Assignment #7: 矩阵、队列、贪心

Updated 1315 GMT+8 Oct 21, 2025

2025 fall, Complied by **韩旭 元培学院**

> **说明：**
>
> 1. **解题与记录：**
>
> 对于每一个题目，请提供其解题思路（可选），并附上使用Python或C++编写的源代码（确保已在OpenJudge， Codeforces，LeetCode等平台上获得Accepted）。请将这些信息连同显示"Accepted"的截图一起填写到下方的作业模板中。（推荐使用Typora https://typoraio.cn 进行编辑，当然你也可以选择Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. 提交安排：**提交时，请首先上传PDF格式的文件，并将.md或.doc格式的文件作为附件上传至右侧的"作业评论"区。确保你的Canvas账户有一个清晰可见的本人头像，提交的文件为PDF格式，并且"作业评论"区包含上传的.md或.doc附件。
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

M12560: 生存游戏

matrices, http://cs101.openjudge.cn/pctbook/M12560/

用时：20min

思路

遍历每一个细胞，并对其相邻的细胞的邻居数+1。当然也可以遍历每一个细胞并计算其邻居数。注意两个小技巧：（1）为了防止数组index溢出，可以加一层**保护圈**；（2）提前设置好neighbor相关的数组可以防止使用过多的选择分支。我直接列举了neighbors的二维坐标所有可能性，当然也可以分别列举两个坐标然后使用两层循环，即`dx = [-1, -1, -1, 0, 1, 1, 1, 0]; dy = [-1, 0, 1, 1, 1, 0, -1, -1]`。

代码

```
def compute(cell, count):
    if cell==1:
        if count<2:
            return 0
        elif count==2 or count==3:
```

```
            return 1
        else:
            return 0
    else:
        if count==3:
            return 1
        else:
            return 0


n,m=map(int,input().split())
Cells=[]
Count=[[0 for _ in range(m+2)] for _ in range(n+2)]
Ans=[[0 for _ in range(m)] for _ in range(n)]
neighbours=[(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]

Cells.append([0]*(m+2))
for i in range(n):
    cells=[0]+list(map(int,input().split()))+[0]
    Cells.append(cells)
Cells.append([0]*(m+2))

for i in range(1,n+1):
    for j in range(1,m+1):
        for x in neighbours:
            Count[i+x[0]][j+x[1]]+=Cells[i][j]

for i in range(n):
    for j in range(m):
        Ans[i][j]=compute(Cells[i+1][j+1],Count[i+1][j+1])
for i in range(n):
    print(" ".join(map(str,Ans[i])))
```

代码运行截图

#50480757提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 50480757
题目: M12560
提交人: 22n2200017737
内存: 3964kB
时间: 46ms
语言: Python3
提交时间: 2025-10-21 15:41:03

源代码

```
def compute(cell, count):
    if cell==1:
        if count<2:
            return 0
        elif count==2 or count==3:
            return 1
        else:
            return 0
    else:
        if count==3:
            return 1
        else:
            return 0


n,m=map(int,input().split())
Cells=[]
Count=[[0 for _ in range(m+2)] for _ in range(n+2)]
Ans=[[0 for _ in range(m)] for _ in range(n)]
neighbours=[(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]
Cells.append([0]*(m+2))
for i in range(n):
    cells=[0]+list(map(int,input().split()))+[0]
    Cells.append(cells)
Cells.append([0]*(m+2))
for i in range(1,n+1):
    for j in range(1,m+1):
        for x in neighbours:
            Count[i+x[0]][j+x[1]]+=Cells[i][j]
for i in range(n):
    for j in range(m):
        Ans[i][j]=compute(Cells[i+1][j+1],Count[i+1][j+1])
for i in range(n):
    print(" ".join(map(str,Ans[i])))
```


English 帮助 关于

M04133:垃圾炸弹

matrices, http://cs101.openjudge.cn/pctbook/M04133/

用时：25min

思路

这道题是上一题的升级版本，因为炸弹的辐射范围扩展为了$1 \leq d \leq 50$。由于垃圾的位置相对稀疏，这时如果直接遍历每个格点计算炸弹所能清除的垃圾数目，时间复杂度过高。所以，我们采用和上一题类似的方法，对于每一处垃圾，增加其周围距离为d的格点对应的垃圾数量。然后遍历垃圾数量的计数数组，统计出所能清除垃圾的最大值和最大值出现的次数即可。

代码

```
d=int(input())
```

```python
n=int(input())
count=[[0 for _ in range(1025+2*d)] for _ in range(1025+2*d)]
for _ in range(n):
    x,y,i=map(int,input().split())
    for c in range(-d,d+1):
        for l in range(-d,d+1):
            count[x+d+c][y+d+l]+=i
bomb_count=0
litter_count=0
for j in range(d,1025+d):
    for k in range(d,1025+d):
        if count[j][k]>litter_count:
            litter_count=count[j][k]
            bomb_count=1
        elif count[j][k]==litter_count:
            bomb_count+=1
print(bomb_count, litter_count)
```

代码运行截图

#50547685提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
d=int(input())
n=int(input())
count=[[0 for _ in range(1025+2*d)] for _ in range(1025+2*d)]
for _ in range(n):
    x,y,i=map(int,input().split())
    for c in range(-d,d+1):
        for l in range(-d,d+1):
            count[x+d+c][y+d+l]+=i
bomb_count=0
litter_count=0
for j in range(d,1025+d):
    for k in range(d,1025+d):
        if count[j][k]>litter_count:
            litter_count=count[j][k]
            bomb_count=1
        elif count[j][k]==litter_count:
            bomb_count+=1
print(bomb_count, litter_count)
```

基本信息

#: 50547685
题目: M04133
提交人: 22n2200017737
内存: 14772kB
时间: 265ms
语言: Python3
提交时间: 2025-10-25 11:09:39

English 帮助 关于

M02746: 约瑟夫问题

implementation, queue, http://cs101.openjudge.cn/pctbook/M02746/

用时：8min

思路

如果问题中有首尾相接的环状结构，则可以用队列（queue）来进行模仿。我们把猴子的序号放进队列，然后依次把队头的猴子拿出来放回队尾，并进行计数。当数到第m个的时候，就直接弹出，不再放回。以此类推，直到队列中只剩下一个猴子。

代码

```
from collections import deque
while True:
    n,m=map(int,input().split())
    if n==0 and m==0:
        break
    else:
        monkeys=deque(range(1,n+1))
        count=0
        while len(monkeys)>1:
            for _ in range(m-1):
                tmp=monkeys.popleft()
                monkeys.append(tmp)
            monkeys.popleft()
        print(monkeys[0])
```

代码运行截图

M26976:摆动序列

greedy, http://cs101.openjudge.cn/pctbook/M26976/

用时：40min

思路

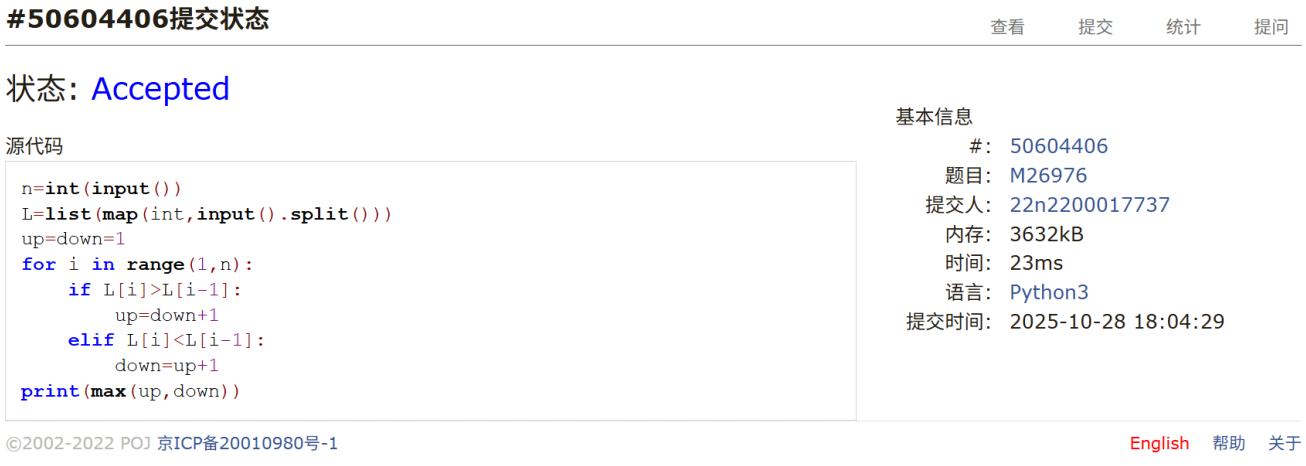
这道题的正确思路不太好想，我们首先分析摆动子序列的特征。在全序列里面，不构成摆动的部分主要是连续上升或者连续下降的部分，如果我们标记前面的上升或下降，似乎可以有效筛选出摆动子序列。进而，如果要筛选最长的摆动子序列，我们就需要维护一个目前已知的最长可能的子序列，而这个子序列和后面的联系最重要的就是它最后是上升还是下降。于是，我们不妨维护两个变量，即已知上升结尾的最长子序列长度（`up`），和已知下降结尾的最长子序列长度（`down`）。

我们需要认定如果要用贪心的话，基本上只需要$O(n)$的循环。然后我们考虑对于每一个元素`L[i]`，我们能知道的主要是它相比于前一个元素是上升了还是下降的。关键在于，**假设它比上一个元素大，那么它一定可以接续在前面的某个以下降结尾的最长子序列里**。这就是为什么局部最优解就是全局最优解，因此我们只需要把已知的上升结尾的最长子序列长度加一即可。反之同理。明确了这一思路，我们最后只需要比较`up`和`down`哪个更长即可。

代码

```
n=int(input())
L=list(map(int,input().split()))
up=down=1
for i in range(1,n):
    if L[i]>L[i-1]:
        up=down+1
    elif L[i]<L[i-1]:
        down=up+1
print(max(up,down))
```

代码运行截图



#50604406提交状态

查看　提交　统计　提问

状态: Accepted

源代码

```
n=int(input())
L=list(map(int,input().split()))
up=down=1
for i in range(1,n):
    if L[i]>L[i-1]:
        up=down+1
    elif L[i]<L[i-1]:
        down=up+1
print(max(up,down))
```

基本信息

#: 50604406
题目: M26976
提交人: 22n2200017737
内存: 3632kB
时间: 23ms
语言: Python3
提交时间: 2025-10-28 18:04:29

English　帮助　关于

T26971:分发糖果

greedy, http://cs101.openjudge.cn/pctbook/T26971/

（本题暂时跳过）

1868A. Fill in the Matrix

constructive algorithms, implementation, 1300, https://codeforces.com/problemset/problem/1868/A

（本题暂时跳过）

2. 学习总结和收获

本周其他事情有点忙，只来得及做了四个题，包括矩阵操作、队列模拟和贪心算法三类内容，整体难度比前几次作业更高，尤其在“摆动序列”一题中，我更加体会到**从直觉到贪心逻辑的转化**是具体如何发生的。

在前两题中（生存游戏、垃圾炸弹），主要考察了如何利用**二维数组与邻域遍历**来减少分支判断。我逐渐掌握了构造“保护圈”和“邻居坐标表”的技巧，让代码更简洁、更不容易越界。约瑟夫问题则进一步巩固了我对**队列结构**的语法和操作的激烈。通过 `deque` 的“旋转”操作，可以模拟出循环淘汰的过程。

“摆动序列”是本次作业里最有收获的一题。一开始我从定义出发，试图写出暴力（或类似动态规划）的版本，但其实没有抓住“贪心”的核心。后来我意识到，关键不是在于记录每一步的变化，而在于**抓住前后趋势的变化**，并维护两种可能的最大值（这个思路有点像第四次作业chips on the board那个题）。当我把问题转化为维护“上升结尾”和“下降结尾”两个变量后，整个逻辑就突然清晰了：每次上升时更新 `up = down + 1`，每次下降时更新 `down = up + 1`，这其实正是贪心的体现。