Assignment #C: 202505114 Mock Exam

Updated 1518 GMT+8 May 14, 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
AC 题目数：5 道

> **说明：**
>
> 1. **月考**：AC?<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

E06364: 牛的选举

http://cs101.openjudge.cn/practice/06364/

思路：先用两个字典记录每一轮所有牛的数据，接着对第一个字典排序并筛选出前 k 头牛，并记录前 k 头牛的编号；最后，把这 k 头牛筛选出来，将他们的数据从第二个字典中提取出来，并再排一次序，取出第一名的编号即可。

时间：10mins

代码：

```python

```
n, k = map(int,input().split())
round_1 = {}
round_2 = {}
for i in range(n):
    a, b = map(int,input().split())
    round_1[i + 1] = a
    round_2[i + 1] = b

sorted_1 = sorted(round_1.items(),key=lambda item:-item[1]) # 排倒序（返回的是一个tuple组成的列表）
# 要记得加items()！不然只会对key排序！
good = [] # 第一轮晋级的牛
for i in range(k):
    good.append(sorted_1[i][0])

finale = {} # 只记录晋级者的数据
for num in good:
    t = round_2[num]
    finale[num] = t
sorted_finale = sorted(finale.items(),key=lambda item:-item[1]) # 对第二轮结果进行排序
print(sorted_finale[0][0])
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>
```

M04077: 出栈序列统计

http://cs101.openjudge.cn/practice/04077/

思路：用数学归纳法可以证明：每一个合法的“push、pop”序列都唯一对应着一个出栈序列~
下面将 push 称为 0，将 push 称为 1；
注意到，要想使第一个出栈的数相同，则第一个 1 所在的位置必须相同；而同样，要想使第二个出栈的数相同，第二个 1 所在的位置也必须相同。如此下去，假如有两个相同的出栈序列，则他们对应的 0-1 操作序列一定相同！
因此，只需要计算有多少个操作序列即可。
操作序列满足：有 2n 个数，n 个 0 和 n 个 1,0 的个数永远不小于 1 的个数；记 n 时的情况数为 an；突破口：每次 push 进 n 的时候，下一步一定是将 n 弹出来，因为外面已经没有数了！
因此，对于 n-1 的每个情况，在输入完所有数之后（序列只剩“111...”），可以往任何一个地方加入一个“01”！（将 n 时的每个序列视为 n-1 的每个序列的“111...”尾巴部分插入一个“01”！）
因此，如果将 n 时，序列最后有 m 个 1（尾巴长度为 m）的操作序列个数为 anm；
则 an=an1+an2+...+ann，
并且，a(n+1)1=an1+an2+...+ann(在每个序列后面加一个 01)
a(n+1)2=an1+an2+...+ann(在每个尾巴长度大于等于 1 的序列的倒数第 1 个数前面加一个 01)
a(n+1)3=an2+...+ann(在每个尾巴长度大于等于 2 的序列的倒数第 2 个数前面加一个 01)
...一直到 a(n+1)(n+1)=ann(在每个尾巴长度大于等于 n 的序列的倒数第 n 个数前面加一个 01)
如此递归即可！
时间：20mins

代码：

```python

```

```
n = int(input())

recursion = [[1]] # 第 i 个列表的第 j 个数就是 a(i+1)(j+1)，而 an 就是第 n-1 个列表的和!
for i in range(1,15):
    recursion.append([sum(recursion[i - 1])])
    for j in range(i):
        recursion[i].append(sum(recursion[i - 1][j:]))
print(sum(recursion[n - 1]))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M05343:用队列对扑克牌排序

http://cs101.openjudge.cn/practice/05343/

思路：按部就班的根据题目中的操作顺序进行操作即可：先进行第一次分牌，再进行第一次出牌；接着，同样的，进行第二次分牌并再进行第二次出牌，最终将结果输出就 OK 了。（输出结果的时候要小心一点，不要多了或者少了空格！）

时间：15mins

代码：

```python
```
```
n = int(input())
cards = list(map(str,input().split()))
lists_1 = [[], [], [], [], [], [], [], [], []]
for t in cards:
    a = int(t[1])
    lists_1[a - 1].append(t)

cards_1 = [] # 第一次出队后的卡牌排序
for i in range(9):
    for card in lists_1[i]:
        cards_1.append(card)

# 进行第二次分队
lists_2 = [[], [], [], []]
for t in cards_1:
    if t[0] == "A":
        lists_2[0].append(t)
    elif t[0] == "B":
        lists_2[1].append(t)
    elif t[0] == "C":
        lists_2[2].append(t)
    elif t[0] == "D":
        lists_2[3].append(t)

cards_2 = [] # 第二次出队后的卡牌排序
for i in range(4):
    for card in lists_2[i]:
        cards_2.append(card)

# 输出结果
for i in range(9): # Part1
    if lists_1[i]:
        items = [f"Queue{i + 1}:{lists_1[i][0]}"] # 所有要打印的元素
        items.extend(lists_1[i][1:])
        print(" ".join(map(str,items)))
    else:
        print(f"Queue{i + 1}:")
```

```
if lists_2[0]: # Part2
    items = [f"QueueA:{lists_2[0][0]}"]
    items.extend(lists_2[0][1:])
    print(" ".join(map(str,items)))
else:
    print(f"QueueA:")

if lists_2[1]:
    items = [f"QueueB:{lists_2[1][0]}"]
    items.extend(lists_2[1][1:])
    print(" ".join(map(str,items)))
else:
    print(f"QueueB:")

if lists_2[2]:
    items = [f"QueueC:{lists_2[2][0]}"]
    items.extend(lists_2[2][1:])
    print(" ".join(map(str,items)))
else:
    print(f"QueueC:")

if lists_2[3]:
    items = [f"QueueD:{lists_2[3][0]}"]
    items.extend(lists_2[3][1:])
    print(" ".join(map(str,items)))
else:
    print(f"QueueD:")

print(" ".join(map(str,cards_2))) # Part3
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M04084: 拓扑排序

http://cs101.openjudge.cn/practice/04084/

思路：可以使用 Kahn 算法来做这道题：先统计所有节点的入度，接着把所有入度为 0 的点放入一个最小堆中，这样每次都能优先取出编号小的点。
然后，不断从堆中取出入度为 0 的节点，加入结果序列；每处理一个点，就把它所有相邻点的入度减 1；如果相邻点的入度变为 0，就加入堆。重复这些步骤，直到堆空为止即可。
时间：25mins

代码：

```python

```
import heapq

v, a = map(int, input().split())
# 初始化图结构和入度数组
graph = [[] for _ in range(v + 1)]
in_degree = [0] * (v + 1)
for _ in range(a):
    u, w = map(int, input().split())

```
    graph[u].append(w)
    in_degree[w] += 1

heap = [] # 最小堆：用于保证编号小的点先出队
# 将所有入度为 0 的点加入堆中 for i in range(1, v + 1):
    if in_degree[i] == 0:
        heapq.heappush(heap, i)

result = [] # 排序结果
# 用 bfs 方法拓扑排序 while heap:
    node = heapq.heappop(heap)
    result.append(f'v{node}')
    for neighbor in graph[node]:
        in_degree[neighbor] -= 1
        if in_degree[neighbor] == 0:
            heapq.heappush(heap, neighbor)
print(" ".join(result))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

M07735:道路

Dijkstra, http://cs101.openjudge.cn/practice/07735/

思路：使用经典的 dij 方法即可：
用一个最小推来记录当前所有合乎题意的路径；其中每个路径的第一个数是这个路径目前走过的长度，第二个数是当前一共所花的钱，第三个数是当前所在的城市。这样一来，最小堆的堆顶永远是当前路径最短且花钱花的最少的一个路径。
另外，还需要注意的是，每一次往堆里面加入路径的时候，要检查这个路径是否合法（也就是这个路径目前所花的钱有没有超出最大预算）！
时间：45mins

代码：

```python

```
```
import heapq

m = int(input()) # maximum money
n = int(input()) # number of cities
r = int(input()) # amount of road

map1 = [] # the (i-1)th list in this map contains all the roads starting from city(i)!
# each road is in the form of "length, money, end"!
for i in range(n):
    map1.append([])
for i in range(r):
    a, b, c, d = map(int,input().split())
    map1[a - 1].append((c, d, b))

def dijkstra():
    status = False # check if the end is reachable
    heap = [(0, 0, 1)] # (a, b, c): current length, money, city
    heapq.heapify(heap)

    while heap:
        (a, b, c) = heapq.heappop(heap)
        if c == n: # if the end is reached
            if status: # if the end has been reached before, then check if this one
is better
                if a < shortest_length:
                    shortest_length = a
                elif a == shortest_length:
                    least_money = min(least_money, b)
```

```
            else: # if the end hasn't been reached before, then mark this record
                status = True
                shortest_length = a
                least_money = b

        else: # if the end is not reached
            for (x, y, z) in map1[c - 1]:
                if status:
                    if a + x < shortest_length and b + y <= m:
                    # check if a new state is legal
                      heapq.heappush(heap, (a + x, b + y, z)) # input all the legal
new states
                else:
                    if b + y <= m:
                    # check if a new state is legal
                      heapq.heappush(heap, (a + x, b + y, z)) # input all the legal
new states

    if status:
        print(shortest_length)
    else: print("-1")

dijkstra()
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T24637:宝藏二叉树

dp, http://cs101.openjudge.cn/practice/24637/

思路：不会做 qwq。。

代码：

```python

```

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>

由于这次模拟考试的时间与其他课程冲突了，因此这次考试我还是在线上考的。我在周末找了个时间，花了两个小时去模拟这场考试；最终的结果是我 AC 了 5 道题。总体来讲，我对这个成绩还是比较满意的。第 6 题时间不够了来不及做，我也后续在规定的考试时长结束之后又想了好一会，但还是没想出来能 ac 的代码，最后看了题解才恍然大悟，稍微有点遗憾。

对于前五题而言：第一题毫无疑问是比较简单的，只需要按部就班的按照题目中的要求，一步一步进行操作就可以了；无非就是对于一个第一题而言，这道题算是略微有点繁琐，要小心不要犯一些小的错误。我花的时间也稍微有点长，将近 10 分钟才 ac。

第二题的话则是一个有关于 stack 的十分经典的题目，我当时一看到这道题目，就意识到了这道题的卡塔兰数背景；但是如果直接把 catalan 数的公式给扔上去，花一分钟就把这道题目给直接 ac，然后立马就做下道题，就稍微感觉有点浪费这道题了，也无法模拟正式考试的真实环境。因此，我还是决定自己再想一个递推或者 DFS 的方法，用自己的思路把它做出来。后续我也是利用了上述提到的数学归纳法和 dp 的思想，最终把这道题目用常规的方法给完整的写了出来。虽然最终 AC 花了 20 分钟，但心里还是蛮有成就感的~

第三题则非常简单，也是和第一题一样按部就班的按照题意来操作即可。第四题和第五题则相对比较套路化，都是使用之前练习过的方法，使用类似于加权 BFS 的思路来做就 OK 了，不算很难，但也花了我一些时间；尤其是第五题，我敲代码的时候中间出了点小 bug，导致我 debug 了很久，浪费了不少时间，最终 45 分钟才 ac。还是得再多做点练习题，多提升点熟练度！