Assignment #3: 惊蛰 Mock Exam

Updated 1641 GMT+8 Mar 5, 2025

2025 spring, 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
考试成绩：AC5

> **说明：**
>
> 1. **惊蛰月考**：AC6<mark>（请改为同学的通过数）</mark> 。考试题目都在“题库（包括计概、数算题目）”里面，按照数字题号能找到，可以重新提交。作业中提交自己最满意版本的代码和截图。
>
> 2. **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。**
>
> 3. **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。
>
> 4. **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

E04015: 邮箱验证

strings, http://cs101.openjudge.cn/practice/04015

思路：一道较为简单的题目，但做起来要颇为的细心，不能遗漏掉任何条件（比如我就不小心把.不能在@前面一个字符这一个条件给忽略了，WA 了蛮久）；
另外，一定要用 try...except EOFError...这个语句验证是否还在输入数据，否则会 RE！

用时：大概 15mins

代码：

```python

```
```
def verify(alpha):
    t = len(alpha)
    status = False # 初始状态设为 no
    if alpha.count("@") == 1:
        x = alpha.index("@")
        if x != 0 and x != t - 1 and alpha[x:].count(".") >= 1:
           if alpha[0] != "." and alpha[-1] != "." and alpha[x - 1] != "." and alpha[x
+ 1] != ".":
                status = True
    if status == True:
        print("YES")
    else:print("NO")

while True:
    try:
        alpha = list(map(str,input())) # 邮箱里的所有字符
        if alpha: # 检查是否还在输入数据
            verify(alpha)
    except EOFError:
        break
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M02039: 反反复复

implementation, http://cs101.openjudge.cn/practice/02039/

思路：对于最终输出结果中的每个字母，先判断它在第几行（用它所在的位置除以列数并取整即可），然后根据他所在的行数判断它所在的列数：如果是偶数行，则正着排列进“列所组成的 list”内，如果是奇数行则反着排进去；最后，把所有的列都整合好，合并成一行解密后的信息即可~

用时：10mins

代码：

```python

```

```
n = int(input()) # 列数
alpha = list(map(str,input()))
k = len(alpha) // n # 行数

columns = [] # 里面有 n 个空列表，每个代表一列
for i in range(n):
    columns.append([])
```

```
for i in range(n * k): # 考虑第 i 个字母
    if i // n % 2 == 0: # 如果在第偶数行，则正着来
        a = i - (i // n) * n # 在第 a 列
        columns[a].append(alpha[i])
    else:
        a = (n - 1) - (i - (i // n) * n) # 在第 a 列
        columns[a].append(alpha[i])

# 接下来，把不同的列拼起来即可
words = []
for i in range(n):
    word = "".join(map(str,columns[i]))
    words.append(word)
print("".join(map(str,words)))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M02092: Grandpa is Famous

implementation, http://cs101.openjudge.cn/practice/02092/

思路：先建立一个字典，用于记录每个人的获选次数；随后，对这个字典的第二项（每个人的获选次数）进行排序（倒序）；然后，找出所有的并列第二名，并根据他们的序号进行排序；最后，将他们所有人的序号输出在同一行即可~

用时：大概 15mins

代码：

```python

```
def second(n, m):
    freq = {} # 一个记录每个人获选次数的字典
    for i in range(n):
        data = list(map(int,input().split()))
        for num in data:
            if num not in freq:
                freq[num] = 1
            else:freq[num] += 1

    sorted_freq = sorted(freq.items(), key = lambda item:item[1], reverse = True)
    # 这一步必须要有.items()，否则如果只有 freq，那么会对 keys 排序！
    # p.s.这一步生成的是一个由元组构成的列表，不是字典了！

    a = sorted_freq[1][1] # 第二名的获选次数
    b = len(sorted_freq) # 总人数
    good = [] # 所有获得并列第二名的人
    i = 1
    while i < b:
        if sorted_freq[i][1] == a:
            good.append(sorted_freq[i][0]) # 将所有第二名加进获奖名单
            i += 1
        else:break
    good.sort()
    print(" ".join(map(str,good)))

while True:
    n, m = map(int,input().split())
    if n**2 + m**2 == 0:
        break
    else:second(n, m)
```
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M04133: 垃圾炸弹

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

思路：方法很直接，挨个枚举每个路口放炸弹能够清除多少垃圾，然后找出最大者即可。

用时：10mins（因为之前做过，第一次做的时候做了好一会）

代码：

```python

```

```
d = int(input()) # damage area
n = int(input())
junk = []
for i in range(n):
    junk.append(list(map(int,input().split())))

best = [] # 最佳垃圾投放点
```

```
now = 0 # 目前最多能够清理的垃圾的数目
for x in range(1025):
    for y in range(1025): # 遍历全世界所有区域
        amount = 0 # 这个区域能够清理的垃圾数目
        for location in junk: # 对每个区域，只需要找到有哪些垃圾在他的爆炸范围内即可
            if abs(location[0] - x) <= d and abs(location[1] - y) <= d:
                amount += location[2]
        if amount > now: # 如果这是迄今为止最好的点，则更新 now 数量、以及 best 列表
            now = amount
            best = [(x, y)]
        elif amount == now: # 如果这个点恰好是并列第一，则在 best 里加入他
            best.append((x, y))

print(f"{len(best)} {now}")
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T02488: A Knight's Journey

backtracking, http://cs101.openjudge.cn/practice/02488/

思路：对于每一个样例，先对起点格子按照字典序进行枚举（先列后行）；确定了起点之后，用 DFS 方法查找可能的路径，在找到一个路径之后即可把 found 设为 1，终止所有查找。注意，在查找的过程中，一定要使用字典序来挨个遍历；这样才能保证最终输出的那一个路径是字典序最靠前的那个路径。最后，把找到的那个路径翻译成标准表示方法即可。

用时：大概 1hour 多一点

代码：

```python
```

```
import sys
sys.setrecursionlimit(10**5) # 设置递归深度限制（默认 1000）

n = int(input()) # 样列个数
found = [0] * n # 记录每个样例的状态（1 已经找到，0 还未找到）
results = [] # 每个样例的路径存储列表
for i in range(n):
    results.append([])

# 一定要先定义 found，再使用 dfs！！

def dfs(route, p, q, i):
    global found
    global result
    if found[i] == 1: # 如果这组样例在此之前就已经找到了
        return # 则不用再找了
    if len(route) == p * q:
        found[i] = 1
        results[i].append(route)
        return
    else:
        a, b = route[-1][0], route[-1][1] # 当前的格子
        dirc = [(-1,-2), (1,-2), (-2,-1), (2,-1), (-2,1), (2,1), (-1,2), (1,2)] # 按字典序枚举方向，列优先
        ok = [] # 下一步可以走到的地方
        for dx, dy in dirc:
            x, y = a + dx, b + dy
            if x in range(p) and y in range(q) and (x,y) not in route:
                ok.append((x,y))

        if ok: # 如果有下一步可以走到的地方，就继续
            for x, y in ok:
```

```
                new_route = route[:] # 创建一个 route 的副本，作为其分支
                new_route.append((x,y))
                dfs(new_route, p, q, i)
                if found[i] == 1: # 这个路径探索完之后，检查是否已经找到，如果有则结束，如果没有则继续
                    break
        else: # 否则，就把当下这个格子删掉！（走到死胡同了！）
            route.pop() # 要记得回溯！！！
            return

# 一定要先定义 dfs，再使用它！！

for i in range(n):
    p, q = map(int,input().split())
    for y in range(q):
        for x in range(p): # 按照字典序枚举不同的初始格子，列优先
            if found[i] == 1: # 如果其他的起点已经找到了，则不用再找了
                break # 这样一来，每个样例就只会输出一个可行路径！
            else:
                route = [(x,y)]
                dfs(route, p, q, i)

    # 开始输出
    print(f"Scenario #{i + 1}:")
    if found[i] == 0:
        print("impossible")
    else:
        route = results[i][0]
        readable = [] # 将路径转化为标准写法
        for block in route:
            if block[1] == 0: # 先添加列
                readable.append("A")
            elif block[1] == 1:
                readable.append("B")
            elif block[1] == 2:
                readable.append("C")
            elif block[1] == 3:
                readable.append("D")
            elif block[1] == 4:
                readable.append("E")
            elif block[1] == 5:
                readable.append("F")
            elif block[1] == 6:
                readable.append("G")
            elif block[1] == 7:
                readable.append("H")
            # p.s.因为总共至多 26 个格子，而两行和一行都不可能对，因此至多只有八列！
```

```
            readable.append(block[0] + 1) # 添加行
        print("".join(map(str,readable)))
    print() # 别忘了要空一行！！
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T06648: Sequence

heap, http://cs101.openjudge.cn/practice/06648/

思路：先用最小堆的方法把两个序列的情况搞定（需要优化到 nlnn 量级，否则会 TLE！），接着再把前两个序列中最小的 n 个和视为一个新的序列，与第三个序列进行同样的上述操作；随后一直进行 m-1 次同样的操作就可以完成这道题目。

用时：2~3hours

代码：

```python

```

```
import heapq

# 以下的函数用于查找两组负数的和中最大的 n 个和（m=2 版本）
def two(seq1, seq2, n):
    good = [] # 最小的 n 个和的储存处
    temp = [(seq1[0] + seq2[0], 0, 0)] # 一些 potential 最小和的储存处
    visited = set((0, 0)) # 已经考虑过的和，用集合来记录
    heapq.heapify(temp) # 先把 temp 变成堆
    while len(good) < n:
        num, i, j = heapq.heappop(temp) # 当下 temp 列表中的最小和，已经可以正式的成为 n 个最小和之一了
        good.append(num)
        if i in range(n - 1) and (i + 1, j) not in visited: # 左脚往上迈一步，并把它加入潜在最小和中
            heapq.heappush(temp, (seq1[i + 1] + seq2[j], i + 1, j))
            visited.add((i + 1, j))
        if j in range(n - 1) and (i, j + 1) not in visited: # 左脚往上迈一步，并把它加入潜在最小和中
            heapq.heappush(temp, (seq1[i] + seq2[j + 1], i, j + 1))
            visited.add((i, j + 1))
    return good # 由于每一轮中都往 good 里添加了一员，因此 n 轮之后循环就结束了，时间复杂度很低！

def find(m, n):
    sequences = [] # 所有的序列
    for _ in range(m):
        data = list(map(int,input().split()))
        data.sort() # 排序，这一步占 m*n*lnn 复杂度，可以接受
        sequences.append(data)

    # 现在摆在面前的是 m 组排序好的数（均为负数）；先考虑前两组数，找出与前两组数构成的最大的 n 个和；
    # 随后，把这 n 个和放进一个列表里，并把它视为 seq1，再把下一组数视为 seq2 即可周而复始的继续了~
    good = sequences[0][:] # 先把第一个列表视为初始的 n 个最小的和
    for i in range(1,m):
        good = two(good, sequences[i], n) # 更新 good 列表

    answer = [] # 最终的答案（要把负数全部变回正数）
    for num in good:
        answer.append(num)
    answer.sort()
    print(" ".join(map(str,answer)))

k = int(input())
for i in range(k):
    m, n = map(int,input().split())
```

```
find(m, n)
```

代码运行截图 ==（AC 代码截图，至少包含有"Accepted"）==

2. 学习总结和收获

<mark>如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算 2025spring 每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。</mark>

本次考试因为我的上机课时间和我的体育课冲突了，因为很遗憾没能在线下准时参加考试，不过我在线上也进行了一次全真模拟，按照规定的时间进行了测试。最终的结果也还是不错，ac了五道，第五题更是卡着时间做完的，蛮开心的~

回到正题，对于这次考试的题目而言：第一题看似很简单（如果我没记错的话应该也是计概的原题吧哈哈），但做的过程中一定要细心，不能轻敌，否则就会像我一样在这道题卡十几分钟（笑死）；我卡住是因为其中有一个条件规定了“@和.不能挨在一起”，但是因为这个条件和“@后面必须有.”这个条件放在了一起，因此我就只考虑到了@后面一个符号不能是“.”，而忽略了前一个也不能是“.”。。。浪费了不少时间；
另外，这道题还有一个坑点，就是代码的最后一定要用 try...; except EOFError...这个语句来验证是否还在输入，否则会 RE！（我花了好几分钟才回忆起来写法 qwq）

随后的第二题和第三题我都做的十分顺利，这两道题的方法都很直接，我觉得思路都很好想到，没有什么太多需要注意的。唯一一个我卡住的地方就是在第三题的一个细节，我有点忘记了该

如何对字典中某一项进行排序的方法，我卡在那里回忆了好一会，但最终还是有惊无险地回忆起来了哈哈~（看来还是得时常复习，不然之前学的模板很容易忘）

第四题的话，则是一个很有趣的题目。如果没记错的话，这也是之前计概课程中的一道作业题；当时我 TLE 了好多次才最终 AC，因此给我留下了非常深刻的印象。我个人感觉这是一道很容易 TLE 的题目，因为路口太多了，稍有不慎就会超时；但是最终的方法却出乎意料的简单，如果能直接想到那个方法，很快就能 AC。其实这个方法十分的朴素，就是挨个枚举每个路口放炸弹能够清除多少垃圾，然后找出最大者即可。
但是，思路归思路，具体实操的过程中要注意两个点：
第一，就是考虑每个路口放炸弹能够清除多少垃圾的时候，只需要在那个垃圾列表里找每个垃圾是否在爆炸范围内就可以了，而并不是搜索完所有这个炸弹范围内的路口是否有垃圾；因为如果是那样的话就得搜索 d 的平方量级个数的路口，最多有可能要搜索 2500 次，非常的耗时间。但是如果只在垃圾列表里搜索，顶多只用搜索 20 次，非常的快速；而在当时，就是因为这么一步小小的优化，就卡了我半个多小时。。
第二，就是在查找有多少个路口能够清除最多的垃圾的时候，可以使用一个实时更新的列表来记录所有当下能够清除最多垃圾的路口，以及这个路口到底能够清除多少垃圾。这样一来，每枚举完一个路口，就能立刻判断这个路口是否是当下能够清除最多垃圾的路口，并对其进行相应的处理（如果是，则把之前的列表清空，并换为这个路口；如果仅仅是并列最多的路口，则在列表里加入它即可；如果不是，则不用管它）；这样边枚举边处理的思想，可以大幅减少时间复杂度，进一步优化方法。

对于第五题而言，我认为这是一道蛮有难度的题目。首先，题目中所需要使用的 DFS 方法是上个学期中我掌握的不太好的一个点；虽然说大部分题目都可以用类似的模式和套路做出来，但是其实里面蕴含的一些内涵和思想，以及不断嵌套的过程还是有点繁琐，我花了一点时间才回忆起来并研究明白每一步的递归与最终的 return 的过程的逻辑。
在做这道题的过程中，因为我对 DFS 方法稍微有点遗忘了，因此我在最开始做的时候在遍历的过程中忘记回溯了，这导致我最终输出的路径有错误，然后 debug 了很久。不仅如此，这道题目里还有两个比较坑的点：
第一个点就是在于一定要输出字典序最靠前的那个路径。但是，在读题的过程中我一开始没有发现题目中强调了这个点，因此 WA 了很久；
第二个点就是，在输出的过程中，不同的案例输出的部分之间还得空一行；这个点我也是检查了三四分钟才发现自己到底为什么 Presentation error，有点烦人（笑死）。。
总的来讲，这道题还是花了我不少时间，但是也学到了很多东西（除了复习了 DFS 方法的内在逻辑以外，我还知道了：必须要先定义一个全局变量，才能在 dfs 函数内部调用它，否则会 compile error；同样，必须得先定义 dfs 函数，才能在外部的代码中使用它，否则会 re。。），可以说做这道题的过程还是蛮有收获的~

第六题我在考试规定的时间里没时间做了，考完之后也是做了好久才做出来。我个人觉得这是一道相当有难度的题目，光是想到思路（先对前两个数组进行处理，然后把前两个数组中最小的 n 个和又视为一个新的数组、并与第三个数组合并，以此类推）就颇为困难，但即使想到了这个思路也很有可能 TLE。我就是花了接近一个小时才想到思路，并且想到思路之后也整整 TLE 了六次 qwq。。主要的优化点还是在合并两个数组的情况，我一开始给的方法是朴素的 n^2，后来用最小堆方法优化成了 n(lnn)^2，但还是 TLE。。直到最后我实在不会做了，就请教了一个信科的同学，他给我讲了一个非常巧妙的贪心+heap 的方法，把时间复杂度降到了 nlnn 量级，才最终 AC；据他所说，这个方法看似巧妙，但还是蛮常用的。在做这道题和与他交流的过程中，我还是学到了蛮多~