `literal_eval` 解析字符串

然后，你可以使用 `literal_eval` 来解析字符串形式的 Python 字面量。

示例 1: 解析列表

```
# 字符串形式的列表
list_str = '["apple", "banana", "cherry"]'

# 使用 literal_eval 将字符串转换为列表
list_obj = literal_eval(list_str)

print(list_obj)  # 输出: ['apple', 'banana', 'cherry']
print(type(list_obj))  # 输出: <class 'list'>
```

示例 2: 解析字典

```
# 字符串形式的字典
dict_str = '{"name": "Alice", "age": 30}'

# 使用 literal_eval 将字符串转换为字典
dict_obj = literal_eval(dict_str)

print(dict_obj)  # 输出: {'name': 'Alice', 'age': 30}
print(type(dict_obj))  # 输出: <class 'dict'>
```

示例 3: 解析混合结构

```
# 字符串形式的复杂结构
complex_str = '[{"key": (1, 2, 3)}, [True, False, None]]'

# 使用 literal_eval 将字符串转换为复杂结构
complex_obj = literal_eval(complex_str)

print(complex_obj)  # 输出: [{'key': (1, 2, 3)}, [True, False, None]]
print(type(complex_obj))  # 输出: <class 'list'>
```

错误处理

如果尝试解析的字符串不是一个有效的 Python 字面量，`literal_eval` 会抛出 `ValueError` 异常。因此，在实际应用中，通常需要加上异常处理来提高代码的健壮性。

```
from ast import literal_eval

data_str = '[[1, 2, 3], [4, 5, 6], [7, 8, 9]]'

try:
    data = literal_eval(data_str)
    print(data)
except ValueError as e:
    print(f"Invalid input: {e}")
```

通过这种方式，你可以安全地将表示 Python 数据结构的字符串转换成实际的数据结构。

2.优化空间复杂度的思路

可以将设置上下左右四个方向的界限，使得空间复杂度优化为O（1），且无需复杂的推断转弯条件

```
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix: return []
        l, r, t, b, res = 0, len(matrix[0]) - 1, 0, len(matrix) - 1, []
        while True:
            for i in range(l, r + 1): res.append(matrix[t][i]) # left to right
            t += 1
            if t > b: break
            for i in range(t, b + 1): res.append(matrix[i][r]) # top to bottom
            r -= 1
            if l > r: break
            for i in range(r, l - 1, -1): res.append(matrix[b][i]) # right to
left
            b -= 1
            if t > b: break
            for i in range(b, t - 1, -1): res.append(matrix[i][l]) # bottom to
top
            l += 1
            if l > r: break
        return res
```

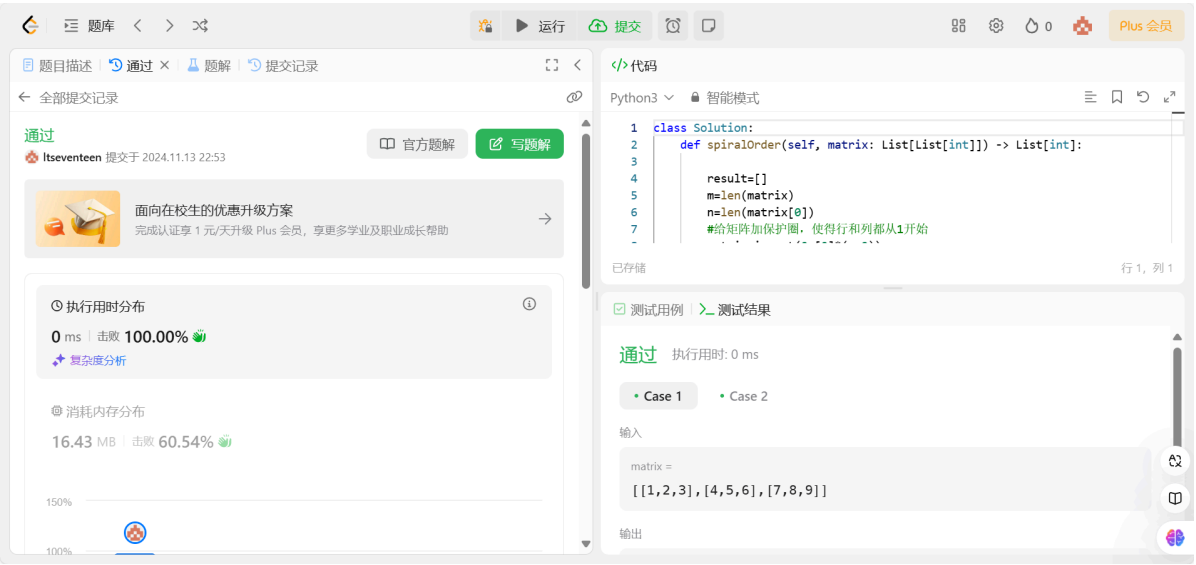
代码：

```
'''
本地运行时使用
from ast import literal_eval
from typing import List
'''

class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:

        result=[]
        m=len(matrix)
```

```
        n=len(matrix[0])
        #给矩阵加保护圈，使得行和列都从1开始
        matrix.insert(0,[0]*(n+2))
        matrix.append([0]*(n+2))
        for i in range(1,m+1):
            matrix[i].insert(0,0)
            matrix[i].append(0)

        #从左上角开始遍历
        i=1
        j=1
        result.append(matrix[1][1])
        while True:
            j+=1
            while i+j<=n+1:
                #从左往右
                result.append(matrix[i][j])
                j+=1

            j-=1 #还原端点处j的值
            #边界判断
            if i+j==n+1 and i-j==m-n:
                break

            i+=1
            while i-j<=m-n:
                #从上往下
                result.append(matrix[i][j])
                i+=1

            i-=1 #还原端点处i的值
            #边界判断
            if i-j==m-n and i+j==m+1:
                break

            j-=1
            while j+i>=m+1:
                #从右往左
                result.append(matrix[i][j])
                j-=1
            j+=1 #还原端点处j的值
            #边界判断
            if j+i==m+1 and i-j==1:
                break

            i-=1
            while i-j>=1:
                #从下往上
                result.append(matrix[i][j])
                i-=1
            i+=1 #还原端点处i的值
            #边界判断
            if i-j==1 and j+i==n+1:
                break

        return result
```

```
'''
下面部分的代码为本地测试时使用
matrix=literal_eval(input())
solution=Solution()

result=solution.spiralOrder(matrix)
print(result)
'''
```

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：注意到有垃圾的地方最多有20个，所以我们可以遍历这20个有垃圾的点，研究为了把对应点的垃圾炸到炸弹可以安放的位置。以威力是1的炸弹为例，则垃圾点周围一圈以及该点都可以放炸弹，然后把对应的垃圾数目加到这一圈点上，最后找到得分最高的点就是炸毁垃圾最多的点

耗时：30min

启示

获得矩阵中最大值的写法：

```
maxk = max(max(l) for l in board)
num = sum(l.count(maxk) for l in board)
print(num, maxk)
```

或

```
a = list(c.values())
s = max(a)
print(a.count(s), s)
```

或

```
max_num=0
max_waste=0
for i in range(1025):
    for j in range(1025):
        if matrix[i][j]>max_waste:
            max_waste=matrix[i][j]
            max_num=1
        elif matrix[i][j]==max_waste:
            max_num+=1

print(max_num,max_waste)
```

代码：

```
d=int(input())
n=int(input())
matrix=[[0]*1025 for _ in range(1025)]

dx=[_ for _ in range(-d,d+1)]
dy=[_ for _ in range(-d,d+1)] #从-d到d是垃圾周围安放炸弹可以炸毁那一堆垃圾的范围

for _ in range(n):
    x,y,i=map(int,input().split())
    for m in dx:
        for n in dy:
            if 0<=x+m<1025 and 0<=y+n<1025: #判断是否超出边界
                matrix[x+m][y+n]+=i #把炸毁垃圾的数量加给对应的投放点

#输出清理垃圾最多的投放点数目和清理垃圾数目
max_num=0
max_waste=0
for i in range(1025):
    for j in range(1025):
        if matrix[i][j]>max_waste:
            max_waste=matrix[i][j]
            max_num=1
        elif matrix[i][j]==max_waste:
            max_num+=1

print(max_num,max_waste)
```

代码运行截图 （至少包含有"Accepted"）

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：先把原数组只有1个数的特殊情况解决。然后考虑原数组至少有2个数的情况，研究以第i个数结尾的摆动子序列，最后是上升的子序列为dp[i][0]，最后是下降的子序列是dp[i][1]。因为两个数的大小关系要么大于要么小于要么等于，所以只需要考虑三种情况：num[i]>num[i-1],num[i]<num[i-1],num[i]==num[i-1]

num[i]>num[i-1]:dp[i][0]=dp[i-1][1]+1；dp[i][1]=dp[i-1][1]

num[i]<num[i-1]:dp[i][0]=dp[i-1][0]；dp[i][1]=dp[i-1][0]+1

num[i]==num[i-1]:dp[i][0]=dp[i-1][0]；dp[i][1]=dp[i-1][1]

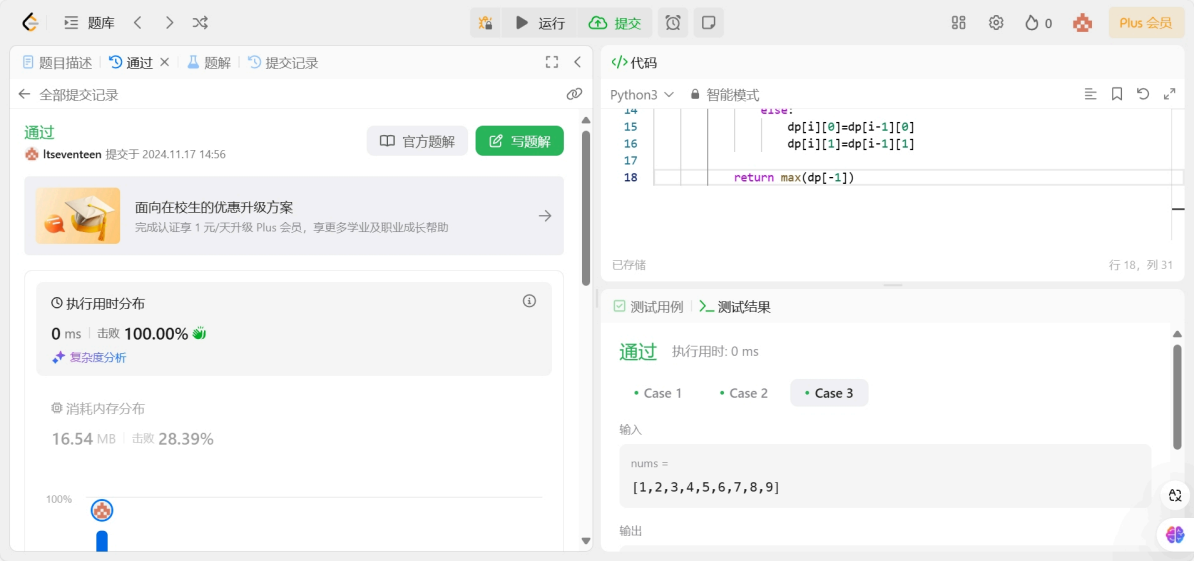
或者可以改成两个dp数组，up和down

代码：

```
#用于本地运行
#from typing import List
#from ast import literal_eval

class Solution:
    def wiggleMaxLength(self, nums: List[int]) -> int:
        if len(nums)==1:
            return 1
        else:
            dp=[[1]*2 for _ in range(len(nums))]
            for i in range(1,len(nums)):
                if nums[i]>nums[i-1]:
                    dp[i][0]=dp[i-1][1]+1
                    dp[i][1]=dp[i-1][1]
                elif nums[i]<nums[i-1]:
```

```
                    dp[i][1]=dp[i-1][0]+1
                    dp[i][0]=dp[i-1][0]
                else:
                    dp[i][0]=dp[i-1][0]
                    dp[i][1]=dp[i-1][1]

            return max(dp[-1])

#用于本地运行
#s=Solution()
#print(s.wiggleMaxLength(literal_eval(input())))
```

代码运行截图 （至少包含有"Accepted"）



CF455A: Boredom

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

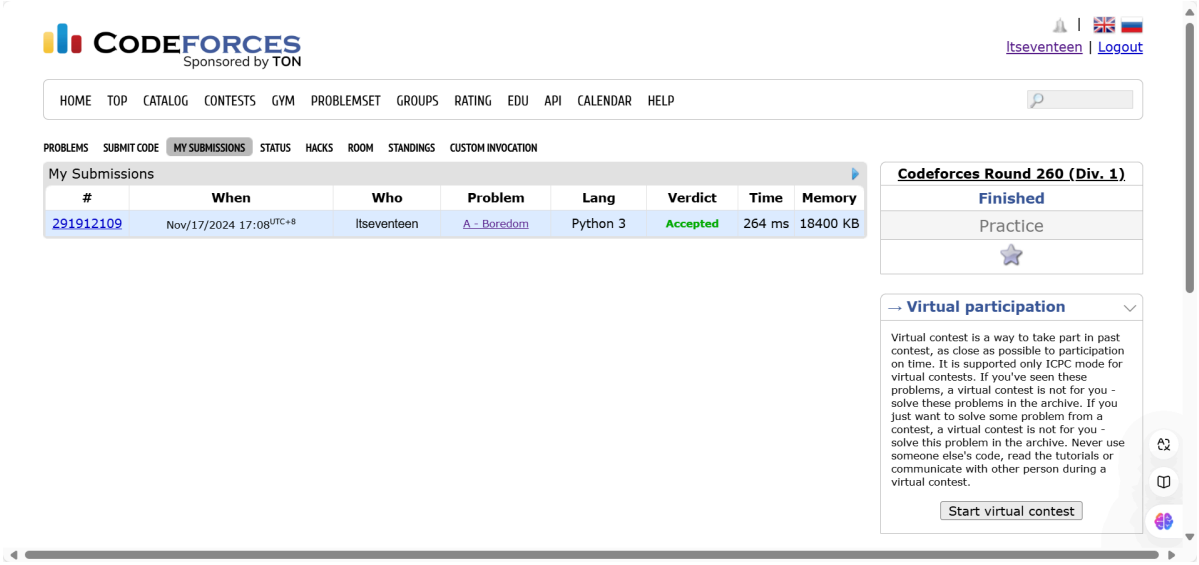
思路：大脑空空先写一个试试））先统计所有数字出现的次数，然后建一个dp数组，dp[i][0]记不加这个数目前的最大和，dp[i][1]记加了这个数目前的最大和

代码：

```
n=int(input())
l=list(map(int,input().split()))
s=[0]*100001

#统计出现次数
for i in l:
    s[i]+=1

#得分动态规划
dp=[[0,0] for _ in range(100001)]
```

```
for i in range(1,100001):
    #dp[i][0]表示值为i的元素不选，dp[i][1]表示值为i的元素选
    dp[i][0]=max(dp[i-1][0],dp[i-1][1])
    dp[i][1]=dp[i-1][0]+s[i]*i

print(max(dp[100000][0],dp[100000][1]))
```

代码运行截图 （至少包含有"Accepted"）



02287: Tian Ji -- The Horse Racing

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

思路：让所有的马都尽量以最小的优势胜出，如果实在找不到就让它当炮灰去消耗对面最强的马。如果我方最强的马已经可以打败对方最强的马就不要浪费炮灰，先赢了再说

耗时：不看题解大脑空空，看一眼题解思路瞬间秒了XD

代码：

```
#田忌赛马
while True:
    n = int(input())
    if n == 0:
        break
    tian=list(map(int,input().split()))
    king=list(map(int,input().split()))
    tian.sort()
    king.sort()
    count=0

    ltian=0
    rtian=n-1
    lking=0
```

```
    rking=n-1
    while ltian<=rtian and lking<=rking:
        if tian[ltian]>king[lking]:
            count+=1
            ltian+=1
            lking+=1
        elif tian[rtian]>king[rking]:
            count+=1
            rtian-=1
            rking-=1
        else:
            if tian[ltian]<king[rking]:
                count-=1
            ltian+=1
            rking-=1

    print(count*200)
```

代码运行截图 （至少包含有"Accepted"）

2. 学习总结和收获

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。

没有赶ddl交作业的一次，好耶！

感觉写代码稍微熟练一点了，第一次接触leetcode，在和它的格式搏斗的过程中顺便了解了一下ast.literal_eval的用法，以及怎么给class Solution传入参数。依然觉得题好难，需要想好久，有的想不出来心态炸烈脑壳痛……

每日选做补天中……努力提高代码熟练度至少做到简单题能快快写出来吧（哭）