**## 1. 题目**

**### E06364: 牛的选举**

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

思路：把第几头牛、第一轮，第二轮的分数打包，并对第一轮分数进行倒序排序，选出第一轮的K头牛，再根据第二轮分数进行排序，最后输出排在最前面的牛的位置

代码：

N,K=map(int,input().split())  
  
score=[]  
for \_ in range(1,N+1):  
 Ai,Bi=map(int,input().split())  
 score.append((\_,Ai,Bi))  
  
score.sort(key=lambda x:-x[1])  
scores=score[:K]  
scores.sort(key=lambda x:-x[2])  
print(scores[0][0])

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
  
大约用时：15分钟

**### M04077: 出栈序列统计**

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

思路：用递归来尝试每一种可能的操作组合（push或pop），只要合法就继续（即push<n：还没把所有数推完&pop<push：栈里必须有数才能弹出），从初始push&pop=0开始，当pop==n代表成功构建了一个出栈序列，计数+1，这同时也是卡特兰数公式 C(n) = (2n)! / (n! \* (n+1)!)

代码：

n=int(input())  
count=0  
  
def dfs(push,pop):  
 global count  
 if pop == n: # 成功构建了一个出栈序列，计数 +1  
 count+=1  
 return  
  
 if push < n:  
 dfs(push+1,pop)  
  
 if pop < push:  
 dfs(push,pop+1)  
  
dfs(0,0)  
print(count)

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：45分钟

**### M05343:用队列对扑克牌排序**

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

思路：用了很笨的方法哈哈哈，一开始还以为是deque但是发现其实也没有必要（，就是把每个对应的卡牌加入到对应的数字，再把排序后的卡牌转换成列表继续加入到对应的字母，最后再将排序后的加入到列表

代码：

n=int(input())  
card=input().split()  
  
dict={}  
for i in range(1,10):  
 if i not in dict:  
 dict[i]=[]  
for c in card:  
 if int(c[1]) in dict:  
 x=int(c[1])  
 dict[x].append(c)  
  
for i,j in dict.items():  
 print(f"Queue{i}:{' '.join(map(str,j))}")  
  
sorted\_cards=[]  
dict2={}  
for i in range(1,10):  
 for c in dict[i]:  
 sorted\_cards.append(c)  
 suit=c[0]  
 if suit not in dict2:  
 dict2[suit]=[]  
  
for c in sorted\_cards:  
 if c[0] in dict2:  
 dict2[c[0]].append(c)  
  
print(f"QueueA:{' '.join(dict2['A'])}")  
print(f"QueueB:{' '.join(dict2['B'])}")  
print(f"QueueC:{' '.join(dict2['C'])}")  
print(f"QueueD:{' '.join(dict2['D'])}")  
  
final=dict2['A'] + dict2['B'] + dict2['C'] + dict2['D']  
print(' '.join(final))

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：30分钟

**### M04084: 拓扑排序**

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

思路：因为是拓扑结构所以选择了建图，用g列表来建图（graph[i]：从i出发能到的点）和记录每个点的入度点，有人指向它便+1，然后判断入度为0的点（最小）并加入结果列表res，接着循环判断每次取出的最小点指向的所有点，对其指向的点入度-1，直到==0便加入排序，最后输出排序后的每个点

代码：

import heapq  
  
v,a=map(int,input().split())  
  
g=[[] for \_ in range(v+1)] # 构建图  
indeg=[0]\*(v+1) # 计算入度有多少个  
  
for i in range(a):  
 a,b=map(int,input().split())  
 g[a].append(b)  
 indeg[b] += 1  
  
heap=[]  
for i in range(1,v+1):  
 if indeg[i] == 0:  
 heapq.heappush(heap,i)  
  
res=[]  
while heap:  
 node=heapq.heappop(heap)  
 res.append(f"v{node}")  
  
 for i in g[node]:  
 indeg[i] -= 1  
 if indeg[i] == 0:  
 heapq.heappush(heap,i)  
  
print(" ".join(res))

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：1小时

**### M07735:道路**

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

思路：因为要算最短路径，所以用heap，Dijkstra和金币的限制条件来做，首先初始化初始化距离数组（先设置为：路径非常大，目前不可达），然后将起点城市1，花0金币，路径长度为0，并且用这个数组来一边走路一边记账（记录金币和路径长度），找出最花最少金币又最短的路。然后用path来记录当前路径长度, 当前城市, 已经花了多少金币，用每次循环来找出有没有更短能到达终点的路径，最后选择其中最短的路径并输出

代码：

import heapq  
  
K=int(input()) # 最多能花多少金币  
N=int(input()) # 城市个数  
R=int(input()) # 路的条数  
  
graph=[[] for \_ in range(N+1)] # graph[i]：表示从城市i出发有哪些路  
  
for \_ in range(R):  
 s,d,l,t=map(int,input().split())  
 graph[s].append((d,l,t)) # 终点d，长度l，金币t  
  
MAX=10\*\*9 # 初始化距离数组（先设置为：路径非常大，目前不可达）  
dist=[[MAX]\*(K+1) for \_ in range(N+1)]  
dist[1][0]=0 # 起点城市1，花0金币，路径长度为0  
  
path=[(0,1,0)] # 当前路径长度, 当前城市, 已经花了多少金币  
  
while path:  
 length,u,coins=heapq.heappop(path)  
  
 if dist[u][coins] < length: # 如果状态比之前记录的还差就跳过  
 continue  
  
 for v,l,c in graph[u]: # 看从当前城市u出发有哪些路可以走  
 new\_coins=coins+c  
 new\_length=length+l  
  
 if new\_coins<=K:  
 if dist[v][new\_coins]>new\_length:  
 dist[v][new\_coins]=new\_length # 到达一个城市，花new\_coins金币，最短路径若更短则更新为更短的路  
 heapq.heappush(path,(new\_length,v,new\_coins))  
  
ans=min(dist[N]) # 选其中最短的  
print(ans if ans<MAX else -1)

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：1小时30分钟

**### T24637:宝藏二叉树**

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

思路：判断是选左右子节点还是选当前节点的值会更大，如果不选当前节点则左右子节点可以选或不选，如果选当前节点则不能选左右子节点。从子节点向上递归，并用字典来储存选择和不选择两种情况的最大收益，防止重复计算。最后输出递归后得到的最大值

代码：

N=int(input())  
value=list(map(int,input().split()))  
value=[0]+value # 让编号从1开始  
  
dict={} # key:节点编号，value:(不选节点时的最大值，选节点时的最大值)  
  
def dfs(i):  
 if i>N:  
 return (0, 0) # 超出节点范围，没宝藏可选  
  
 if i in dict:  
 return dict[i]  
  
 left=2\*i  
 right=2\*i+1  
  
 left\_not,left\_sel=dfs(left) # 递归计算左子树、右子树  
 right\_not,right\_sel=dfs(right)  
  
 not\_select=max(left\_not,left\_sel)+max(right\_not,right\_sel) # 不选当前节点：左右子节点可以选也可以不选，取最大  
  
 select=value[i]+left\_not+right\_not # 选当前节点：左右子节点不能选  
  
 dict[i]=(not\_select,select)  
  
 return dict[i]  
  
not\_sel,sel=dfs(1)  
print(max(not\_sel,sel))

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：1小时

**## 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>

这次因为月考的时候在外面，所以没有参加，只是点进去看了一下题目，目测自己能做出第一和第四题，结果实际做的时候果然完全不靠AI的话只能做出这两题hhh，有关图的题目虽然看懂了但是思路还是不清楚（但是基本上都有大概的思路，感觉如果正式考试的时候我能够争气一点，提早把会的题做了，努努力应该能答出一题图），第二题感觉如果不提前知道条件/公式是什么的话可能也得做很久，队列卡牌那题还以为要用deque，之后发现好像不一定就用了比较笨的方法（果然这种方法比较适合我。