

True_Final_Sum 陈勇 2021fall

前言

为了回馈课堂，同时记录学习历程，笔者整理出了此文档。内容包括笔者在闫老师的计概 B 班级——2021fall-cs101 所提交过的每一份作业文档。作为零基础学编程的同学，笔者的思路、代码、表达、风格等都不会是最好的。若读者期待参考大神的创作，此文档绝非理想选择；若读者也是零基础，此文档应该能在一些时刻带来帮助，让读者越过几道坎。在此列出四个：初识语法、二维数组（矩阵）、动态规划（dp）及深度或广度搜索（dfs/bfs）。

使用说明：

由于时间及技术限制，笔者未能设置目录超链接，读者可使用书签功能快速浏览小标题。若使用 Adobe Acrobat 打开此文档，应该可在左侧参看书签。另外，善用 Ctrl+F 检索关键词。

重要须知：

第一、任何题目请优先以老师维护的两份官方题解为准，请勿将此文档视作标准题解。Please regard teacher's answer collections as main references, DON'T regard this document as standard answer.

第二、文档内容可能存在的问题包括但不限于：复制粘贴错误、错别字、OJ 题号及链接改变（详情可咨询老师）。

第三、若读者发现此文档有误，可以咨询老师、助教或班上同学，也可以直接咨询笔者（有空才能维护）。

第四、强烈建议查阅此文档时与笔者留下的代码包中相关内容对照：True_Code_Pack_yeagle_ting2021fall.zip。

第五、笔者 Python 入门，目前也只对 Python 掌握，顶多能看懂小部分 C++ 基础语法，风格 Pythonic 勿见怪。

免责声明：

本文档仅供参考，不绝对保证资讯的准确性及完整性。若任何人因查阅此文档造成期末考试考不好，知识点理解不全面或被其误导，笔者对类似情况一概不负任何责任。若文档中有任何错误敬请见谅，望读者们关注对自己有益的部分，无须在意无用之处。

Disclaimer:

This document is just a reference, accuracy and completeness are not absolutely ensured. If anyone messed up in final exam, failed to fully understand knowledge or found misleading because of referring this document, author will not responsible for similar circumstances. If there is any mistake in this document, please excuse author, hope readers focus on the useful part, don't bother with the useless part.

在此致谢：

感谢老师及助教们对班级的管理及支持，也特别感谢所有帮助过我的、被我帮助过的、与我有过交流的同学。希望这份文档也会对读者带来一些帮助与启发。

注：笔者在期末考前曾发布过 Final_Sum 及 Code_Pack，期末考后两者内容皆有修订补充，于是取“True_”作文件名前缀以区分。另外，若读者是首次阅读此文档，建议先参阅文档最后的总结部分。

陈勇

2021fall-cs101 || 13 班线上学生
马来西亚留学生 || 环境科学与工程学院 21 级本科生
2021 年 12 月 28 日 (星期二)

Assignment#1：陈勇2100092701

文档撰写日期：2021年9月21日（星期二），中秋节

前言：作为几乎0基础的小白，第一次完成编程作业意外顺利，肯定要在交功课前仔细审视自己，所以写的中文字比较多，希望之后可以更精简易。

4A. Watermelon, brute force/math

<https://codeforces.com/problemset/problem/4/A>

解题思路：确认是初阶数学题，先看本质再看条件，后思考关键词及符号。

1. 本质：判断输入值是否为偶数（2除外）。
2. 条件：“偶数”可以简单地通过能否被2除尽来判断；“2除外”用不等于即可。
3. 关键词及符号：if 和 else (判断)、% 和 == (处理“偶数”)、!= (处理“2除外”)、其他常见语法。

```
w = int(input())
if w%2==0 and w!=2:
    print("YES")
else:
    print("NO")
```

注：其实在看submit示范时偷（顺便？）看了一眼，捕捉到了这题的code，但还是自己理了一遍，结果觉得其他方法都不如此法简洁，最终就这么用了。

71A. Way Too Long Words, strings

<http://codeforces.com/problemset/problem/71/A>

解题思路：确认是初阶字符串处理题，先看本质再看条件，后思考关键词及符号。

1. 本质：要针对n个输入字符串做长度判断后逐个变换输出。
2. 条件：“针对n个”是限定范围，“长度”要先取值后判断，“变换”要从字符串里取前后并将[“长度”值-2]作为字符串嵌入，才能得到新的字符串。
3. 关键词及符号：for 和 in (逐个处理)、if 和 else (判断)、range (处理限定范围)、len 和 > (处理“长度”)、[0] + str() + [-1] (处理“变换”)、其他常见语法。

```
n = int(input())
for word in range(n):
    word = input()
    length = len(word)
    if length > 10 :
        print(word[0] + str(length-2) + word[-1])
    else :
        print(word)
```

注：诚实说，参考了“2020fall_Codeforces_problems.pdf”里的解法，看懂了“shortcode”后很震惊，不过也理解到这题的套路跑不掉是这样，就用自己的方式写了一遍。

2750. 鸡兔同笼, math

<http://cs101.openjudge.cn/practice/2750>

解题思路：确认是初阶数学题，先看本质再看条件，后思考关键词及符号。要先分三个情况：

- 一，当a是4的倍数，则全是兔最少=a/4，全是鸡最多=a/2。
 - 二，当a是2的倍数，则若干兔+1只鸡最少=(a-2)/4 + 1 = (a+2)/4，全是鸡最多a/2。
 - 三，都不是，不用算，直接输出(0, 0)。
1. 本质：判断2的倍数或4的倍数，据各别条件输出各别公式的运算结果。
 2. 条件：“倍数”通过能否被除尽来判断，建立对应情况一&二的输出公式。
 3. 关键词及符号：if 和 elif 和 else (判断)、% 和 == (处理“倍数”)、其他常见语法。

```
a = int(input())
if a%4==0:
    print(int(a/4), int(a/2))
elif a%2==0:
    print(int((a+2)/4), int(a/2))
else:
    print(0, 0)
```

注：今天看到老师在群里发的同学作业，看了发现程序写得都一样，思路却很不同，我的理解方式比较初级繁琐，证明了数学可能是多数留学生的短板，不过我想能写出好程序的思路也算是好思路了吧。

2733. 判断闰年, math

<http://cs101.openjudge.cn/practice/2733>

解题思路：确认是初阶数学题，先看本质再看条件，后思考关键词及符号。

1. 本质：判断4的倍数(不是100的倍数)或400的倍数(是100的倍数)。
2. 条件：“倍数”通过能否被除尽就可以判断。
3. 关键词及符号：if 和 elif 和 else (判断)、% 和 == 和 != (处理“倍数”)、其他常见语法。

```
a = int(input())
if a%100!=0 and a%4==0:
    print("Y")
elif a%100==0 and a%400==0:
    print("Y")
else:
    print("N")
```

注：题目最后给的提示作用太大了(不曾知道1900和3200不是闰年)，直接翻译成程序语言就完事。不过事后反思也没想到更好的方法，猜想 if 和 elif 应该有办法再合成简化。

结语：希望有哪里不足可以获得点评与建议，我需要累积很多经验才能写出更快，狠，准的程序。

祝中秋节快乐安康！

Assignment#2：陈勇2100092701

文档撰写日期：2021年9月29日（星期三）

前言：交了第一次功课后注意到CF会统计所解决问题的数据，还记录你每天有没有解决问题，看这签到机制我的玩家魂就燃了。于是开始每日至少刷两题的习惯，一题CF，一题OJ。找那些简单同时自己感兴趣的题来做，和打电子游戏有相似的体验，又别有一番风味，特别有意思。然而OJ的题没提示难易度就踩坑了...还是CF做得多。这次功课只提前做了theatre，其他皆现做。以下排序是根据canvas上的，不是我实际完成的顺序。

1A. Theatre Square, math, 1000

<http://codeforces.com/problemset/problem/1/A>

解题思路：确认是数学计算题，从面积公式下手，原本以为可以直接向上取整 $(n*m) / (a**2)$ ，试错后才发现关键在于“覆盖”和“至少”这两回事。所以需要分开长和宽的考量，各别计算“覆盖”长和宽各需要“至少”几个。

1. 本质：接收三个数值输入，计算覆盖一个长方形面积至少需要几个正方形。
2. 条件：“接收”的数值需要拆开保存；“面积”需要做乘法；“覆盖”需要做除法；“至少”要把除数向上取整。
3. 关键词及符号：split()（“接收”）、*（处理“面积”）、/（处理“覆盖”）、import math 和 ceil（处理“至少”）

```
from math import ceil
n, m, a = (int(x) for x in input().split())
L = ceil(n/a) ; w = ceil(m/a)
print(L*w)
```

注：这题应用了ceil函数，是之前读RUNOOB没太注意的，所以多读了math库巩固一下记忆。

231A. Team, bruteforce/greedy, 800

<http://codeforces.com/problemset/problem/231/A>

解题思路：确认是判断题，照题意我可以直接把三个数加起来，再来判断大小。如此，需要逐个处理，要存列表。而且答案是累积的，还需要准备一个变量。

1. 本质：接收数据组数及若干组数据，每组数据若三个号码总和大于等于2，答案就多加1，输出最终答案。
2. 条件：“接收”的数值需要拆开及不同的保存方式，“逐个”处理，“判断”大小，还有一些“计算”。
3. 关键词及符号：map split（“接收”）、for in range（“逐个”）、if 和 >（“判断”）、sum 和 +（“计算”）

```
n = int(input())
ans = 0
for x in range(n):
    solve = map(int, input().split())
    if sum(solve)>1: ans+=1
print(ans)
```

注：反思后发现我其实用不上map，以字符串存列表后我用count就可以知道有几个‘1’了，会更简短些。

158A. Next Round, *special problem/implementation, 800

<http://codeforces.com/problemset/problem/158/A>

解题思路：针对前两个数字，n用途是遍历判断的次数，k是指示数位，我各别存变量。数据组则存列表，方便k的作用，这里注意要用 k-1 才合题意（列表从0开始）。同team题，需要一个变量存答案的累积值。重点是循环内部，我从前面看起，考虑0出现的特殊情况，需要判断的条件是该项 $\geq k$ 指示项 及 该项 > 0 ，这样碰到0自然就停了，也不必多此一举在 for 循环中设置 break 或者用太多次的 if（一开始想歪了搞得很复杂...）。

1. 本质：妥当提取数值及数据组，逐个判定数据中的数字，若符合条件，答案就多加1，输出最终答案。
2. 条件：“提取”输入需要拆开及不同的保存方式，“逐个”做“判断”，并给答案“加值”。
3. 关键词及符号：map split 和 for in split（“提取”）、for in range（“逐个”）、if $\geq >$ （“判断”）、 $+=$ （“加值”）

```
n, k = map(int, input().split())
seq = [int(x) for x in input().split()]
ans = 0
k_th = seq[k-1]
for i in range(n):
    i_th = seq[i]
    if i_th >= k_th and i_th > 0 :
        ans += 1
print(ans)
```

注：这题意外耗了很久，是这次耗时最久的一题。最初看到special problem想说会比较难？做题时接近半夜状态不太好，开始时脑回路已经当机，彻底忘记 range 的用处，一直 if 和 for 嵌套。还以为自己非常接近答案了，修补好几次都不能通过自己的调试。然后睡了一觉，今天一早猛地想起了 range，于是删掉重写，就这么过了。

教训：写程序脑子要清醒，不清醒不写，免得犯傻。

50A. Domino piling, greedy/math, 800

<http://codeforces.com/problemset/problem/50/A>

解题思路：确认是简单的数学计算题，是一个类似theatre的面积处理问题。因为domino的面积是恒定的(2*1)，又因为题目是限制了一个不可超出的范围，这题的答案需要向下取整。

1. 本质 / 条件：“接收”的数值需要拆开并转为整型，“计算”是数值相乘后除2，并向下取整，再输出答案。
2. 关键词及符号：map split（“接收”）、* 和 //（“计算”）

```
M, N = map(int, input().split())
ans = (M*N)//2
print(ans)
```

注：这里习得新技能：map，之前读RUNOOB又没注意到这个有多好用，在写这题时刚好看见其他同学的代码，因为好奇所以查了。map也可以被应用到后面很多题中（确实用了，就上边几题）。同时感觉这题难度低过西瓜...

03087. 邮箱验证, string

<http://cs101.openjudge.cn/practice/03087/>

解题思路：确认是字符串判断题，由于条件较多所以需要较多的命题来做判断，那么我可以把所有命题的布林值各别存好，过后合并判断即可。过程中发现多个条件写在一起的括号和and很容易出错，索性多整些变量，有几组是相互对应的，复制贴上修改关键部分即可，看着也容易明白，自己还算满意。

1. 本质：判断字符串是不是存在或不存在一些特定的字符。
2. 条件：1，不能超过1个'@'。2，首尾不能是'@'或'.'。3，'@'后至少有一个'.'且没有'@.'及'@'。
3. 关键词及符号：while 和 try except EOFError (特殊的输入方式&停止判定)、count (处理1)、[] != (处理2)、rfind < (处理3前半部)、not in (处理3后半部)、if else ("判断")

```
while True:  
    try:  
        mail=input()  
        r1 = mail.count('@') == 1  
        r2a = mail[0] != '@' and mail[-1] != '@'  
        r2b = mail[0] != '.' and mail[-1] != '.'  
        r3 = mail.rfind('@') < mail.rfind('.')  
        r4a = '@.' not in mail  
        r4b = '.@' not in mail  
        if r1 and r2a and r2b and r3 and r4a and r4b:  
            print("YES")  
        else : print("NO")  
    except EOFError: break
```

注：因为这题认识了EOFError，也第一次应用try语法，学到了新一种处理输入的方式，知道这个后其实并不难，就是条件有点多。后来还发现，其实这个不完全符合题意，我似乎应该把YES和NO先存个list，break了再print，可是这个是我真正AC的代码，所以我就交了这个。

选做：1221A. 2048 Game, brute force/greedy/math, 1000

<http://codeforces.com/problemset/problem/1221/A>

解题思路：确认是有点小复杂的数学题，一定需要进行计算。一开始想照着题面去写代码，敲了三行觉得不对劲。花了一些时间思考验证，发现可以把输入值中大过2048的去掉，剩余直接加总，这样省事多了。可以这么做是因为题目限定了输入值一定是2的次方，就跟一般的2048游戏一样。后来代码总报错去看了tutorial，确实如此。

1. 本质：先接收两个数字，后接收需拆分的数据，逐个判断其中2048或以下的数值加总能否达到2048。
2. 条件：“in&out”要先输入数据组数并一次输出，“计算”加总，“存储”答案。
3. 关键词及符号：=[] 和 for in 和 map split (“in&out”)、range (“逐个”)、if < >= else (“判断”)、sum (“计算”)、append (“存储”)

```

q = int(input())
ans=[]
for i in range(q):
    n = int(input())
    s = [x for x in map(int, input().split()) if x<2049]
    win = sum(s)>=2048
    if win: ans.append('YES')
    else: ans.append('NO')
for a in ans: print(a)

```

注：这次作业耗时第二久的题，就是不明白怎么test2数据总是过不去。特别感谢老师及同学们在微信群的帮助，在大家的指点下把关键问题解决了，开通了新一种思路。

选做：270A. Fancy Fencegeometry, implementation/math, 1100

<https://codeforces.com/problemset/problem/270/A>

解题思路：确认是数学计算题，把计算结果进行一个判断就行。正多边形的内角公式是知道的： $(n-2)*180/n$ ，这个公式就等于单个内角 'a' 嘛，而这题就可以利用n来判断，所以我需要把n算出来。那我不妨做个推导，以a表示n就行了啊，于是得到 $360/(180-a)$ 。其中n必是大过3的整数，那我刚好可以通过取模'%的方式处理，那就用简单的 if else 句即可解决。

1. 本质：通过计算逐个判断输入的角度是否是一个正多边形的内角。
2. 条件：“in&out”要先输入数据组数并一次输出，“计算”取模，“存储”答案。
3. 关键词及符号：=[] 和 for in (“in&out”)、range (“逐个”)、if == else (“判断”)、% (“计算”)、append (“存储”）

```

t = int(input())
ans=[]
for a in range(t):
    a = int(input())
    if 360%(180-a)==0:
        ans.append('YES')
    else: ans.append('NO')
for x in ans: print(x)

```

注：啊？这选做题意外简单？(下次就不一定了...)应该是数学思维帮了我，因为一个合适的公式变换简化了很多。

结语：如前言所提，这些天刷了好多题，稳定投入了蛮多时间，加上这次作业的经验，都让我更深入地了解编程的世界。从几乎0基础，到乐在其中，到尝试去帮助班上的同学们，我享受着这成长的过程。做一题学会一个函数，学会一种输入方式，学会一套解题套路。这就像打游戏一样，通关升级，解锁技能，攻击防御敏捷各方面，技能树不断水平拓宽，垂直延伸。一点一滴建立了老师所说的“思维”。最终我还是写了一堆文字，主要是因为我决定把每份 assignment 同时当作是个人笔记了，所以做得详尽，方便之后参阅复习，这也是很有满足感的一件事情。

最后祝老师国庆快乐！

Assignment#4：陈勇2100092701

文档撰写日期：2021年10月12~13日（星期二~三）

前言：在坚持每日刷题的努力下，这次作业皆提前完成（包括老师提前剧透的选做）。虽说都做过了，但我尝试把之前AC的代码做改进，对照题解改良自己的代码其实可以学到很多。我发现代码不一定越短就越好，有时候写得长一些反而运行得更顺利或省时，更重要的是能保证可读性。因此，本次提交的某些题有两版代码（都AC过），对比下显出优缺点，也记录着一个编程小白的成长，说不定也能帮到其他同学，也可以给老师您看个笑话(bushi)。

112A. Petya and Strings, implementation/strings, 800

<http://codeforces.com/problemset/problem/112/A>

解题思路：按字典排序比较可以直接通过字符串的大小比较实现，先统一大小写再分情况输出即可。

- 本质/条件：接收两个输入，把大小写“统一”，“判断”大小，按三种情况分别输出答案。
- 关键词及符号：lower（“统一”）、if elif else < >（“判断”）

```
1 s1=input().lower()
2 s2=input().lower()
3 if s1>s2: print(1)
4 elif s1<s2: print(-1)
5 else: print(0)
```

旧版：以下是很早之前自学摸索时写的一个版本，是第一次AC，有很多奇怪的操作，四个字形容：不堪入目。

```
1 lst = []          #old_code
2 for n in range(2):
3     lst.append(input())
4 s1=str.lower(lst[0]) ; s2=str.lower(lst[1])
5 if s1 == s2:
6     print(0)
7 else:
8     for i in range(len(s1)) :
9         if s1 > s2 :
10             print(1)
11             break
12         else :
13             print(-1)
14             break
```

注：看了题解 short code，这么容易的一题能解得那么妙，那数学思维和计算机思维融会贯通真的服了。另外也注意到可以把变量赋值为函数的操作，相信以后会用得上。

118A. String Task, implementation/strings, 1000

<http://codeforces.com/problemset/problem/118/A>

解题思路：题目已经把要求的操作列明，需要考虑的是怎么实现而已。若要精简快速一些，可以在接收输入时就完成变小写及过滤元音的步骤。考虑到 join 不能让字符串开头也有 ".", 不妨用列表存储时加上 ".", 最后再 join。

1. 本质：把输入字符串变小写，去掉元音字母，其余字符前皆加上一个 ".", 连着输出单个字符串。
2. 条件：“小写”的数值需要拆开及不同的保存方式，“逐个”字符检查，“判断”是否可取，“编辑”后“输出”。
3. 关键词及符号：lower (“小写”)、for in (“逐个”)、if not in (“判断”)、append + (“编辑”) =[] join (“输出”)

```
1 string = input().lower()
2 vowels = ['a', 'o', 'y', 'e', 'u', 'i']
3 ans = []
4 for i in string:
5     if i not in vowels:
6         ans.append("." + i)
7 print("") .join(ans))
```

改良：input不必另存变量，aoyeui也不必存列表，直接以字符串方式判断即可。(省了两行代码和两个变量)

```
1 ans = [] #improved_code
2 for i in input().lower():
3     if i not in 'aoyeui':
4         ans.append("." + i)
5 print("") .join(ans))
```

注：题解中的 short code 也是同一思路，只是精简成了一行，这题做法想不到其他特别的思路了。

263A. Beautiful Matrix, implementation, 800

<http://codeforces.com/problemset/problem/263/A>

解题思路：矩阵可以被理解为二维列表，其中只有0和1，那我就可以用sum的方式找到1在哪一行。步数可以拆解为垂直和水平两个方向，不妨各别计算步数后加起来得到答案。而刚巧因为 beautiful 的1位于二维列表正中心，即[2][2]，那么用两个索引位置减去2再取绝对值就可以算出距离了。

1. 本质：用合适的数据结构接收一个矩阵，找到1的位置，计算并输出其与中心的距离(步数)。
2. 条件：“逐个接收”输入并存进二维列表，“判断”1在哪行，针对“位置”进行“计算”，最后输出答案。
3. 关键词及符号：for in range split (“逐个接收”)、if sum (“判断”)、index (“位置”)、abs + - (“计算”)

```
1 row=[]
2 for _ in range(5): row.append([int(x) for x in input().split()])
3 for r in row:
4     if sum(r): ans=abs(row.index(r)-2)+abs(r.index(1)-2)
5 print(ans)
```

另一个很笨的做法：把仅有的5种答案(0124)及其对应的位置配对，当时的我甚至连接收输入都做得很拙劣...

注意：以下代码辣眼睛，请不要认真查阅。

```
1 matrix = [] #noob_code
2 row = 0
3 while row<5:
4     matrix.append(input())
```

```

5     row += 1
6     for num in matrix:
7         if matrix[2][4]=="1": print(0);break
8         elif matrix[0][0]=="1" or matrix[0][8]=="1" or matrix[4][0]=="1" or matrix[4]
9             [8]=="1":
10            print(4);break
11        elif matrix[2][2]=="1" or matrix[2][6]=="1" or matrix[1][4]=="1" or matrix[3]
12            [4]=="1":
13            print(1);break
14        elif matrix[0][4]=="1" or matrix[1][2]=="1" or matrix[1][6]=="1" or matrix[2]
15            [0]=="1" or \
matrix[2][8]=="1" or matrix[3][2]=="1" or matrix[3][6]=="1" or matrix[4][4]=="1":
16            print(2);break
17        else: print(3);break

```

注：当我们想不到好办法的时候，就承认自己不够聪明，然后做笨蛋也知道怎么做的事，AC是唯一的最终目的。

282A. Bit++, implementation, 800

<http://codeforces.com/problemset/problem/282/A>

解题思路：找共性，无论X在头尾，字符串的中间一位肯定是 + 或 - 号，以此为判断机制即可。

1. 本质/条件：接收若干行输入，“逐个”进行“判断”，并做出相应的“计算”，输出最终答案。
2. 关键词及符号：for in range (“逐个”)、if == (“判断”)、+ - (“计算”)

```

1 n = int(input())
2 x = 0
3 for i in range(n):
4     i = input()
5     if i[1]=="+":
6         x += 1
7     elif i[1]=="-":
8         x -= 1
9 print(x)

```

改良：直接在range里input可以少存一个变量，再把循环里的语句简化排好，这样的代码整体上就精简多了。

```

1 x=0                      #improved_code
2 for i in range(int(input())):
3     if input()[1]=="+": x+=1
4     else: x-=1
5 print(x)

```

注：题解里的short code是看懂了，但个人觉得不至于为了代码短而大幅牺牲可读性。另外反思，用find函数较难处理返回值，index更麻烦了还会报异常。其实使用 in 判断应该是更简易的解法，直接索引差别也不大。

02808. 校门外的树, implementation

<http://cs101.openjudge.cn/practice/02808/>

解题思路：见有重叠的情况，就想到了集合。既然题目用数线上的整数表示树的位置，我应该可以构建两个集合：一个 tree 代表原本的数线，另一 cut 代表要砍树的部分。tree 以第一行输入的数字 L(+1) 为 range，cut 以第二行输入的两个数字—一起点 start & 终点 end(+1) 为 range，添加其中的每个整数就成了。最后把 tree 和 cut 做差，这时候 tree 所剩余的元素数量(也就是他的长度)即为剩余树的数量。

1. 本质/条件：利用“接收”的数据“构建”集合，将集合“做差”后的“长度”输出。
2. 关键词及符号：map split (“接收”)、set for in range add (“建构”)、difference_update (“做差”)、len (“长度”)

```
1 L, M = map(int, input().split())
2 tree = set() ; cut = set()
3 for i in range(L+1): tree.add(i)
4 for i in range(M):
5     start, end = map(int, input().split())
6     for i in range(start, end+1): cut.add(i)
7 tree.difference_update(cut)      # tree -= cut (better)
8 print(len(tree))
```

注：看起来较长是因为变量名和一些较长的语法，这样好处是可读性强，单看代码大概也能知道逻辑思路。做这题对集合的运用提升了，认识了difference_update函数(其实完全可以直接用集合的减法简易地实现)。题解里用的似乎是动态规划dp，我还不会，也遇到好些题目需要这个知识了，决定列为本周自学目标。

选做：25A. IQ test, brute force, 1300

<https://codeforces.com/problemset/problem/25/A>

解题思路：因为特别项只有一个，我要是记录到两个同性的就能分辨出特别项的奇偶性。判断奇偶性常用取模，只要配合适当的 if 语句就可以解决了。哦对了，第一行输入的数字n用不上。

1. 本质/条件：先接收一个数字，后接收需“拆分”的数据，“计算判断”特别项奇偶性，输出其所在“位置”(+1)。
2. 关键词及符号：for in split (“拆分”)、if == != else % (“计算判断”)、index + (“位置”)

```
1 n=input() ; odd=0 ; even=0
2 num = [int(x) for x in input().split()]
3 for x in num:
4     if odd==2 or even==2: break
5     elif x%2!=0: odd+=1
6     else: even+=1
7 if odd==2:
8     for x in num:
9         if x%2==0: print(num.index(x)+1)
10 else:
11     for x in num:
12         if x%2!=0: print(num.index(x)+1)
```

注1：原本想到可以借sum(num)判断，但很快发现bug：若sum是奇数则无法判断，这里再次因为学编程把数学基础巩固了一遍(笑)。结果还是得存两个计数的变量，本来觉得这个思路的简洁度还是有局限的，但AC后我又看了题解，发现了全新的操作：在接受数据时就取模。那可省事多了，因为取模后只有两种情况：所取的模加总为1则特别项是奇数，否则特别项是偶数。于是我马上写出了精简版：

```

1 | input()                                     #improved_code
2 | mod=[int(x)%2 for x in input().split()]
3 | if sum(mod)==1: print(mod.index(1)+1)
4 | else: print(mod.index(0)+1)

```

注2：题解里index套sum的操作实在太高级，看微信群的讨论大概理解，但我想我的功力还不支持我使这招。

选做：1374B. "Multiply by 2, divide by 6", math, 900

<https://codeforces.com/problemset/problem/1374/B>

解题思路：看到题目就想逆向思考，若一个数可由6的倍数除若干次2得到，则符合题意。那就是说，这个数必可拆解成 $3^x 2^y$ ，其中 $x \geq y$ 。由此我又想到要用取模的方式处理了，套循环不断除3，直到无法被除尽为止，把x和y当作累积计数用的变量，若最终得到的是1，且 $x \geq y$ ，就有答案。答案至少满足3的次幂(x)，再补上3与2次幂的差(x-y)，即 $x+(x-y)$ ，输出答案 $(2x-y)$ 即可。

1. 本质：通过计算，逐个判断输入的数是否可由2和3质因分解，输出操作所需步数。
2. 条件：“in&out”先输入数据组数并一次输出，“逐个”通过“循环”去“计算”取模&加减乘除，最后做“判断”。
3. 关键词及符号：`=[] for in append ("in&out")`、`range ("逐个")`、`while ("循环")`、`% + - * / ("计算")`、`if >= == else ("判断")`

```

1 | t = int(input())
2 | ans = []
3 | for _ in range(t):
4 |     n = int(input())
5 |     x=0 ; y=0
6 |     while n%3==0: n/=3 ; x+=1
7 |     while n%2==0: n/=2 ; y+=1
8 |     if x>=y and n==1:
9 |         ans.append(2*x-y)
10 |    else: ans.append(-1)
11 | for a in ans: print(a)

```

注：看了直接按照题意实现的思路，好像比较快捷一点点？不过我最快想到能用的就是如此，对我个人来说看起来也比较易懂，面对一个还未解决的问题还是不多纠结，先有办法再想好办法。

结语：随着刷题量的累积，明显感觉越写越轻松了，也敢于挑战一些更难的题目。目前正在养成一个习惯：每次做完一题就去题解里看看，时常发现新大陆，再勤一点可以每题都在网上搜索看看有没有人公开解法或思路。这样做可以萃取新知识，也可以刷新自己的观念，避免自己被习惯性思维束缚住。可能自己的兴趣比较浓厚吧，学习特别带劲，也很愿意主动给自己定制任务，在这优良环境下我相信只要感兴趣总能学好的。目前小目标是自学dp，听说这是一道坎，我会慢慢地去研究他，过程中肯定还会接触到各种零碎的相关知识，一点点养成编程技能树。

定向学习，自由探索，皆有收获，岂不美哉！

Assignment#5：陈勇2100092701

文档撰写日期：2021年10月19日（星期二）

前言：这次作业除选做题外皆提前完成，自己多刷题确实让之后作业压力小了很多，能快速交上作业之余也能帮助到其他同学，感觉很不错。**230B确实是魔王题，我了解筛法，写代码，调试修改，前后耗了至少4小时，终于在下午3点上课前不久成功，那一刻太感动了。**之后也和上次一样审视了其他每题的代码，对照题解进行了优化。过程中几乎每次都会发现另类的解题思路或代码结构，也让我不断趋近于最好的代码。虽然这样肯定很烧时间，但我想也肯定是值得的，这就是我心目中“做作业”应该有的效果。同时本次作业在一些地方放了**黑体字**，这样“**可读性**”应该会提高一些。

339A. Helpful Maths, greedy/implementation/sortings/strings, 800

<http://codeforces.com/problemset/problem/339/A>

解题思路：把字符串接收后处理一下再输出，三个步骤搞定：拆解、排序、拼接，只要用对关键语句就可以了。

1. **本质/条件：**把输入值针对 '+' “拆解”，把得到的列表做增序“排序”，最后输出用 '+' “拼接”出的字符串。
2. **关键词及符号：** split (“拆解”)、 sort / sorted (“排序”)、 join (“拼接”)

```
1 | s = [int(x) for x in input().split('+')]
2 | s.sort() ; ss = [str(x) for x in s]
3 | print('+'.join(ss))
```

改良：其实可以直接用字符串比较，因为 '1' < '2' < '3'... 规律和整数比较一样，所以先 int() 后再 str() 的步骤完全可以省略，再用sorted函数替代sort方法，就可以把整个代码优化成一行：

```
1 | print('+'.join(sorted(input().split('+')))) #improved_code
```

注：这个缩成一行的可读性也还行（我能接受），运行时间和占用内存也有优化，所以我认为是 improved。

281A. Word Capitalization,implementation/strings, 800

<http://codeforces.com/problemset/problem/281/A>

解题思路：甭管首字符是否大写吧，一律给他变大了再输出就完事儿。（在python里没毛病，别的语言我不懂）

1. **本质/条件：**接收一个输入，确保首字符是“大写”字母后输出，需要“索引”及“拼接”。
2. **关键词及符号：** upper (“大写”)、 [:] (“索引”)、 + (“拼接”）

```
1 | x=input() ; print(x[0].upper()+x[1:])
```

注：写成一行看着舒服，只是可惜没法直接把 input 放在 print 里，不然就搞得太麻烦了。

266A. Stones on the Table, implementation, 800

<http://codeforces.com/problemset/problem/266/A>

解题思路：需要设置一些初始变量控制循环做判断，要进行逐项比较的话不应该用直接取字符串中的字符遍历，用range设置索引数字会灵活许多，通过循环不断累计答案，最后输出答案即可。

1. **本质/条件：**通过“循环”，对输入字符串做“判断”，据结果“计次”或“滑动”，最后输出答案。
2. **关键词及符号：**for in range len (“循环”)、if == else (“判断”)、+= (“计次”)、= (“滑动” - 更新赋值的操作)

```
1 | input() ; s=input() ; ans=0 ; a=s[0]
2 | for i in range(1,len(s)):
3 |     if s[i]==a: ans+=1
4 |     else: a=s[i]
5 | print(ans)
```

旧版：一开始直觉想法是把字符串拆成列表，然后逐项比较，相同的就去掉并计次，这个整体较劣质就不细说了...

```
1 | n = int(input())          #odd_code_not_so_good
2 | s = [x for x in input()]
3 | i=0 ; j=1 ; ans=0
4 | while j<len(s):
5 |     if s[i]==s[j]:
6 |         s.remove(s[j])
7 |         ans+=1
8 |     else: i+=1 ; j+=1
9 | print(ans)
```

注：这题中要用到字符串长度时，直接len好像比储存第一个输入'n'更好。前者只是调用一个len函数，后者用了一个int函数之余还要另存一个变量，所以不妨无视第一行输入以求更高效率(速度快&内存小)。这个想法在这里看起来没有太大的区别，不过到后期或许能帮上大忙，这是写程序的奥妙了。

96A. Football, implementation/strings, 900

<http://codeforces.com/problemset/problem/96/A>

解题思路：只有两个情况是YES：7个0和7个1存在于字符串中。这里我首先想到要用的是find方法，利用他找不到会返回-1的特性，我存为布林值以便判断。

1. **本质/条件：**对输入的字符串做“判断”，“查找”其中是否有7个0或1，依“条件”输出答案。
2. **关键词及符号：**if else (“判断”)、find (“查找”)、< and (“条件”)

```
1 | s = input()
2 | s0 = s.find('0000000')<0
3 | s1 = s.find('1111111')<0
4 | if s0 and s1: print("NO")
5 | else: print("YES")
```

改良：其实in有时候用起来特省事儿，而固定重复次数的字符可以用*来处理，这样代码就简化许多。

```
1 | s=input() ; print(['NO','YES'][('0'*7 in s)or('1'*7 in s)])      #improved_code
```

注：一行解决，跟题解的基本一样，可读性也还行，谨慎一点就在or前后两段话加了括号。

615A. Bulbs, implementation, 800

<http://codeforces.com/contest/615/problem/A>

解题思路：先注意到两个细节：一是灯泡凡是被开过就亮了所以会出现重叠，二是自第二行起每行输入的首个数字是无用且造成干扰的；前者提醒我们可以用集合处理，后者则需要我们加上排除手段。这样就和校门外的树类似，也是建构两个集合，只不过后面是判断两者的子集关系而已。

1. **本质/条件：**利用“接收”的数据“建构”集合，注意“排除”干扰，“判断”子集关系后输出相应答案。

2. **关键词及符号：**map split (“接收”)、set for in range add (“建构”)、remove (“排除”)、if (“判断”)

#以下用到了列表，以及一些比较奇怪的操作，都是当时糊涂，详见改良代码说明。

```
1 n,m = map(int,input().split())
2 bulbs = set() ; on = set()
3 for i in range(1,m+1): bulbs.add(i)
4 for _ in range(n):
5     raw = [int(x) for x in input().split()]
6     raw.remove(raw[0])
7     for x in raw: on.add(x)
8 if bulbs.issubset(on): print("YES")
9 else: print("NO")
```

改良：省略大量不必要的数据结构及方法/函数，并利用update方法取代 issubset+add 方法的功用，remove 方法则用列表索引代替了，print的条件也用列表索引及 len 的判断简化了。这边是当时的我跟上回校门外的树犯了一样的傻：已知集合长度，而它又是整数序列，所以我们只要判断小集合(被点亮的灯)所有的元素个数(长度)是否等于大集合(所有的灯)就行了，根本不必再建构集合，最终结果跟题解基本上一模一样。

真·关键词及符号：map split (“接收”)、set for in range update (“建构”)、[:] (“排除”)、[] len == (“判断”)

```
1 n,m = map(int,input().split())      #improved_code
2 on = set()
3 for _ in range(n):
4     on.update(input().split()[1:])
5 print(['NO','YES'][len(on)==m])
```

注：这说明一个好的思路是优秀代码的稳固地基，而好的算法和数据结构就是优秀代码的钢筋和水泥。

19944. 这一天星期几v0.3, math, cs10119 Final Exam

<https://codeforces.com/problemset/problem/25/A>

解题思路：我需要先存好答案，最终会计算得出的数字会成为索引。为了获取四个参数cymd，需要用索引对输入的日期切片。其余的基本上就是把题目所提供的转换成程序语言即可，这里不赘述。

1. **本质/条件：**设置“循环”接收输入，将接收的日期“切片”保存，“判断”特殊情况并进行处理，利用参数“计算”索引，最后利用索引“输出”对应的答案。

2. **关键词及符号：**for in range (“循环”)、[:] (“切片”)、if <== (“判断”)、+ - * // % (“计算”)、=[] append (“输出”)

```

1 day=['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
2 ans=[]
3 for i in range(int(input())):
4     dt=input()
5     c=int(dt[0:2]) ; y=int(dt[2:4]) ; m=int(dt[4:6]) ; d=int(dt[6:8])
6     if m<3 and y==0: m+=12 ; y=99 ; c-=1
7     if m>3: m+=12 ; y-=1
8     w = (y + y//4 + c//4 - 2*c + (26*(m+1))//10 + d - 1)%7
9     ans.append(day[w])
10    for a in ans: print(a)

```

注：这题在模拟考上因为看漏了“前一年”三个字，硬是让我崩溃了。这看起来几乎是送分的一题，直接影响我没法完成后面的装箱问题。这题会永远提醒我要仔细读题，尤其是关键字眼一定要通通捕捉到，否则哪怕再有能耐也必败于此。很庆幸这不该发生的失误是发生在模拟考，但愿下一次模拟考我会做得更好吧。

01017. 装箱问题, greedy

<http://cs101.openjudge.cn/practice/01017/>

解题思路：这是一个概念比较杂的数学题，步骤会很繁琐。网上查图解比较容易解释思路，以下文字叙述：

1. 需要接收输入中6个数字，分存变量比较合适。输入停止于一个特别的输入，while 循环加 break 可以处理。
2. 为了方便我称6乘6箱子为“6号箱”，以此类推。首先是计算大箱 (3456)，其中3号箱比较特别，要除4后向上取整，其余每个皆须开一个新包裹。
3. 接着处理2号箱：可以往4号箱里塞5个，而针对3号箱可以利用取模为索引去对应允许塞入的数量(提前设置列表)，接着需要判断：若能把2号箱塞完那就不必再开新包，否则把剩余的除4后向上取整就是需再开的新包数。(补充：9是由36/4得到的，这个操作是因为一个空包可以塞9个2号箱)
4. 最后处理1号箱：到最小单位可以用面积处理了，此时已开包裹面积与23456号箱所占据面积的差就是还能塞下1号箱的数量，同上面2号箱逻辑，只是把除数换成了36 (36/1)。
5. 可能这样的解释会比题解完整(冗长)一些，至少可以让我自己清晰记住整个解题思路。

关键词及符号： map split (输入)、while break (循环)、+ - * // (计算)、[] % (索引)、if == > (判断)

#以下没有先存储后一次输出答案，但OJ上可以过，是一个bug？一般还是要写 ans=[] 加上 append，不碍事。

```

1 extra=[0,5,3,1]
2 while 1:
3     a,b,c,d,e,f = map(int,input().split())
4     if a+b+c+d+e+f==0: break
5     box = f + e + d - (-c//4)
6     can_b = 5*d + extra[c%4]
7     if b>can_b: box += -((can_b-b)//9)
8     can_a = 36*box - (36*f + 25*e + 16*d + 9*c + 4*b)
9     if a>can_a: box += -((can_a-a)//36)
10    print(box)

```

注：当时被星期几的题雷到了，思绪不太好，一开始还浪费时间写了更冗长繁琐的代码，自然错误百出。后来把星期几那题搞定了，再看这题就比较有想法了，有几处还是参考了同学的代码(实际上就是题解的代码)，就把脑子里想的用比较合适的方式写成了代码。这题可说是目前看过最难的 greedy，做完颇有收获。

选做：230B. T-primes, binary search, implementation, math, number theory, 1300

<http://codeforces.com/problemset/problem/230/B>

解题思路：这是一个因为老师被学生感动而突然召唤的boss，刚出新手村不久的我苦战许久，最终成功击破：

1. 题目特别要求因数有且仅有3个，不难发现只有 **质数的平方** 符合题意。
2. 我想接收数据时就做**开方**处理，这样其中的**最大值** (我的变量叫做mx) 就是我需要创建的**质数表上限**，这样就不必每题都用筛法搞个 10^6 为上限的质数表了。
3. 需设置质数的**身份标记列表** (binary search)，长度就是 mx-1。
4. 重头戏，我用**埃氏筛法** (number theory) 得到质数，再存进集合。其中我对range及索引做了特别处理，是为了配合前面的操作，这应该算是一种优化了。(不细说，尽在代码中)
5. 最后，将原始列表**遍历对照**质数集合，输出**对应的**答案。对了，第一个输入值在我的代码里无用处。

关键词及符号： int ** split (开方)、 max (最大)、 [] * (上限)、 for in range if add [] (筛法)、 [] in (对应)

```
1 | input()
2 | num = [int(x)**(0.5) for x in input().split()]
3 | mx = int(max(num))
4 | prime = set() ; isprime = [1] * (mx-1)
5 | for i in range(2,mx+1):
6 |     if isprime[i-2]:
7 |         prime.add(i)
8 |         for j in range(i**2,mx+1,i): isprime[j-2]=0
9 | for x in num: print(['NO','YES'][x in prime])
10| # python3 1622ms , pypy3-64 716ms
```

注：由于还没有好好研究复杂度的概念，看来这个速度没有超过题解里的几套优秀代码，只不过终于见识到了pypy的提速效果，用的是CF上新开放的64-bit版本，在这里效果很是显著。还在想改进空间在哪里，之后有空再慢慢试吧。

结语：今天好刺激了，直接24小时内冲完功课(要是刚刚不去看高数视频早冲完了我真是要疯)。这种冲功课的感觉特别棒，虽然真的累，但成就感爆棚，一种一天内把难度开最大再刷完副本的感觉。功课量的提升也促使了我的提升，理解题意、编写代码、改良旧码、写作业文、做好排版，每件事的都获得了巨大提升。冲刺路上还有余力支援了群里正努力做作业的同学们，对于自己也有收获。该怎么说呢？爽。

~>>>today = 20211019 ; completed.time = 2357

~>>>print(['NO','YES'][completed in today] + '!')

~YES !

Assignment#6：陈勇2100092701

文档撰写日期：2021年10月26日（星期二）

前言：这次作业除选做题外皆提前完成（有空要多尝试选做题了），也花了比较大的功夫研究选做题。这期间自己通过各种管道去了解一些典型算法，包括贪心，二分查找，动态规划等。**多方资讯汇总**可以进行**对比分析与整理归纳**，感觉收获颇丰。虽然不一定懂了多少，**多接触这些思维也会比较灵活**，更重要的是**借此认识一些函数的用法**，甚至比直接去看各别函数的用法来得更有效果。这次照常尝试优化了每题早已完成的代码，已经养成铁打的习惯了，不试着改两下都不舒服，我相信坚持下去必会稳定进步。

236A. Boy or Girl,brute force/implementation/strings,800

<https://codeforces.com/problemset/problem/236/A>

解题思路：题目挺有意思，抽出重点：**判断字母种类数量的奇偶性**。这里要去掉重复元素，可以用set，数量用len可以得到，奇偶性的判断用过好多次了：就是取模%。

1. **本质/条件：**先“**提取**”输入值中字母种类的“**数量**”，后“**判断**”奇偶性，输出相应答案。
2. **关键词及符号：**set（“**提取**”）、len（“**数量**”）、if % == else（“**判断**”）

```
1 if len(set(input()))%2==0:  
2     print("CHAT WITH HER!")  
3 else: print("IGNORE HIM!")
```

缩短式代码：这是一个二元简单判断，且直接对应输出，满足这两个条件就可以搬出列表索引来缩短一切了。

```
1 print(["CHAT WITH HER!", "IGNORE HIM!"] [len(set(input()))%2])
```

注：x%2的值只能是1或0两种可能，所以我们经常可以利用这一点做一些取巧的手法，**当然前提是保证可读性和效率**。通常if else式语句比列表索引式跑得快，不难理解：前者只跑一部分，后者多跑了一个列表。

69A. Young Physicist, implementation/math, 1000

<https://codeforces.com/problemset/problem/69/A>

解题思路：要确定合力为0，既然输入都贴心地把力分解好了，不妨各别计算xyz三个方向，sum都是0就对了。

1. **本质/条件：**“**接收**”输入的数字，各别“**存储**”，通过求和“**判断**”是否都为0，输出相应答案。
2. **关键词及符号：**map int split（“**接收**”）、=[] append（“**存储**”）、if sum == else（“**判断**”）

```
1 n = int(input())  
2 Fx=[] ; Fy=[] ; Fz=[]  
3 for i in range(n):  
4     x,y,z = map(int,input().split())  
5     Fx.append(x) ; Fy.append(y) ; Fz.append(z)  
6 if sum(Fx)==0 and sum(Fy)==0 and sum(Fz)==0:  
7     print("YES")  
8 else: print("NO")
```

改良：把测试数据组数的输入 n 直接放 range 里就不用存了。用 **any** 函数判断任一 sum 有值就 NO，这样把 if 判断的条件简化，整体就可以跑得快一些。（最后其实也可以写成列表索引缩短式，考虑到可读性还是算了）

```
1 Fx=[] ; Fy=[] ; Fz=[] #improved_code
2 for _ in range(int(input())):
3     x,y,z = map(int,input().split())
4     Fx.append(x) ; Fy.append(y) ; Fz.append(z)
5 if any((sum(Fx),sum(Fy),sum(Fz))):
6     print("NO")
7 else: print("YES")
```

122A. Lucky Division, brute force/number theory, 1000

<https://codeforces.com/problemset/problem/122/A>

解题思路：这里注意题意中的 "almost lucky"，因为限制范围不大，不妨直接列出来逐个遍历。可以用集合列出数据，省时一些。另外注意到 444、777、744 三者是 4 或 7 的倍数，所以判断时可以省略。

1. **本质/条件：**对输入的一个数做“判断”，其是否能被“**特定的数中任一项**”整除，据结果输出相应答案。
2. **关键词及符号：**if % == else (“**判断**”)、for in {} break (“**特定的数中任一项**”）

```
1 n = int(input())
2 for i in{4,7,47,74,447,474,477,747,774}:
3     if n%i==0: print('YES') ; break
4 else: print('NO')
```

缩短式代码：用 **all** 函数，在里面就可以遍历判断并形成列表，于是缩得很短，但不见得是更优秀的代码。而且经过测试，要是把 input 也嵌入 all 里，会出bug，说明这个写法也不太安全，要不然就是我还无法驾驭吧。

```
1 n = int(input())
2 print(['YES', 'NO'][all(n%i for i in{4,7,47,74,447,474,477,747,774})])
```

19949 : 提取实体v0.2, cs10119 Final Exam, string

<http://cs101.openjudge.cn/practice/19949/>

解题思路：只要计算 "###" 的个数并取其整除2的值，再减去 "### ###" 的个数，就可以得到“实体”的数量。

1. **本质/条件：**对输入字符中的特殊字符数量做“**计算**”，“**存储**”每行的结果，最后输出“**总和**”。
2. **关键词及符号：**count // - (“**计算**”)、= [] append (“**存储**”）、sum (“**总和**”）

```
1 ans = []
2 for _ in range(int(input())):
3     s = input()
4     real = s.count('###')//2
5     link = s.count('### ###')
6     ans.append(real-link)
7 print(sum(ans))
```

注：如果把 real 和 link 直接放进 append 里会更短，除了用 for 基本就和题解一样，但为了可读性还是别了。

选做：1000B. Light It Up, greedy, 1500

<https://codeforces.com/problemset/problem/1000/B>

解题思路：我把我想的概念用文字一层层说明，包括我起的变量名会在括号内或加黑表示，也算是趁我还理得清的时候要详细记下给未来的自己看。以下思路含有部分**逆向思维**，次序不完全对应每行代码的功用：

1. 先按题意接收数据，把 0 和 M 加入 program 的前后形成**列表(p)**。
2. 想想插入一个数字后会发生什么？该区间后所有时长的“亮暗性”将会**互换**。由此我构建一个公式：
最大总亮灯时长 (**mx**) = 某区间前的亮灯时长 + 某区间在插入后**新的亮灯时长** + 某区间后的暗灯时长
3. 三个要素中第二个很特殊，分析这个区间的性质：
 1. 若在亮灯中，插入位置前为亮，后为暗，则**靠后**插入以期望亮灯时长最大。
 2. 若在暗灯中，插入位置前为暗，后为亮，则**靠前**插入以期望亮灯时长最大。
 3. 而**尽可能靠近两边**意味着差距为 1，也就是该时长 -1。因此时长为 1 的不能再插入，要设法排除。
4. 如此，我只要**遍历每个可能的区间计算 mx**，再逐个比较，**更新赋值**，最终即可获得最大值并 print 出。
5. 那么，我需要两个列表**各别存储插入前亮暗灯的时长 (lit & off)**，通过 for 循环搭配 range 为索引，加上取模**判断奇偶性**即可实现。（**p** 中从第2项起，偶数项减前一项为亮灯时长，奇数项减前一项为暗灯时长）
6. 接着**初始化变量**，**mx** 默认值就是 sum(lit)，**区间前亮时长(sumlit)**由 0 起随所取区间往后移而**增加**，**区间后暗时长(sumoff)**则由 sum(off) 起随所取区间往后移而**减少**。
7. 我想遍历 **p** 中每个区间，同时从 **lit & off** 中定位特定的项。由此借 **p** 的索引 **k** 创造一个通用于 **lit & off** 的索引 **ki**。其中 **k** 用于判定奇偶项，**ki = k // 2**，当 **k=0,1**, **ki=0**，以此类推。
8. 当取奇数项时，需**执行前面 6. 提及对于 sumlit & sumoff 的处理**。后者较特殊，因为从 **k=2** 起前面才有暗灯的时间，所以需避免误算 **k=0** 的情况，且 **ki** 的值需要 -1 以与其对应。同时记得 **3.3. 后半的排除处理**。
9. 尾声，又因为 **3.3. 前半**，可以计算当时的**总亮灯时长(tp) = sumlit + sumoff -1**，最后一步就是前面的 **4.**。
10. 代码中附上逐行注释，我肯定自己绝不会忘了自己是怎么解这题的了。

关键词及符号：略 (原为错误编辑残留，细节参见逐行注释)

```
1 n,M = map(int,input().split()) #1 注: # 后的数字表示行数(取个位)
2 p = [0]+[int(x) for x in input().split()]+[M] #2 创建"program"列表，把 0 和 M 加在首尾
3 lit=[] ; off=[] ; n+=2 #3 创建亮灯和暗灯的时长列表，n等效于len(p)
4 for i in range(1,n): #4 防止越界访问，从1开始，跟前一项做差
5     if i%2: lit.append(p[i]-p[i-1]) #5 若偶数项(索引是奇数)，与前一项的差是*亮灯*
6     else: off.append(p[i]-p[i-1]) #6 若奇数项(索引是偶数)，与前一项的差是*暗灯*
7 mx = sum(lit) ; sumlit=0 ; sumoff = sum(off) #7 准备下个循环需要的变量(详见解题思路)
8 for k in range(n-1): #8 最后一项不处理，range里需要-1
9     ki = k//2 #9 创造一个通用于两个列表的索引
10    if k%2: #0 若偶数项(索引是奇数)
11        if off[ki]<2: continue #1 若时长为 1，不能插入，跳出循环试下一个
12    else: #2 若奇数项(索引是偶数)
13        sumlit += lit[ki] #3 加上前面亮灯的时长
14        if k>1: sumoff -= off[ki-1] #4 减去后面暗灯的时长(详见解题思路)
15        if lit[ki]<2: continue #5 若时长为 1，不能插入，跳出循环试下一个
16    tp = sumlit + sumoff -1 #6 暂存当前的总亮灯时长(前亮+后暗-1)
17    if tp>mx: mx = tp #7 比大小，更新答案的值，最后就是最大的
18 print(mx) #8 用时: python3 202ms , pypy3-64 155ms
```

注：我以为上周选做已经很难，然而这周把灯泡开关开关的都快傻了... 在参考了各方面的提示和题解后没明白多少，我就使劲想，拿稿纸画，带数据算，还是自己搞明白了。下一道坎是把想法变成代码实现，这里更耗时，途中利用自己创造的测试数据调试，答案对了。没完，逃不过tle，开始修改算法，几度想拿题解的简洁代码参考。但我知道思路不同就不该这样，又看到群里老师提及“**sum太慢了**”，着手改良，最后终于通关了，**那一刻还真是light up了**。感觉我使出的操作都很基础，但也能解决那么复杂的题，或许利用一些更高级的函数或者框架可以把我的思路实现得更好吧。代码虽长，也是白纸起稿凭空创造的，也很满意了。这是CF上AC的第85题。

结语：这次题少了些，难度全压在选做题上了，所以总结了上面这题大概也总结了这份作业。这周还体会到的是老师发群里的作业中藏着不少宝贵的经验，每次我都要好好参阅，指不定就大有启发。想到萧伯纳的那句名言：“你有一个苹果，我有一个苹果，彼此交换一下，我们仍然是各有一个苹果；但你有一种思想，我有一种思想，彼此交换，我们就都有了两种思想，甚至更多。”我觉得他说得很对，我想这也是我认真写好每份作业的一大理由吧。

```
~>>>completed_in_24h = 2
```

```
~>>>will_continue = True
```

Assignment#7：陈勇2100092701

文档撰写日期：2021年11月2日（星期二）

前言：先反省：**百密一疏！** 才发现上次选做题的“关键词及符号”没编辑，提交前检查得更严谨，**慢工出细活**。还要做的是：准备好 assignment 的 markdown 排版样本，然后每次取用就可以。我之前都是把前一份作业文件复制后编辑内容，这样每次删掉内容再填充其实挺费神的，果然也出了纰漏。这应该算是一个简单的**计算机思维**，我早该想到了，好处多多啊！也许**学会有效率地做 assignment** 也是这门课的收获之一吧。这次作业同样除选做题外皆提前完成，选做题研究了一阵子，两三次尝试后也AC了。

723A. The New Year: Meeting Friends, implementation / math / sorting, 800

<https://codeforces.com/problemset/problem/723/A>

解题思路：按题意，三个朋友应该在**最大点与最小点的中点**见面，总移动距离最小。设中点为 m，距离为d，则其关系符合等式： $d = \max - m + m - \min$ ，简化得 $d = \max - \min$ ，如此一来做法就变得很简单了。

1. **本质/条件：**先“接收”输入值，后“取极值”，输出两者的“差”。
2. **关键词及符号：**[int for in input split] (“接收”)、 max min (“取极值”)、 - (“差”)

```
1 | x = [int(i) for i in input().split()]
2 | print(max(x)-min(x))
```

注：我和题解的差别是对列表p的处理方式，题解用了多层函数 (list(map(int,...))), 这在题解里是很常见的操作。个人觉得可以用 list comprehension，少几层括号也比较容易检查阅读吧。（同 200B题）

705A. Hulk, implementation, 800

<https://codeforces.com/problemset/problem/705/A>

解题思路：随输入的数字变大而增长输出，输出内容是反复循环的，马上想到用 while & 计次变量，搭配列表存储答案，需要的话可以嵌入判断条件在必要时break。虽然这思路略显繁琐，但应该可以直接实现。

1. **本质/条件：**通过“循环”，累计“存储”答案，用判断句“控制”循环结束，最后输出“连接”的答案。
2. **关键词及符号：**while (“循环”)、= [] append (“存储”)、if else break (“控制”)、join (“连接”)

```
1 | n = int(input())
2 | ans = []
3 | while True:
4 |     ans.append('I hate') ; n-=1
5 |     if n: ans.append('that')
6 |     else: ans.append('it'); break
7 |     ans.append('I love') ; n-=1
8 |     if n: ans.append('that')
9 |     else: ans.append('it'); break
10 |    print(' '.join(ans))
```

改良：上边的代码一整大坨，且多重复，看着就不太行啊，一定还有更好的答案。不妨直接利用**奇偶性**判断(用取模%)，因为下标从0开始，所以下标偶数(包括0)是hate，奇数是love。再特别处理开头("I")、中间的连接("that I")与结尾("it")。功能上，for代替while，range代替break，这样代码就精简许多。

```
1 ans=[]
2 for i in range(int(input())):
3     ans.append(['hate','love'][i%2])
4 print('I ' + ' that I '.join(ans) + ' it')
```

一行式代码：由上述改良，利用python的list comprehension，把for直接嵌入列表，再把列表嵌入print中的join，如此即可把代码超级压缩到一行，效率/复杂度基本一样，可读性很差，但很省行数。(近乎走火入魔bushi)

```
1 print('I ' + ' that I '.join(['hate','love'][i%2] for i in range(int(input())))) + ' it')
```

注：看似简单且机械的一题，真的要做到极简与精致也不那么容易，还是要多想几遍，让思维慢慢拓展提升。

200B. Drinks, implementation/math, 800

<https://codeforces.com/problemset/problem/200/B>

解题思路：把每个体积分数加总后除以其个数就是答案。(题目Note的部分太明显，直接把x消掉即可看出)

1. **本质/条件：**把接收的数据都转为“**整型**”，“**计算**”并输出第二行的数据加总除去第一行数字的值。
2. **关键词及符号：**int (“**整型**”)、sum / (“**计算**”)

#注：见273A题的“注”。

```
1 n = int(input())
2 p = [int(i) for i in input().split()]
3 print(sum(p)/n)
```

492B. Vanya and Lanterns, binary search / implementation / math / sortings, 1200

<https://codeforces.com/problemset/problem/492/B>

解题思路：翻译 - 求最小需要每个点向左右延伸多大范围(d)才能覆盖整条线。先把每两个点之间的范围砍一半(取中点)，左右侧的灯笼就平分了照亮范围，这样d就最小。由此，遍历每两点间的范围，计算两点距离的一半(也就是d)，并保存每个d。最终输出**最大的 d**，**那就是每个点最小的需求**。补充用到排列的原因：**首尾两个灯笼不一定在街道首尾，所以可能有左右两端的“暗区”**，排列后直接计算这两灯笼与边缘的距离放进集合里就省事了。

1. **本质/条件：**接收并“**排列**”数据，“**遍历**”数据，“**计算并存储**”d的值，最终输出“**最大**”的d。
2. **关键词及符号：**sort (“**排列**”)、for in range (“**遍历**”)、set - / add (“**计算并存储**”)、max (“**最大**”)

```

1 n,L = map(int,input().split())
2 loc = [int(x) for x in input().split()]
3 loc.sort()
4 d_set = set((loc[0],L-loc[-1]))
5 for i in range(n-1):
6     d = (loc[i+1]-loc[i])/2
7     d_set.add(d)
8 print(max(d_set))

```

注：原本 `d_set` 是 `d_list` 的，列表的时间复杂度比较大，这里换成集合无伤大雅，且时间空间都省了，是更优的。这题难度看着挺大实际还行，**不太确定二分查找的元素体现在哪**，看了题解思路，对比后发现与上面代码没有明显的区别。我猜是在遍历的过程采取二分查找以节省时间？如果测试数据的量级再大些可能需要，但这题不怎么会超时，所以大可不必多写一大串代码，**杀鸡焉用牛刀嘛**。

选做：16528:充实的寒假生活, cs10117 Final Exam, greedy

<http://cs101.openjudge.cn/practice/16528/>

解题思路：典型greedy题，我用**活动结束时间判定**，结束得早也意味着开始得早，不妨用**二维列表**存储数据，针对结束时间进行排序。接着从左到右遍历列表，只要**后一个的开始时间比前一个结束时间大**就可以参加。注意每参加一个新活动时，结束时间就需要更新。最后累计的答案就是最多能参与的活动数。

1. **本质/条件：**特别排序“接收”的数据，“遍历”数据，通过判断“控制变量”，输出最终答案。
2. **关键词及符号：**for in range int reversed split sort (“接收”)、for in range (“遍历”)、if > + = [] (“控制变量”)

```

1 n = int(input()) ; event=[]
2 for _ in range(n):
3     event.append([int(x) for x in reversed(input().split())])
4 event.sort()           #event.sort(key = lambda x:x[1]) (replacing reversed())
5 end = event[0][0]
6 ans = 1
7 for i in range(1,n):
8     if event[i][1]>end:
9         ans += 1
10        end = event[i][0]
11 print(ans)

```

注：第3行 `reversed` 函数的功能 (以便针对每个小列表的第二项排序)，可以在第4行 `sort` 方法内用 `lambda` 函数代替 (如注释)。另外，已知这题可用 **dp思路** 实现，所以尝试挑战。原本按自己想法写得很怪，答案也不对。于是参考题解，写着写着基本上和题解没区别了，关键是都还没搞明白题解的思路来着。目前对于 dp 是似懂非懂，有的题一下转过来了，有的就还不能。之后有空再研究，开窍了能写出来再补交吧。

选做：545C. Woodcutters, dp/greedy, 1500

<https://codeforces.com/problemset/problem/545/C>

解题思路：先让首尾两棵树各别往左右倒，中间的统一先判断能否左倒，能就倒，否则判断能否右倒。过程中注意一旦前一棵树是右倒的，后一棵树左倒的空间就少了，需要特别处理这一点（我设了变量 **rsp** 代表之，就是占用的 right space）。通过循环累计所砍的树应该能行（先不管超时）。依他的输入模式，可以用二维列表存储数据，另外注意只有1棵和2棵树时必然全砍。

1. **本质/条件：**“接收”一个二维列表，“遍历”中间项做“判断”，“控制变量”以辅助判断并得到累计的答案。
2. **关键词及符号：** [list map int split for in range] (“接收”)、 for in range (“遍历”)、 if elif else < - [] (“判断”)、
+ = (“控制变量”）

```
1 n = int(input())
2 xh = [list(map(int,input().split())) for _ in range(n)]
3 cut=2 ; rsp=0
4 if n<3: cut=n                                # just one or two tree
5 for i in range(1,n-1):                         # index except first&last
6     h = xh[i][1]
7     if h < xh[i][0] - xh[i-1][0] - rsp:        # can fall left
8         cut += 1 ; rsp = 0                      # no right space occupied
9     elif h < xh[i+1][0] - xh[i][0]:           # can fall right
10        cut += 1 ; rsp = h                      # right space occupied 'h'
11    else: rsp=0                                # didn't cut a tree
12 print(cut)                                    # python3 343ms , pypy3-64 795ms
```

注：这个解法和题解里的第一套是一样的，只不过我处理往右倒的树占用空间的方式另设一变量，题解是直接在列表里改赋值。可能题解版本比较省内存吧，但我自己觉得这样比较方便控制，于我也容易理解。这题用pypy好像有超时的风险，python倒还行，相信在题目限制的范围内都无大碍。特别记录这题的 **两次WA(犯傻)**：

1. 第7行：- **rsp** 写成了 + **rsp**，写公式草稿时弄错的，好在这题没那么复杂，不然debug时不堪设想...
2. 第4行：针对 **1棵树** 和 **2棵树** 的特例忘了处理，一定要考虑到极端情况，题目总会隐藏一些陷阱，要是考试的时候在 OJ 上看不到错哪，那就很痛苦了...

结语：这周除了稳定每日刷题外，更多地去尝试了选做题（难题），尤其是形如 **greedy** 和 **dp** 一类的题特别常见。因此对各类型算法有了深一层的认识，这可能也是我觉得这次选做题比较能做得出来的原因。我自学时读到《算法基础与在线实践》第一章，章末小结有句话提醒了我：“**算法可以说是程序的灵魂。**”很幸运的是，在这门课里不断强调的“**数学思维和计算机思维**”和所谓的“**算法**”其实是同一个概念。再过些时日，再付些努力，再走向提升，再迈向卓越。

```
~>>>completed_in_24h = 3
~>>>solved_CF>100 ; solved_OJ>50
~>>>will_continue = True
```

Assignment#8：陈勇2100092701

文档撰写日期：2021年11月9日（星期二）

前言：这份文档有三部分：**一，必做&选做、二，月考六题、三(附件)**，上回作业“寒假”题的补充。这次比较长，月考六题的代码简单就不必细看了，附件是交过的但老师好像漏了没改到，为了方便批改在此附上，老师辛苦了。

12560: 生存游戏, matrix, cs10116 final exam

<http://cs101.openjudge.cn/practice/12560/>

解题思路：特别注意要**全都同步更新**，按题意写函数，需要的**参数**应该包括**原本的状态**(会以**二维数组**方式接受)以及他的**坐标(x&y)**，严格按照**四条规则**执行就行。而这个函数不能只跑一次，要跑多次得出更新后每个坐标的值，过程中先把结果**存进一个新的二维数组**，最后输出就可以了。(保护圈太方便，和二维数组根本绝配)

- 本质/条件：**写一个“**函数**”，通过“**计算**”邻居数，“**判定**”细胞生死，利用“**二维数组**”及保护圈，“**遍历**”每个坐标，最后“**输出**”一个新二维数组。
- 关键词及符号：**def return (“**函数**”)、sum [:] (“**计算**”)、if else <== (“**判定**”)、[] for in range * (“**二维数组**”)、for in range [] (“**遍历**”)、for in * (“**输出**”)

```
1 def game(old,y,x):
2     left = x-1 ; right = x+2 ; center = old[y][x]
3     n = sum(old[y-1][left:right] + old[y][left:right:2] + old[y+1][left:right])
4     if center: c = [0,1][1<n<4]
5     else: c = [0,1][n==3]
6     return c
7 n,m = map(int,input().split())
8 old = [[0]*(m+2),*[0,*map(int,input().split(),0)] for _ in range(n)], [0]*(m+2)]
9 new = [[0]*m for _ in range(n)]
10 for y in range(1,n+1):
11     for x in range(1,m+1):
12         new[y-1][x-1] = game(old,y,x)
13 for row in new: print(*row)
```

注：这里的*号有时是用于**复制**([*]), 有时用于**解包**(*map [*]), 位置不同功能也不同了，挺奇妙的。

18182: 打怪兽, sorting/math

<http://cs101.openjudge.cn/practice/18182/>

解题思路：**字典肯定好用**，时间为键，伤害为值。同时能放俩技能咋办？以**列表形式存储**伤害值即可。Q神的最优解是：**按时间顺序，每次打出至多m个伤害最高的技能**。过程中**检查血量**，怪死了(b<1了)就输出该次时间。

- 本质/条件：**建立技能“**字典**”，对键做“**排序**”，“**遍历**”每个键，再对各时刻技能的伤害值排序，据m的限制“**切片**”能打出的招数，“**检查**”打完后怪的血量，存储套数据的答案最后一并输出。
- 关键词及符号：**{ } [] (“**字典**”)、sorted (“**排序**”)、for in range (“**遍历**”)、[:] (“**切片**”)、if < break (“**检查**”)

```

1  ans=[]
2  for _ in range(int(input())):
3      n,m,b = map(int,input().split())
4      skill={}
5      for _ in range(n):
6          ti,xi = map(int,input().split())
7          if ti in skill: skill[ti].append(xi)
8          else: skill[ti] = [xi]
9      time = sorted(skill)
10     for t in time:
11         pro = sorted(skill[t],reverse=True)
12         b -= sum(pro[:m])
13         if b<1: ans.append(str(t)) ; break
14     else: ans.append('alive')
15 print('\n'.join(ans))

```

注：我省略了不少对代码构思的解释，其实和题解里其中一套思路一样，只是这些处理让代码精简许多，也易读。

选做：466C. Number of Ways, binary search/ brute force / data structures / dp / two pointers, 1700

<https://codeforces.com/problemset/problem/466/C>

解题思路：先发现若 $\text{sum}(a)$ 不能被3整除就不可能分（答案为0），这些数据可以直接排除。其余的，我们需要把 a 平分为三部分，第一部分的和是 $\text{sum}(a)$ 的 **三分之一($n1_3$)**，第一和第二部分的和是 **三分之二($n2_3$)**，由此只要通过累加一定可以寻得 $n1_3$ ，每个这样的节点后 **有至少一个节点是满足 $n2_3$ 的，且可能有多个**。所以先找到一个 $n1_3$ ，再累计其后边满足 $n2_3$ 的点数，遍历找到每个 $n1_3$ 及其对应的 $n2_3$ ，就得到答案了。

1. **本质/条件：**接收数组，通过“计算”得到判定的依据，再累加“遍历”数组，“控制变量”以累计答案。
2. **关键词及符号：** $\text{sum} \% / *$ (“计算”)、 for in range (“遍历”)、 $+ \text{if} == +=$ (“控制变量”)

```

1  n = int(input())
2  a = [*map(int,input().split())]
3  s = sum(a) ; ans = 0
4  if s%3==0:
5      n1_3 = s/3
6      n2_3 = 2*n1_3
7      sum_i = match = 0
8      for i in range(n-1):
9          sum_i += a[i]
10         if sum_i == n2_3:
11             ans += match
12         if sum_i == n1_3:
13             match += 1
14 print(ans)                                     # python3 420ms , pypy 3-64bit 545ms

```

注：第一次 debug 是第8行 range 要 $n-1$ ，这是当 $\text{sum}(a)$ 为 0 时出的 bug，题解里解释得很好确实受教了。第二次是原本写的 for 里还有 for ，整太复杂了不知怎的还 WA，参考题解发现 **明明是同一个思路，不同的控制结构得出的代码品质是天差地别，其实两个 if 要放对次序就可以实现了**，醍醐灌顶，改完马上 AC 了。

选做：1443C. The Delivery Dilemma, binary search / greedy / sortings, 1400

<https://codeforces.com/problemset/problem/1443/C>

解题思路：送餐时间是 平行计算 的，取餐时间则 叠加计算，两者中 更大的决定 了能吃上饭的时间。可以从最大的送餐时间开始考虑，预设全都用送的。若把最大送餐时间降低了，取餐时间必定增加，这样遍历时 上限(送餐时间)不断拉低，下限(取餐时间)不断升高，直到两者最接近时，大的一项 就是所需的 最小时间。代码主要可以分成两大块，逻辑如下：

1. 每次接收 a 和 b 两个数组后，将两者 组装 成一个新数组 ab，顺便以送餐时间 (a) 的降序 “排序” 好。
2. 设变量 time 为所求时间，遍历 ab 时每次取出送餐时间 deli (delivery) 及取餐时间 pick，比较 原本的 time 加上 pick 是否超出 deli，若是，意味着上下限已经达最靠近处，两者中 取大 的为答案，终止循环。

关键词及符号：* zip [] (“组装”)、sorted reverse (“排序”)、for in (“遍历”)、if >= (“比较”)、max (“取大”)

```
1 ans=[]
2 for _ in range(int(input())):
3     input()
4     a = [*map(int,input().split())]
5     b = [*map(int,input().split())]
6     ab = sorted([*zip(a,b)], reverse=True)
7     time = 0
8     for z in ab:
9         deli = z[0] ; pick = z[1]
10        time += pick
11        if time >= deli:
12            time = max(deli,time-pick)
13            break
14        ans.append(str(time))
15 print('\n'.join(ans))                                # python3 1185ms , pypy 3-64bit TLE
```

加速版代码：觉得pypy tle不太应该，看了加速方法，参考微信里同学总结的经验，得到了更快的速度。

```
1 import sys
2 ans=[]
3 for _ in range(int(sys.stdin.readline())):
4     sys.stdin.readline()
5     a = [*map(int,sys.stdin.readline().split())]
6     b = [*map(int,sys.stdin.readline().split())]
7     ab = sorted([*zip(a,b)], reverse=True)
8     time = 0
9     for z in ab:
10        deli = z[0] ; pick = z[1]
11        time += pick
12        if time >= deli:
13            time = max(deli,time-pick) ; break
14        ans.append(str(time))
15 print('\n'.join(ans))                                # python3 779ms , pypy 3-64bit 764ms
```

注：原本想到要用的是 enumerate 函数，写出来又复杂了，才发现更好用的 zip 函数，就省事很多。

2021/11/4 (四) 月考六题

注：这些题都很基础，题意也易懂，就不细写思路了（约瑟夫问题除外），我把代码交上来请老师过目。

02701. 与7无关的数, math

<http://cs101.openjudge.cn/practice/02701/>

```
1 ans=0
2 for i in range(1,int(input())+1):
3     if ('7' not in str(i)) and (i%7!=0):
4         ans+=i**2
5 print(ans)
```

02689. 大小写字母互换, string

<http://cs101.openjudge.cn/practice/02689/>

```
1 for a in input():
2     if a.isupper():
3         print(a.lower(),end="")
4     elif a.islower():
5         print(a.upper(),end="")
6     else: print(a,end="")
```

02712. 细菌繁殖, math

<http://cs101.openjudge.cn/practice/02712/>

```
1 ans=[]
2 mds=[31,28,31,30,31,30,31,31,30,31,30,31]
3 for _ in range(int(input())):
4     m0,d0,x,m,d = map(int,input().split())
5     if m0==m: k = d-d0
6     else: k = mds[m0-1]-d0 + sum(mds[m0:m-1]) + d
7     ans.append(str(x*(2**k)))
8 print('\n'.join(ans))
```

18223. 24点, implementation/math

<http://cs101.openjudge.cn/practice/18223/>

做法一：直接列出每个可能计算，复制贴上后修改也还可以，反正有算到24就对了。

```
1 ans=[]                                     # just lay out every permutation
2 for _ in range(int(input())):
3     a,b,c,d = map(int,input().split())
4     p = a+b+c+d ; h = -a+b+c+d
5     q = a-b+c+d ; i = -a-b+c+d
6     r = a+b-c+d ; j = -a+b-c+d
7     s = a+b+c-d ; k = -a+b+c-d
8     t = a+b-c-d ; l = -a+b-c-d
9     x = a-b+c-d ; m = -a-b+c-d
10    y = a-b-c+d ; n = -a-b-c+d
11    z = a-b-c-d
12    if 24 in (h,i,j,k,l,m,n,p,q,r,s,t,x,y,z): ans.append('YES')
13    else: ans.append('NO')
14 for a in ans: print(a)
```

做法二：用 for 循环搭配 if 可以绕开一些不必要的计算，但还是显得繁琐。

```
1 ans=[]                                     # be smarter with the for_s
2 for _ in range(int(input())):
3     a,b,c,d = map(int,input().split())
4     can = 0
5     for w in(a,-a):
6         if can: break
7         for x in(b,-b):
8             if can: break
9             for y in(c,-c):
10                if can: break
11                for z in(d,-d):
12                    if w+x+y+z==24: can=1
13                    elif can: break
14                if can: ans.append('YES')
15            else: ans.append('NO')
16 for a in ans: print(a)
```

注：还可以 import itertools 的 permutation 处理。虽是穷举题，仍可通过剪枝与控制结构提高效率，缩短代码。

02943. 小白鼠排队, dict

<http://cs101.openjudge.cn/practice/02943/>

```
1 queue={} ; weight=[]
2 for _ in range(int(input())):
3     w,h = input().split()
4     w = int(w)
5     queue[w] = h
6     weight.append(w)
7 weight.sort(reverse=True)
8 for i in weight: print(queue[i])
```

02746. 约瑟夫问题, implementation

<http://cs101.openjudge.cn/practice/02746/>

解题思路：预先了解过这个经典问题的逻辑才做的，我应用了**递归计算**的函数。每次有一只猴子退出后，函数可以实现对圈内其他猴子的下标进行调整，使其形成新的一圈，并从上一轮退出猴子的下一位开始计为1号，然后一直循环到剩下一只猴子($n=1$)。整个过程在后台处理，最终函数的返回值将会符合原始圆圈的下标值。

```
1 ans=[]
2 def josephus(n,m):
3     if n==1: return 0
4     return (josephus(n-1,m)+m)%n
5 while 1:
6     n,m = map(int,input().split())
7     if n==0: break
8     ans.append(josephus(n,m)+1)
9 for a in ans: print(a)
```

注：这题明白是明白了，但不知怎么解释得清楚，属于**代码看着一目了然却难以用言语描述**的那种。不过我确实能掌握这个函数了，通过一些调整可以解出相似的题目，比如 OJ 的 03253：约瑟夫问题No.2。

总结：这次作业有好几个重要的新认知或新知识点，最后梳理列出：

1. **二维保护圈：**当题目需要**对一个二维数组中某些坐标做判定**时，很常需要**依据该坐标周遭的坐标**，凡是遇到此类情况，都应该加上保护圈，方法是类似的，写好存个漂亮的通用副本，说不定考试就用了。
2. ****号解包：**不仅是乘法或复制，通过**解包**可以简化代码，避免多重的函数包裹而拖慢运行速度(亲测有效)。这尤其在需要**对数组或映射对象进行多次拆解拼装**的步骤里好用。
3. **灵活的切片：**通过简单的索引*[i:j]*，可以**调用外部的变量作为指针**，实现灵活变动的限制范围。
4. **拓展的字典：**字典的键和值可以为各种数据类型，利用这一点，可以实现**一对多的映射**。
5. **zip函数：**可以直接连接两个或多个数组，将其用'*'解包会得到多个元组，可以存入列表当作二维数组处理，缺点只是每项都不能编辑。相对于**enumerate**函数，多数情况下**zip**的性能更好。

```
~>>>completed_in_24h = 4
~>>>solved_CF>120 ; solved_OJ>60
~>>>will_continue = True
```

附件：Assignment#7_holiday+

文档撰写日期：2021年11月8日（星期一）

前言：这份文档只针对充实寒假题，之前只写出了greedy，我对自己不满意，现在学会了dp，把关于这题的部分做了些整理补充再交一遍（主要新内容在下一页）。今天考完化学测验，赶紧把这块补上。期中比较忙，只好跟着sorted(任务优先度)逐个完成，希望之后都跟得上。

选做：16528:充实的寒假生活, cs10117 Final Exam, greedy

<http://cs101.openjudge.cn/practice/16528/>

greedy思路：

我用活动结束时间判定，结束得早也意味着开始得早，不妨用二维列表存储数据，针对结束时间进行排序。接着从左到右遍历列表，只要后一个的开始时间比前一个结束时间大就可以参加。注意每参加一个新活动时，结束时间就需要更新。最后累计的答案就是最多能参与的活动数。

1. **本质/条件：**特别排序“接收”的数据，“遍历”数据，通过判断“控制变量”，输出最终答案。
2. **关键词及符号：**for in range int reversed split sort (“接收”)、for in range (“遍历”)、if > += [] (“控制变量”）

```
1 n = int(input()) ; event=[]
2 for _ in range(n):
3     event.append([int(x) for x in reversed(input().split())])
4 event.sort()           #event.sort(key = lambda x:x[1]) (replacing reversed())
5 end = event[0][0]
6 ans = 1
7 for i in range(1,n):
8     if event[i][1]>end:
9         ans += 1
10    end = event[i][0]
11 print(ans)
```

注：第3行 reversed 函数的功能（以便针对每个小列表的第二项排序），可以在第4行 sort 方法内用 lambda 函数代替（如注释）。补充：老师在微信群为同学们答疑时捡到新知识——**sort(key=lambda x: (x[1],x[0]))** 可以在二维数组实现**先以第二项为主排列，再考虑第一项排列**的操作。之后可以举一反三处理类似的情况了。

dp思路：

借鉴了 2020fall-cs101 蔡清远同学 收录于题解中的代码，结合我自己的理解写的，以下列出每步思路：

1. 先记录每个活动，这里用一个有61项的列表 **event**，索引值=结束时间；索引项=开始时间。这0-60的索引值，也刚好方便之后遍历的动作。另外考虑到接收数据中 可能有 '0'，默认值取 '-1' 是妥当的替代。
2. 可能有几个活动同时结束却不同时间开始，我们希望：**每个结束时间对应的开始时间尽量晚(大)**，这样可以保证参加到**当天结束的活动中时长最短的**，所谓的“**预处理**”。(这步其实内涵一个 greedy)
3. 开一个 **dp** 列表，存储最多能参与的活动数，**dp** 和 **event** 的 索引值相互对应，0-60也代表这61天。
4. 利用索引值 '**d**' (day)，遍历 **dp** 并赋值。设两个辅助变量：**remain = dp[d-1]** 为直到前一天能参加的最多活动数；**taking = dp[event[d]-1] + 1** 为选择参加这一天结束的活动后，能参加的活动总数。
补充 4.: "event[d]" 是该活动的开始时间，"dp[event[d]-1]" 是保证不和该活动冲突下最多能参与的活动(不包括该活动)总数。另外，**d从0开始是可以的**，反正 dp[-1] 也是0，remain就没毛病。**也是必须的**，否则不能处理到 "0 0" 的活动。
5. 遍历每一天无非两个情况：一，**有活动，则决策是否参加**；二，**没活动，那肯定保留前一天的决策**。如此即可一路更新最优解直到第61天。完事后，**dp** 的最后一项就是答案。

关键词及符号：= []* (开数组)、for in range (接收&遍历)、max (每次取更优数据/解)、if > else (决策判定)
#注释说明解题思路 1. - 5. 对应代码中的哪几行。

```
1 event = [-1]*61                      # 1. line 1
2 for _ in range(int(input())):           # 2. line 2-4
3     start,end = map(int,input().split())
4     event[end] = max(event[end],start)
5 dp = [0]*61                           # 3. line 5
6 for d in range(61):                   # 4. line 6-8
7     remain = dp[d-1]
8     taking = dp[event[d]-1] + 1
9     if event[d]>-1:                   # 5. line 9-13
10        dp[d] = max(remain,taking)
11    else:
12        dp[d] = remain
13 print(dp[60])                         # credit: 2020fall-cs101 蔡清远
```

注：当时发现这位同学的代码符合我的思路就借鉴了。相比他的原代码 **我多存了两个变量**，以便于人类理解及程序调用，同时发现到**题解的版本有些部分是不必要的**(没有实际功用)，我就拿掉了，果然还是AC。**在这个班里我们可以把前人留下的成果加以消化改良，而得到一个更好的成果，这个文化太赞了。**

结语：写到最后觉得这题也太妙了吧，可以发现 **dp** 和 **greedy** 两种算法元素的融合，这应该是许多题目背后隐藏的秘密。想起胡适一句名言，改编一下：编程有如金字塔，要能广大要能高。习惯每日写代码，一题两做学更多。

~>>>Assignment7_nicely_done = True

Assignment#9：陈勇2100092701

文档撰写日期：2021年11月16日（星期二）

前言：有感受到这周作业的难度，神奇的是 cf 的四题 dp 我竟然都提前做过（所以我这次才能保持今天内交作业），那时候确实研究了好一阵子来着。现在回过头去看，我还是能对以前的程序做不少优化，说明我持续在进步（暗自窃喜）。另外，二维数组算是比较掌握了，加上自己探索时刷的题，大多题型都想明白一遍了，希望期末时靠谱。

19943:图的拉普拉斯矩阵(matrix)

<http://cs101.openjudge.cn/practice/19943/>

解题思路：发现矩阵 D 和 A 并无关联也互不影响，不妨 **直接以二维数组形式创建矩阵 L**，其中每位初值为0，之后对每个位置做相应的加减运算即可。根据接收的数据，其实只需要做 **两件事**，最后就能得到 L 了：

1. 数据中数字 x 每出现一次，令 $L[x][x] += 1$ 。
2. 对每组数字 x y，令 $L[x][y]$ 及 $L[y][x] -= 1$ 。

关键词及符号： [] * for in range (**创建二维数组**)、 [] + - = (**两件事**)、 for in * (**输出**)

```
1 n,m = map(int,input().split())
2 mx = [[0]*n for _ in range(n)]
3 for _ in range(m):
4     a,b = map(int,input().split())
5     mx[a][a]+=1
6     mx[b][b]+=1
7     mx[a][b]-=1
8     mx[b][a]-=1
9 for row in mx: print(*row)
```

19942:二维矩阵上的卷积运算v0.2 (matrix)

<http://cs101.openjudge.cn/practice/19942/>

解题思路：充分理解题面后翻译成程序语言即可。接收两个二维数组，再准备一个每位为0的二维数组，之后通过计算增加赋值使其成为答案。难点在于怎么像题面那样去“滑动”，可以利用 **双层 for 循环控制索引值** 来实现，实际上因为 **二维矩阵相当于“二维的二维数组”**，应该需要 **四层 for 循环**，想清楚各个索引值对应的维度就可以了。

关键词及符号： [] * map for in range (**接收/创建**)、 for in range [] (“**控制索引**”)、 += * (**计算**)、 for in * (**输出**)

```

1 m,n,p,q = map(int,input().split())
2 mat = [[*map(int,input().split())]for _ in range(m)]
3 con = [[*map(int,input().split())]for _ in range(p)]
4 ans = [[0]*(n+1-q) for _ in range(m+1-p)]
5 for i in range(m+1-p):
6     for j in range(n+1-q):
7         for x in range(p):
8             for y in range(q):
9                 ans[i][j] += mat[i+x][j+y]*con[x][y]
10 for row in ans: print(*row)

```

注：上两题二维数组最后输出部分都用了'*'解包号，如此输出一个二维数组(矩阵)还挺方便的。

368B. Sereja and Suffixes, data structures/dp, 1100

<https://codeforces.com/problemset/problem/368/B>

解题思路：先把题面翻译成人话：从左到右，计算自该位置起(含)，往右去有几个独特数字(distinct numbers)？

1. 因为这是**往后取**的计算，需要把数据**逆向处理**。
2. 从后往前遍历，逐个做**判定**，把每个独特数字记录在一个集合，则遍历每位时，**当时该集合元素个数就是独特数字的个数**。
3. 再把每个“个数”存进一个列表，存完再把它**转逆序**，它就象征了**原始数组在每个索引值时，往右去的独特数字个数**。最后**把接收的数据当作索引**找回去就行了。

关键词及符号：reverse(逆序)、for in range(遍历)、= [] append set add len(存/取数据)、if not in(判定)

```

1 n,m = map(int,input().split())
2 a = input().split()
3 a.reverse()
4 cnt=[] ; dp=set() ; ans=[]
5 for i in range(n):
6     if a[i] not in dp: dp.add(a[i])
7     cnt.append(len(dp))
8 cnt.reverse()
9 for _ in range(m):
10    ans.append(cnt[int(input())-1])
11 for x in ans: print(x)           # python3 373ms , pypy3-64 810ms

```

改良：我优化了4部分：一，变量命名更有意义；二，先开好列表而后对每项赋值；三，用负数索引取代转逆序；四，先把答案包装好后一次输出。一是便于理解，二三四都是让程序整体更有效率且更简洁。

```

1 n,m = map(int,input().split())
2 array = [*reversed(input().split())]
3 distinct = set() ; n_dstt = [0]*n
4 for i in range(n):
5     a = array[i]
6     if a not in distinct: distinct.add(a)
7     n_dstt[-i-1] = len(distinct)
8 print('\n'.join([str(n_dstt[int(input())-1]) for _ in range(m)]))
9 # improved_code : python3 249ms , pypy3-64 733ms

```

706B. Interesting drink, binary search / dp / implementation, 1100

<https://codeforces.com/problemset/problem/706/B>

解题思路之 dp 法：先做一些 **前置作业**，接收每个价格，取其最大值作为开空间的上限，开一个 **buy** 作为动态列表存储下一步计算的数据。两个循环：第一次记录哪些节点可以多买一瓶，第二次从左到右每一项加上前一项的值。如此得到的 **buy** 中每项的 索引象征硬币数，值象征能买几瓶。再将后边输入的 每日硬币数作为索引 就能取到答案了。注意一个 特例：若硬币数超出最贵价格，答案直接取最大瓶数，不然数组访问越界就 RE 了。

关键词及符号：[] * map max (**前置作业**)、for in range [] += (**两个循环**)、if < (**特例**)、join map str (**输出**)

```
1 n = int(input())
2 price = [*map(int, input().split())]
3 x = max(price)
4 buy = [0] * (x+1)
5 for i in price:
6     buy[i] += 1
7 for j in range(2, x+1):
8     buy[j] += buy[j-1]
9 day = int(input())
10 ans = [0] * day
11 for d in range(day):
12     m = int(input())
13     if m < x: ans[d] = buy[m]
14     else: ans[d] = n
15 print('\n'.join(map(str, ans)))      # dp_code : python3 280ms , pypy3-64 794ms
```

解题思路之 bisect 法：把所有价格 **排序**，再查找当天的硬币数位于价格列表的位置，它 **位置索引的值直接就等于能买的饮料瓶数**。例：在 [1,4,5,9,10] 中查 7 的位置，是第四位，则能买前边三瓶，而索引值刚好是 4-1=3。

补充：因为觉着 bisect 虽好用但写出来太长了，而且我反正只用这一个函数，就写了第一行。又觉着 int 和 input 用好多遍，又写了第二行。发现这样的处理果然耗时，细节写在注释，可以发现 bisect 和 dp 的效率基本一样。

关键词及符号：import bisect_right (**二分查找**)、sorted (**排序**)、[] for in range (**存储**)、join map str (**输出**)

```
1 from bisect import bisect_right as bi_r
2 i = int ; p = input ; p()
3 price = sorted(map(i,p().split()))
4 ans = [bi_r(price,i(p())) for _ in range(i(p()))]
5 print('\n'.join(map(str,ans)))
6 # bisect_code : python3 311ms , pypy3-64 842ms
7 # if without short_naming function : python3 280ms , pypy3-64 811ms
```

注：当时第一次做没多想，刚好在自学 dp 就按 dp 去写，整老半天才 AC。回过头看才发现，天呐，就写了一步排序而已，这题突然就简单多了... 搭配 bisect 库简直送分了，就算不 import，自己写个二分查找也不难（相对于 dp，写起来应该更容易一些，毕竟 dp 15 行，bisect 才 5 行，整整三倍啊！）。

选做：313B. Ilya and Queries, dp, 1300

<https://codeforces.com/contest/313/problem/B>

解题思路：三个步骤来帮小狮子帮他的朋友们解决问题：

1. 人话翻译：要据输入的区间输出其中 **相邻且相同的字符对数**，比如 '....' 有3对点， '#...#.##' 有2对点+1对井。
2. 接着要想如何取得并累计数据。我开了列表 **same** 存储从左到右累计的对数，以及变量 **cnt** 累计当前有几对，这样遍历字符串时 **每发现一对** 就让 **cnt+=1**，每次把 **cnt** 存进 **same**，就得到完整数据了。
3. 将输入的左右区间当作 **same** 的索引 (注意需要减1)，**右索引项(上限)减去左索引项(下限)** 就是区间内的相邻相同字符对数。存储答案后一次输出可以节省时间，这题需要善用内存来换时间，否则容易 TLE。

关键词及符号：`= []` (开空间)、`for in range len` (遍历)、`if == [] - +=` (判定&累计&索引)、`join map str` (输出)

```
1 s = input()
2 same = [0] ; cnt = 0 ; ans = []
3 for i in range(1, len(s)):
4     if s[i]==s[i-1]: cnt+=1
5     same += [cnt]
6 for _ in range(int(input())):
7     li, ri = map(int, input().split())
8     ans += [same[ri-1]-same[li-1]]
9 print('\n'.join(map(str, ans)))           # python3 654ms , pypy3-64 1590ms
```

选做：189A. Cut Ribbon, brute force/dp, 1300

<https://codeforces.com/problemset/problem/189/A>

解题思路：设列表 **dp** 的每一项为彩带上每个单位长度点 (包括首尾两点)，这样 **索引值就象征长度**，当然 **dp[0]=0**。**dp** 的预设值取 **-4000** 或任意更小的值，这样避免后边维护最大值时出问题 (利用tutor发现的)。遍历 **dp** 时取该长度 **i** 减去各个 **cut** 的长度 **x** 后到达节点的值 **dp[i-x]** 再 **+1**，这个值的内涵很深奥：**若之前有一种cut的最优方案允许这次新的cut，就决定这次要cut**。而同一节点又有多种 **cut** 方案，所以要维护 **max(dp[i], dp[i-x]+1)**。最后的一项 **dp[n]** 即为最大值 (最优解)。

关键词及符号：`= [] * (开空间)`、`{ } if <= min (剪枝)`、`for in range (遍历)`、`= [] max (维护dp)`

```
1 n,a,b,c = map(int, input().split())
2 dp = [0]+[-4000]*n                         # default==inf so won't bother others
3 cut = {k for k in {a,b,c} if k<=n}          # filter impossible cut value
4 for i in range(min(cut),n+1):                # start from first legal cut
5     for x in cut:                            # trying each cut
6         dp[i] = max(dp[i],dp[i-x]+1)          # dp[i]==can't cut, dp[i-x]+1==can cut
7 print(dp[n])                                # python3 61ms , pypy3-64 77ms
8
9 # if change line 3 to code below, python3 46ms, fastest I could get
10 cut = set()
11 for x in {a,b,c}:
12     if x<=n: cut.add(x)
```

注：上边第10-12行的功用与第3行等效，运行效率还更快 (主要因为第3行是comprehension，虽说代码短了，但理解所需时间增长)。另外我发现我还是无法足够精简地说明思路，说明我理解得不够透彻，核心意思写注释了。这两题难度较大的选做我都仔细研究了题解，学习到很多，但还远不足以说是熟练掌握了，还须再努力才行。

总结：经过多次调试代码，从 WA，到 TLE，到 AC，再到高效 AC，我有几题提交了接近20遍，有了以下发现：

1. 需要 **输出二维数组** 时，**'*'解包号** 是一个极其方便的工具。(本次两题二维数组都用上了)
2. **list.append** 的功能可以被 **`+=` 取代**。后者的效率通常略快一些，而放到数据量庞大的题(尤其是某些dp)，提速效果就非常显著了。好像是看某同学和老师在群里的交流注意到这点的，实测发现真香。不过，需要注意 **`+=` 后边的数据需要 **加一层 []****，因为这本质是视作两个列表的连接。
3. 忘了读哪本书时领悟到：**先开内存，后改赋值** 通常会更高效一些，比如已知要存5个答案后一次输出，不妨先开一个 `[0]* 5`，之后按索引赋值，这样 **比 append 或 `+=` 都快一些**。(其实就是 dp 的核心思想嘛)
4. 对于 **一次性输出多组答案** 的处理，有几种方法(招数)是比较常用的：(以下设 `ans` 为一存有答案的普通列表)
 1. **for a in ans: print(a)**：多次调用 `print` 所以很慢，为 **最低效能** 的基础手段。
 2. **print(*ans, sep='\n')**：今天课上老师提起我才想起这回事，实测发现 **效率往往输给后面几种**。
 3. **sys.stdout.write('{}\n'.format(ans))**：`sys` 库有一些酷炫的操作，有些特省时，只是我个人对这个写法了解不深，它也比较难调试(需要命令窗口)，但 **效率不错**。credit：启发自 [叶晨熙同学 assignment8 列出的数据表](#)，当时我 1443C. The Delivery Dilemma 写出了超快的代码也启发自此。
 4. **print('\n'.join(ans))**：高级做法，先合成一个巨大的字符串后再输出，此法的 **限制** 是 `ans` 内的 **元素都必须是字符串**，如若不是则需要 **调用 str 函数转换**，数据量稍微大点就比上边的都快，再分两种：
 1. **存进 ans 时先 str**：即 **在得到 ans 内的数据时就套上 str()**，数据量较小时比下一招 **略快一点**。
 2. **输出 ans 时才 str**：即 `print('\n'.join(map(str, ans)))`，数据量较大时都很快，**效率拔群**。
5. 一句话总结：上边的方法 **越往下越好**。但这是据我观察到的经验归纳的，不一定完全正确或标准，仅能作为一个参考。希望后续到学习中还能发现更妙的招数吧！

```
~>>>completed_in_24h = 5  
~>>>solved_CF>140 ; solved_OJ>70  
~>>>if burden < energy : will_continue = True  
~>>>else: speed /= 2
```

Assignment#A: 陈勇2100092701

文档撰写日期: 2021年11月24日 (星期三)

前言: 果然, 这周 burden > energy 了, 就是被寒假突然提前给压倒的... 虽说如此, 还算是比较快地完成了这周作业, 靠着提前刷过的题以及认真花时间去琢磨, 耗费我周二 (昨天) 几乎一整天的时间, 成功把代码都写出来了, 收获颇丰。要特别感谢的两个工具是 **tutor** 和 **leetcode**, 前者把 **抽象或复杂的操作清晰地具象化**, 后者在 **调试数据时效率非常高** (而且还能显示运行时间, 简直太棒)。

03532: 最大上升子序列和, dp

<http://cs101.openjudge.cn/practice/03532/>

解题思路: 先想如果子序列只有一项, 那他的和就是自己, 不妨直接 **让 dp 的初始状态和输入数组一样**。接着用双层循环控制索引, 比大小来 **判定“上升”**, 再维护最大值就好。这里要维护的是 **取 ($dp[j]+s[i]$) 或不取 ($dp[i]$) 当下那一项可得的子序列和更大** (所谓的状态转移方程)。注意 **dp 的最后一项未必是答案, 需要取 max**。

关键词及符号: *, list ("准备数组")、for in range [] ("控制索引")、if < ("判定上升")、max = + ("维护&输出")

```
1 n = int(input())
2 *s, = map(int, input().split())
3 dp = list(s)
4 for i in range(n):
5     for j in range(i):
6         if s[j] < s[i]:
7             dp[i] = max(dp[i], dp[j]+s[i])
8 print(max(dp))
```

注: 看到题解有一模一样的做法, 还使出了deepcopy, 我觉得大可不必, 善用数据结构及相关的函数就好。

16528:充实的寒假生活 (cs10117 Final Exam) 请用dp实现

<http://cs101.openjudge.cn/practice/16528/>

(这题我在之前的 Assignment#8 附录里详细说过, 这里就复制贴上, 当时看到作业被截屏发群里, 表示我的匠人精神受到了认可还是很开心的, 啊对这个部分的代码和后注已经被老师截过发过给同学们了 (~▽~))

借鉴了 2020fall-cs101 蔡清远同学 收录于题解中的代码, 结合我自己的理解写的, 以下列出每步思路:

1. 先记录每个活动, 这里用一个有61项的列表 **event**, 索引值=结束时间; 索引项=开始时间。这0-60的索引值, 也刚好方便之后遍历的动作。另外考虑到接收数据中 可能有 '0', 默认值取 '-1' 是妥当的替代。
2. 可能有几个活动同时结束却不同时间开始, 我们希望: **每个结束时间对应的开始时间尽量晚(大)**, 这样可以保证参加到 **当天结束的活动中时长最短的**, 所谓的 “**预处理**”。(这步其实内涵一个 greedy)

3. 开一个 **dp** 列表，存储最多能参与的活动数，**dp** 和 **event** 的 索引值相互对应，0-60也代表这61天。
4. 利用索引值 '**d**' (**day**)，遍历 **dp** 并赋值。设两个辅助变量：**remain = dp[d-1]** 为直到前一天能参加的最多活动数；**taking = dp[event[d]-1] + 1** 为选择参加这一天结束的活动后，能参加的活动总数。
补充 4.: "**event[d]**" 是该活动的开始时间，"**dp[event[d]-1]**" 是保证不和该活动冲突下最多能参与的活动(不包括该活动)总数。另外，**d从0开始是可以的**，反正 **dp[-1]** 也是0，**remain**就没毛病。**也是必须的**，否则不能处理到 "0 0" 的活动。
5. 遍历每一天无非两个情况：一，有活动，则决策是否参加；二，没活动，那肯定保留前一天的决策。如此即可一路更新最优解直到第61天。完事后，**dp** 的最后一项就是答案。

关键词及符号：= []* (开数组)、for in range (接收&遍历)、max (每次取更优数据/解)、if > else (决策判定)
#注释说明解题思路 1. - 5. 对应代码中的哪几行。

```

1 event = [-1]*61                                # 1. line 1
2 for _ in range(int(input())):                   # 2. line 2-4
3     start,end = map(int,input().split())
4     event[end] = max(event[end],start)
5 dp = [0]*61                                     # 3. line 5
6 for d in range(61):                            # 4. line 6-8
7     remain = dp[d-1]
8     taking = dp[event[d]-1] + 1
9     if event[d]>-1:                           # 5. line 9-13
10        dp[d] = max(remain,taking)
11    else:
12        dp[d] = remain
13 print(dp[60])                                 # credit: 2020fall-cs101 蔡清远

```

注：当时发现这位同学的代码符合我的思路就借鉴了。相比他的原代码 **我多存了两个变量**，以便于人类理解及程序调用，同时发现到 **题解的版本有些部分是不必要的** (没有实际功用)，我就拿掉了，果然还是AC。在**这个班里我们可以把前人留下的成果加以消化改良，而得到一个更好的成果，这个文化太赞了。**

02760:数字三角形, dp

<http://cs101.openjudge.cn/practice/02760/>

解题思路：每个数字有两条路可往下走，每往下一层会多一个数字，这就分解成了多个 **子问题**。直觉想到，**从底部逆行而上** 的话，在每两个数之间 **选大的加到上面的数**，最后顶层就是答案了，想通了马上 AC。

关键词及符号：双层 for in range [] + - ("控制索引")， max + ("取最大的加")

```

1 n = int(input())
2 tri = [[*map(int, input().split())]] for _ in range(n)]
3 for i in range(n-2, -1, -1):
4     for j in range(i+1):
5         tri[i][j] += max(tri[i+1][j], tri[i+1][j+1])
6 print(tri[0][0])                                # going up

```

从上至下的做法：反着走可以，正着来不行吗？带着好奇我尝试写了写，很快发现一个麻烦：**索引上一行时在边缘会越界**。说到“边缘越界”，我想不妨引入个**保护圈**的概念处理，在每行左右两侧加上0，不影响max判定又方便了之后的索引，结果竟然还真的成了，最后一行取最大即可。很满意的是自己的解法和题解基本一样，正中红心。

```
1 n = int(input())
2 tri = [[0, *map(int, input().split()), 0] for _ in range(n)]
3 for i in range(1,n):
4     for j in range(1,i+2):
5         tri[i][j] += max(tri[i-1][j], tri[i-1][j-1])
6 print(max(tri[-1])) # going down
```

18211: 军备竞赛, greedy/two pointer

<http://cs101.openjudge.cn/practice/18211>

解题思路：直观想法，把武器排序，有钱就造便宜的，没钱就卖最贵的，**凡造或卖了就确实地把它从数组里除去，那双指针一直指着首位和末位即可**。我用两个变量记录自家(me)和敌人(enemy)的武器数，一个特殊考虑是代码**第14行：若还有至少三份武器，且卖了最贵的还能再造便宜的，就卖，只要两个条件有一个不符就此打住**。这是因为只有两份时的买卖不再能使结果更好，三份或以上都有可能卖一份造两份，只有一份时则是会造成bug。不知道这样的“删除法”算不算规范的双指针解法，好像有点取巧了，希望这个AC不只是歪打正着罢了。

关键词及符号：sorted [0] [-1] (“**排序&指针**”)、if语句 < >= len (“**判定**”)、while break (“**控制结构**”)、del (“**删除**”)

```
1 p = int(input())
2 weapon = sorted(map(int, input().split()))
3 if p < weapon[0]:
4     ans = 0
5 else:
6     me = enemy = 0
7     while weapon:
8         make = weapon[0]
9         sell = weapon[-1]
10        if p >= make:
11            p -= make
12            me += 1
13            del weapon[0]
14        elif len(weapon) > 2 and sell + p >= make:
15            p += sell
16            enemy += 1
17            del weapon[-1]
18        else: break
19    ans = me - enemy
20 print(ans)
```

注：这个greedy一样是直观设想，不知道怎么证明局部最优和全局最优的等效性，还是老话一句：AC就好。考虑到这个好像不是很“two pointer”下面还有我第二次尝试写的代码。

可能比较规范的双指针版代码: 这里就用了两个变量做指针，从左右往中间靠拢，指针重合或不再获益时就结束(即 else 中的两个 break)。其中的变量 extra 直接把 me 和 enemy 取替了，也使得 if 语句的结构改变。

```
1 p = int(input())
2 weapon = sorted(map(int, input().split()))
3 extra = left = 0
4 right = len(weapon)-1
5 while left <= right:
6     make = weapon[left]
7     sell = weapon[right]
8     if p >= make:
9         extra += 1
10    p -= make
11    left += 1
12 else:
13     if left == right: break
14     p += sell
15     extra -= 1
16     if extra < 0:
17         extra = 0
18         break
19     right -= 1
20 print(extra) # truly two pointer code?
```

18106: 螺旋矩阵, matrices

<http://cs101.openjudge.cn/practice/18106/>

解题思路: 从左上出发，遇到边界就转向，我就知道这题矩阵又要请出 **保护圈** 了。先创建都是 0 的矩阵，用 1 围着，到时候我走着走着碰到前面不是 0 就转向，于是就可以一路转到中心去了。**如何转向？这是关键**。列出四个方向的索引公式后我悟了：**每当判定转向后，只要更改公式的参数就能控制索引实现螺旋了**。四个方向就四套参数，之前看题解文档时瞄到一眼，没去在意那是什么神秘的操作，自己推导出来的瞬间真是大彻大悟了。最后值得一提的是输出一个被加过保护圈的矩阵时，可以用简单的切片来避免保护圈被输出，着实方便。

关键词及符号: [] for in range * (“**矩阵&保护圈**”)、[] + % (“**控制转向**”)、for in [:] * (“**输出**”)

```
1 n = int(input())
2 edge = [[1]*(n+2)]
3 mx = edge + [[1, *[0]*n, 1] for _ in range(n)] + edge
4 turn = [(0, 1), (1, 0), (0, -1), (-1, 0)]
5 y = x = 1 ; t=0
6 dy, dx = turn[t]
7 for num in range(1, n**2 +1):
8     mx[y][x] = num
9     if mx[y+dy][x+dx]:
10        t = (t+1)%4
11        dy, dx = turn[t]
12        y += dy
13        x += dx
14 for row in mx[1:-1]: print(*row[1:-1])
```

注: 想通了这题，OJ 上其他带螺旋的题大多应该都能做了，之后有空我必定要刷他一波。

选做：16531: 上机考试 (cs10117 Final Exam) , matrices

<http://cs101.openjudge.cn/practice/16531/> (写到这才发现canvas作业说明里的题目编号是16532，一个小笔误)

解题思路：这题难就难在同一时间有太多事情要做了，我尽量简洁地列出我的 **五大步骤** (关键词太多就不列了)：

1. **前置作业**：声明变量：**字典cnt**，键=答对题数，值=答对此题数的人数 (对2题的共5人就有 2:5)；**字典ids**，键=考生编号，值=考生的答题情况 (01构成的数组)，特别的 **b** 象征边缘 (保护圈)。**good**=优秀人数、**same**=答题情况相同人数，两者是最后需要输出的答案。**n_problem**=总题数，处理特殊空输入；**total**=总人数。**edge**=保护圈上下层。这里也会把输入的编号存进加过保护圈的 **矩阵mx**。
2. **处理数据**：用第一个输入取总题数，若有第一个输入为空而后边的不为空就可能有bug (虽然不能解决，但测试数据不那么刁钻所以没事，甚至n_problem显得有点多余，但我还是尽可能实现题意的要求)。之后通过计算答对题数更新 **字典cnt**，也用原输入的01数组更新 **字典ids**。
3. **计算相同**：遍历矩阵判定每个考生的情况，**这里以矩阵mx的每一项为字典ids的键，实现了编号与答对题数的对应**，累加即可得 **same**。(代码中这块的写法启发自螺旋矩阵，简洁许多)
4. **计算优秀**：先补一个 **limit**=优秀率限制，**取字典cnt的每个键逆序排，再逐个用键取值，实现了从高分者开始累加优秀人数，直到超过优秀率停止**，最终就得到了 **good**。
5. **最后一步**：输出 **same** 和 **good**，事情就办完了。各步骤在代码里顺序进行且用一行空行隔开，方便理解。

```
1 M, N = map(int, input().split())
2 cnt = {} ; ids = {'b':-1}
3 good = same = n_problem = 0
4 total = M*N
5 edge = [['b']*(N+2)]
6 mx = edge + [[['b', *map(int, input().split()), 'b']] for _ in range(M)] + edge
7
8 for i in range(total):
9     try:
10         point = [*map(int, input().split())]
11         if n_problem == 0: n_problem = len(point)
12     except: point = [0]*n_problem
13     sum_pt = sum(point)
14     if sum_pt in cnt:
15         cnt[sum_pt] += 1
16     else:
17         cnt[sum_pt] = 1
18     ids[i] = point
19
20 for y in range(1, M+1):
21     for x in range(1, N+1):
22         ct = ids[mx[y][x]]
23         for dy, dx in ((-1, 0), (1, 0), (0, -1), (0, 1)):
24             if ct == ids[mx[y+dy][x+dx]]: same += 1 ; break
25
26 limit = total*0.4
27 key_cnt = sorted(cnt, reverse=True)
28 for k in key_cnt:
29     if good + cnt[k] > limit: break
30     good += cnt[k]
31
32 print(same, good)
```

第一次 AC 版代码：一些操作较冗长甚至多余，不过毕竟是难题，代码也难免很长，记录下这拨云见日的时刻。

```
1 M, N = map(int, input().split())
2 count = {}
3 ids = {'b':-1}
4 row = -1           #这一行是前面修改残留的，在此无意义
5 total = N*M
6 mx = [[*map(int, input().split())] for _ in range(M)]
7 for i in range(total):
8     try:
9         point = [*map(int, input().split())]
10    except:
11        point = [0, 0, 0, 0]
12    sum_pt = sum(point)
13    if sum_pt in count:
14        count[sum_pt] += 1
15    else:
16        count[sum_pt] = 1
17    ids[i] = point
18 same = 0
19 border = [['b']*(N+2)]
20 pro_mx = border + [[['b', *r, 'b']] for r in mx] + border
21 for y in range(1, M+1):
22     for x in range(1, N+1):
23         ct = ids[pro_mx[y][x]]
24         up = ids[pro_mx[y-1][x]]
25         do = ids[pro_mx[y+1][x]]
26         le = ids[pro_mx[y][x-1]]
27         ri = ids[pro_mx[y][x+1]]
28         if any((ct==up, ct==do, ct==le, ct==ri)):
29             same += 1 ; continue
30 key_count = sorted(count, reverse=True)
31 good = 0
32 limit = total*0.4
33 for k in key_count:
34     if good + count[k] > limit:
35         break
36     else:
37         good += count[k]
38 print(same, good)
```

注：当时一开始读题没读好，那么多件事情要办我就给办漏了好几样... 所以 **面对长长的题目和数据还是先要仔细理解题意**，我理解过后就觉得这是一题 **非常庞大的 implementation**，把事情办妥了就对了。另外，题解似乎没有考虑空的输入，代码里一些细节不太一样，但整体框架就是这样了。我的解法是利用了 **两个字典和矩阵对应**，比较像 **data structure** 吧，我猜想应该还有更好的解法，不过我对自己按部就班写的代码很满意了。

选做：455A. Boredom, dp, 1500

<https://codeforces.com/contest/455/problem/A>

解题思路：这题是提前做的，在CF第一页说明很多人都能做出来，虽然难度高但我还是尝试了，结果就花了2小时多... 我终于分解出可行的子问题：当最高分值为 n 时，需要考虑是取 **至 $n-1$ 分的最优解** 或取 **至 $n-2$ 分的最优解 + n · 数组中 n 的个数**。唉？怎么好像有点递归的味道？无论如何我发现可行就赶紧敲了代码，过了。我建两个数组：**an**，索引=分值，值=原数组中该分值的数量；**dp**，索引=所考虑的最大分值，值=考虑得到的最高分。先遍历一次原数组建造出 **an**，再遍历 **dp** 维护最大值，即：**取 ($dp[j-2] + an[j]*j$) 或不取 ($dp[j-1]$)**，这样 **dp** 的最后一项 即为**考虑了所有分值后的最优解**。特别说明：当 $j=1$ 时不会有bug，反正当时 $dp[-1]$ 也是0，直接就取 $an[j]*j$ 了。

关键词及符号：* (“接收输入”)、max [] * (“开数组”)、for in += (“遍历赋值”)、for in range max (“维护dp”)

```
1  input()
2  *a, = map(int, input().split())
3  x = max(a)
4  an = [0]*(x+1)
5  for i in a:
6      an[i] += 1
7  dp = [0]*(x+1)
8  for j in range(1,x+1):
9      dp[j] = max(dp[j-1], dp[j-2] + an[j]*j)
10 print(dp[-1])
```

tuple assignment 绝妙解法：当时看到题解最后面那一小段代码我震惊了... 这个操作看着很高级，其实又很好理解，太神了！我一定要吸收一点仙气！于是按照自己的理解并用自己的方式写了一版：

```
1  input()
2  *arr, = map(int, input().split())
3  x = max(arr)
4  an = [0]*(x+2)
5  for i in arr:
6      an[i] += i
7  a=b=0
8  for n in an:
9      a, b = max(a,b), a+n
10 print(a)
```

注：感觉 Boredom 这题 对数学思维的要求极高，只要能够想明白每次的策略其实代码就不难写了，关键就是这个太难想明白了，耗了我几张纸去代入和推理... 另外这个 tuple assignment 在上机考试题里的第三步其实也起到了启发作用，学会一套操作方法是长远的投资，之后遇到各种复杂的情况就有多一式招数可以应对了。

总结：这周作业做完，加上反复消化了书里和网上的 dp 介绍与教程，对 dp 的实际操作掌握更稳固了，之后要刷题保持感觉。也学到或发现到了几个新技巧这里列一下：

1. 其实 **deepcopy** 可以用 **data structure** 的方法实现，善用 **能改 class 的函数**。(启发自：**最大上升子序列和**)
2. 可以**让双指针一直在[0]和[-1]**，用 **while** 和 **del** 实现滚动。前提：所删的数据不再有用。(启发自：**军备竞赛**)
3. 利用保护圈及参数实现 **螺旋转向**，内容就不再细说，懂的都懂。(启发自：**螺旋矩阵**)
4. 读题要仔细，说白了一切可能都是 **implementation** 罢了，**别怕步骤多，先写写再说**。(启发自：**上机考试**)
5. 有一个东西特别妙，他叫 **tuple assignment**。(启发自：**Boredom的题解**)

另外，下边记录的刷题数其实是 **靠相对简单的题累积的**(被截屏好多次感觉自己有点虚，要说明一下)，比如 CF 我刷 800-1100 居多，OJ 也是挑出了所有我能做的题来。我是这么考虑的：既然老师说平时作业比期末难嘛，**得对简单的题保持一些感觉，我可不想成为那个“把题想复杂了”的人**，看过那么多题型应该会让我的编程基本功扎实一些。另外期末后有空了也会把所有资料整理一下提供给老师，希望后人少走冤枉路，在前人的基础上再有突破。

```
~>>>completed_in_24h = False  
~>>>will_continue_hardworking = True  
~>>> solved_CF>160 ; solved_OJ>80
```

Assignment#B：陈勇2100092701

文档撰写日期：2021年12月1日（星期三）

前言：到了这周，还是被绊了一下，好在还有题解可以参考，不然 dfs 我是不太能够做出来的。这份作业写得算是仓促，不再如以往般详尽，也略去了关键词对应，虽然记录少了但精华都在，脑子里有东西能行的。

18161: 矩阵运算(cs101-2017 期末机考备选), matrix

<http://cs101.openjudge.cn/practice/18161/>

解题思路：先判定条件，能算才算，否则输出Error。计算部分用了三层循环的组合，这是因为写完发现乘法和加法其实可以一起做（刚好A的列数和B的行数相同），所以就给拼起来了。

```
1 rowA, colA = map(int, input().split())
2 A = [[*map(int, input().split())] for _ in range(rowA)]
3 rowB, colB = map(int, input().split())
4 B = [[*map(int, input().split())] for _ in range(rowB)]
5 rowC, colC = map(int, input().split())
6 C = [[*map(int, input().split())] for _ in range(rowC)]
7 if colA != rowB or rowA != rowC or colB != colC:
8     print('Error!')
9 else:
10    for i in range(rowC):
11        for j in range(colC):
12            for k in range(colA):
13                C[i][j] += A[i][k]*B[k][j]
14    for row in C: print(*row)
```

注：接收部分不知道还可以怎么简化，可能可以写个函数重复调用？不过感觉效益不大，4次或以上可以试试。

02757: 最长上升子序列, dp

<http://cs101.openjudge.cn/practice/02757>

解题思路：就是 **最大上升子序列和** 的 变异型，差别在于 **dp的初始化值** 以及 **状态转移方程的"+1"**。

```
1 n = int(input())
2 s = [*map(int, input().split())]
3 dp = [1]*n
4 for i in range(n):
5     for j in range(i):
6         if s[j] < s[i]:
7             dp[i] = max(dp[i], dp[j]+1)
8 print(max(dp))
```

02773:采药, dp

<http://cs101.openjudge.cn/practice/02773>

解题思路: 很早之前(那时甚至还没学二维数组), 尝试了这题, 直到前两周正式进入了dp那时才做出来。dp列表在此以索引象征时间, 每项的值为若考虑该时限的最优解, 子问题就出来了。从逆序考虑每个药草, 因为后边的时限需要索引回前边, 顺序会错。这里有一个优化是若采某个药时长超过总时限则跳出, 不必进入dp遍历。

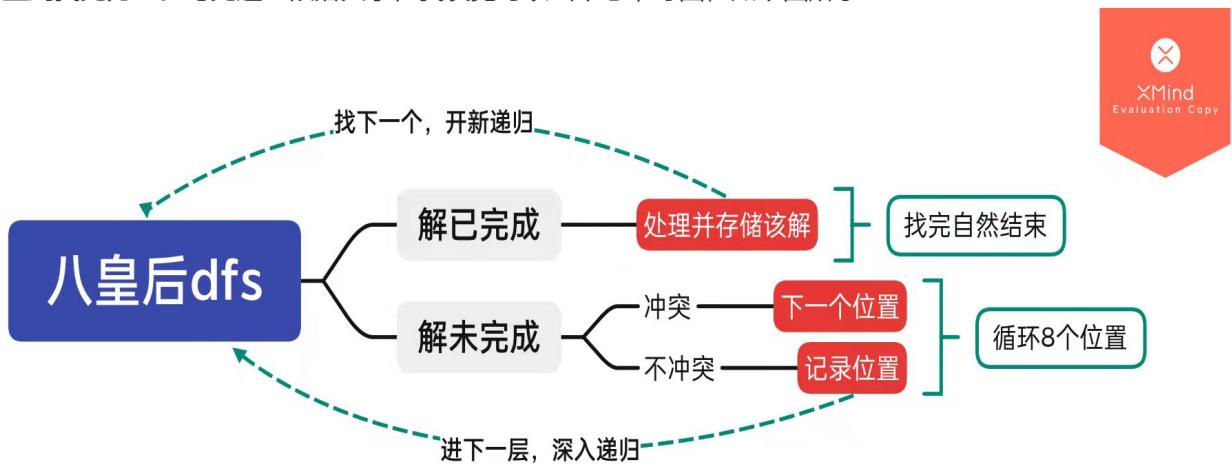
```
1 T, M = map(int, input().split())
2 herb = [[*map(int, input().split())] for _ in range(M)]
3 dp = [0] * (T+1)
4 for i in range(M):
5     pick = herb[i][0]
6     if pick > T: continue
7     earn = herb[i][1]
8     for time in range(T, 0, -1):
9         if time >= pick:
10             dp[time] = max(dp[time], dp[time-pick] + earn)
11 print(dp[-1])
```

注: 越来越喜欢把变量名起成**四个字母的单词**, 好多常用词都四个字母, 这样感觉很整齐, 又有意义, 还挺好。

02754: 八皇后, dfs

<http://cs101.openjudge.cn/practice/02754>

解题思路: 先看终止条件: 同行或同列或对角线, 因为自己刷过chess的题, 马上想起对角线可以用类似数学里的斜率概念处理。若不终止, 就记录位置后继续递归, 直到找到完成一个解(8个位置齐了), 就存起来并开启新一轮递归, 直到找完为止。写完这一段落文字, 我发觉可以画个思维导图, 如下图所示:



递归版dfs: 这是实实在在地调用同一个函数，如上图所示。不过，对于chess我也算略懂，因为棋盘是对称的，所以有一半的解通过 翻转棋盘 就可得到(其实还能再折半只是思考后觉着代码实现太复杂写不出来...)。所以找到46个解就可以停止递归，**取这些解与 99999999 (8个9) 的差即为其对称的另一解**，要把排序翻转一下符合题意。最后就是取答案输出了，最后两行用了很多 **comprehension**，这应该是 python 特别的风格吧，省了很多行。

```
1 # recursive_dfs_code
2 sol=[]
3 def queen8_rec(s, cx=0):
4     if len(sol) > 45: return
5     if cx == 8:
6         sol.append(''.join([str(i+1) for i in s]))
7         return
8     for c in range(8):
9         for r in range(cx):
10            if s[r] == c or abs(s[r]-c) == cx - r: break
11        else:
12            s[cx] = c
13            queen8_rec(s, cx+1)
14 queen8_rec([0]*8)
15 full_sol = sol[:] + [*reversed([str(99999999-int(s)) for s in sol])]
16 print('\n'.join([full_sol[int(input())-1] for _ in range(int(input()))]))
```

不递归版dfs: 这是第一次实际应用 **栈stack** 数据结构。其余大同小异，主要就是 **pop** 和**逆序遍历**。另外这里我选择了不用折半而直接 dfs 得到所有解，效果也不错，代码看着也简洁一些。这里要注意使用 **[:] 深拷贝列表**。

```
1 # using_stack_dfs_code
2 def queen8_stack(sol):
3     stack = [([0]*8, 0)]
4     while stack:
5         s, cx = stack.pop()
6         if cx == 8:
7             sol.append(''.join([str(i+1) for i in s]))
8             continue
9         for c in range(7, -1, -1):
10            for r in range(cx):
11                if s[r] == c or abs(s[r]-c) == cx - r: break
12            else:
13                s[cx] = c
14                stack.append((s[:], cx+1))
15    return sol
16 full_sol = queen8_stack([])
17 print('\n'.join([full_sol[int(input())-1] for _ in range(int(input()))]))
```

注：每个解的初始值是什么似乎不影响结果，因为都会被覆盖的关系吧，所以不必是None。

02698: 八皇后问题解输出, dfs

<http://cs101.openjudge.cn/practice/02698>

解题思路：前半段与上一题同，这里取的是**不递归版(stack)**，一个细节差异是**每个解将会被直接当作索引**，所以**不需再+1**。后边每次输出前**先备好8x8的0矩阵(空棋盘)**，**取每个解做索引赋上对应的1(放皇后)**，放好输出即可。

```
1 def queen8_stack(sol):
2     stack = [([0]*8, 0)]
3     while stack:
4         s, cx = stack.pop()
5         if cx == 8:
6             sol.append(s[:])
7             continue
8         for c in range(7, -1, -1):
9             for r in range(cx):
10                if s[r] == c or abs(s[r]-c) == cx - r: break
11            else:
12                s[cx] = c
13                stack.append((s[:], cx+1))
14    return sol
15 full_sol = queen8_stack([])
16
17 for n in range(92):
18     out = [[0]*8 for _ in range(8)]
19     for i in range(8): out[full_sol[n][i]][i] = 1
20     print('No. ' + str(n+1))
21     for row in out: print(*row)
```

另一种输出方式：把所有内容先装进列表，再合成一个巨大的字符串，print一次就输出了。虽然这题的时间限制允许我们一定程度上无视时间复杂度问题，但以下这段略长一些的代码确实跑得更快一些。

```
1 #another kind of output a huge string
2 output=[]
3 for n in range(92):
4     out = [[0]*8 for _ in range(8)]
5     for i in range(8): out[full_sol[n][i]][i] = 1
6     output.append('No. ' + str(n+1))
7     for row in out: output.append(' '.join(map(str, row)))
8 print('\n'.join(output))
```

注：这题还有一点要特别注意：行列的对应不能搞混，out要**先索引行(解里的数字)**，**再索引列**。

18108: 池塘数目 (cs101-2016 期末机考备选) , dfs

<http://cs101.openjudge.cn/practice/18108/>

解题思路：这是我的第一题成功AC的dfs，听完课才尝试的。我用三句话形象地把这题说完：踩到水了就把周边的水都踩过，凡踩过就把水踩没了，连续地踩过一轮就记一个池塘。

```
1 ans=[]
2 ng_8 = ((-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1))
3 def dfs(mtx, r, c, max_r, max_c):
4     at_edge = any((r<0, c<0, r >= max_r, c >= max_c))
5     if at_edge or mtx[r][c] != 'w': return
6     mtx[r][c] = '.'
7     for k in range(8):
8         dfs(mtx, r + ng_8[k][0], c + ng_8[k][1], max_r, max_c)
9     for _ in range(int(input())):
10        a = 0
11        N, M = map(int, input().split())
12        farm = [[*input()] for _ in range(N)]
13        for i in range(N):
14            for j in range(M):
15                if farm[i][j] == 'w':
16                    a += 1
17                    dfs(farm, i, j, N, M)
18        ans += [a]
19 print('\n'.join(map(str, ans)))
```

选做：02287, Tian Ji -The Horse Racing田忌赛马, greedy

<http://cs101.openjudge.cn/practice/02287>

解题思路：策略很多种，先考虑打平的话似乎就行不通（反正我试了1个小时多都行不通，哭啊）。这里应用了启发自军备竞赛的删除头尾，这手法在较复杂的贪心里还蛮常见。我的做法细节和题解略有不同，整体思路一样。

```
1 ans = []
2 while True:
3     n = int(input())
4     if n == 0: break
5     tian = sorted(map(int, input().split()), reverse=True)
6     king = sorted(map(int, input().split()), reverse=True)
7     win = lost = 0
8     for _ in range(n):
9         if tian[0] > king[0]:
10             win += 1
11             del tian[0], king[0]
12         elif tian[-1] > king[-1]:
13             win += 1
14             del tian[-1], king[-1]
15         elif tian[-1] < king[0]:
16             lost += 1
17             del tian[-1], king[0]
18     ans += [200*(win - lost)]
19 print('\n'.join(map(str, ans)))
```

选做：01088: 滑雪, dp

<http://cs101.openjudge.cn/practice/01088>

解题思路：这题基本上就是把题解完全消化后复现的... 我优化了某些部分，变量名也按我自己的理解来。一个遗憾是我原本想先找到最高点，后自该点起寻找最长路径，但没能把想法用代码成功实现，不知道这个思路行不行得通，我可能也没时间去研究了。另外这个想法好像也很耗时，所以才会需要dp吧。

```
1 R, C = map(int, input().split())
2 path = [[0]*(C+2) for _ in range(R+2)] #dp
3 edge = [[10001]*(C+2)]
4 hill = edge + [[10001,*map(int, input().split()),10001] for _ in range(R)] + edge
5 ng_4 = ((-1,0), (1,0), (0,-1), (0,1))
6 def ski(hill, i, j):
7     if path[i][j] > 0: return path[i][j]
8     for d in ng_4:
9         dr = d[0] ; dc = d[1]
10        if hill[i+dr][j+dc] < hill[i][j]:
11            path[i][j] = max(path[i][j], ski(hill, i+dr, j+dc)+1) #dp
12    return path[i][j]
13 ans = 0
14 for r in range(1, R+1):
15     for c in range(1, C+1):
16         ans = max(ans, ski(hill, r, c))
17 print(ans+1)
```

本周作业总结：还是列出几个值得注意的点：

- 函数相关变量：**尤其要注意 **递归函数的设置**，要区分个别 **变量适合什么位置**，就我粗浅的认知，主要有四个位置：**函数前、参数中、函数内、函数后**，四个位置怎么方便怎么来吧。
- 函数的通用性：**若想直接复制粘贴到别的代码中使用，要注意 **调整各个变量位置**，尤其是 **参数** 的部分。之后有空一定要好好整理一下，做好一套一套的通用函数集合，考试肯定用得上。
- 关于 return：**无返回值的 **return** 和 **continue** 有类似的作用，在 **递归dfs** 的时候可以应条件如此设置。一个函数的主要功能也很常不需要 **return** 来体现，把第1点应用得好即可解决许多问题。
- 关于 del：**适当的删除有时候能帮上大忙，**del** 后面 **还可以用逗号连续删除**，操作顺序会从左到右，过程中对数组的索引会重新判断，例：`a=[1,2]; del a[0], a[1]`，这就会报错了。如果是不同的数组就没事了。
- dfs的一些延伸理解：**一种方法是：“**凡走过必留下痕迹**”，全面覆盖的过程中取出或累计想求的答案。还可以这样：“**悄悄的我走了，正如我悄悄的来**”，这是面对原数据不可改的情况，记得每次“**带走一片云彩**”就行。

随着期末靠近，各科目学习压力一直提升，这周明显不再有时间仔细雕琢我的笔记，**更多时间花在研究课上ppt和题解里的代码**。对我来说，现阶段自己对学习编程安排的节奏已经缓了一些，暂且把策略调整成如老师说的“**bfs**”，备考前冲刺再“**dfs**”一下好了，在这两种学习模式间切换能够避免掉队，也能换来比较好的效率吧。我也不再要求自己把所有内容都吃得透透的，毕竟编程这行水很深，以后我还可以慢慢学，重点是稳住期末分数就好了。**全掌握就创造，半掌握就模仿，模仿着多写几遍，可能就都通了吧。**

~>>> will_continue_hardworking = True

~>>> solved_CF>190 ; solved_OJ>90

Assignment#C：陈勇2100092701

文档撰写日期：2021年12月9~10日（星期四~五）

前言：这周作业交得相对晚，是受各科的测验和作业影响，也是消化月度开销和帮助Jimmy两道难题耗时极长的缘故。看同学们都在积极做作业，我也就尽快完成了。此次代码大多都较长，思路叙述也相对应需要写得长些了。这次作业内容应该是最多的了，虽然我已经做好了大多数题目，还是耗时更久一些，老师批改也辛苦了。

20211125刘梓豪的MockExam

个人觉得这6题确实不太容易，比老师形容的期末难度还高些吧，只是不包含dfs而已，就算是前边的“简单题”也充满了坑，我当时能 AC 5 觉得很满意了。btw这位同学看来是要在闫老师的cs101千古留名了[doge]

03143:验证“歌德巴赫猜想”，math

<http://cs101.openjudge.cn/practice/03143>

解题思路：按照题意实现即可，精髓在于**如何获取素数**。看数据量好像还真的允许暴力计算，但我还是把筛法搬了出来，就是从之前CF 230B T-prime就写好的埃氏筛法函数 **eratosthenes**，略做调整即可。

```
1 ans=[]
2 x = int(input())
3 if x%2 or x<5:
4     print('Error!')
5 else:
6     def eratosthenes(n):
7         isprime = [1] * (n+1)
8         for i in range(2, int(n**0.5)+1):
9             if isprime[i]:
10                 for j in range(i*i, n+1, i): isprime[j]=0
11         return [x for x in range(2,n+1) if isprime[x]]
12 x_prime = set(eratosthenes(x))
13 for i in range(2, x//2 +1):
14     j = x-i
15     if i in x_prime and j in x_prime:
16         ans += ['%d=%d+%d'%(x, i, j)]
17 print('\n'.join(ans))
```

04030:统计单词数, string

<http://cs101.openjudge.cn/practice/04030>

解题思路：这个可说是段位极高的 string 题了，要求对字符串函数的透彻理解，**暗藏杀机**。幸运的是我当时在模拟考几次WA后结合老师在群里回答过其他同学的话，想出了比较巧妙的解法。(也没想到被好多同学认可借鉴，很荣幸)个人觉得小瑕疵是变量名有点长，不过意思比较明确，这里 **loc** 是 **location** (第一个位置)，**pro_stc** 是把原文句子 **sentence** 去除空格拆成的列表。精简总结我的思路：**预处理是接收时统一大小写并外加前后保护，先以字符串形式找位置，后以数组形式算个数**。在对的环境下做对的事，在对的数据上用对的函数，这就是精髓。

```
1 string = ' ' + input().lower() + ' '
2 sentence = ' ' + input().lower() + ' '
3 loc = sentence.find(string)
4 if loc < 0:
5     print(-1)
6 else:
7     pro_stc = sentence.split()
8     print(pro_stc.count(string[1:-1]), loc)
```

注：其实我也是受正则表达式思想启发，只是自己不熟悉怎么写就没 import re，后来老师果然用 re 写出来了。

02995:登山, dp

<http://cs101.openjudge.cn/practice/02995>

1. 看题第一眼：这不就是最长上升子序列么？看我复制粘贴，居然WA？
2. 再看第二眼：没毛病啊？限制条件和状态方程改改？题目里还藏着什么信息吗？
3. 终于第三眼：哦！你还可以下山是吧？！登山这个词还真博大精深...

解题思路：看完第三眼才领悟要点，然后就一直思考着直到模拟考将近结束才灵光一闪：**分别考虑上山和下山的两个dp过程，最后再结合成答案**，好像行得通。把景点 **翻转过来**，再进行一次dp，此时观察 spyder variable explorer 发现 **dp_down** 就是——对应每个景点的“**下山版 最长上升子序列**”，只是他也是反的，那我 **再翻转结果** 即可。把两个 dp 结果 **对应每项索引两两相加结合**，发现好像都多了 1，这是因为该项的景点在我两次dp中被重复选择了，那我减 1 就好，最后和最长上升子序列一样找 **max(dp)** 就得到答案了。

```
1 n = int(input())
2 view = [*map(int, input().split())]
3 dp_up = [1]*n
4 for i in range(n):
5     for j in range(i):
6         if view[j] < view[i]:
7             dp_up[i] = max(dp_up[i], dp_up[j]+1)
8 view.reverse()
9 dp_down = [1]*n
10 for i in range(n):
11     for j in range(i):
12         if view[j] < view[i]:
13             dp_down[i] = max(dp_down[i], dp_down[j]+1)
14 dp_down.reverse()
15 dp = [dp_up[x] + dp_down[x]-1 for x in range(n)]
16 print(max(dp))
```

注：记得有人尝试了同时考虑上下山的做法，不知道那位大佬做出来没有，挺想瞧一眼代码。(我想不到 @A@)

03670:计算鞍点, matices

<http://cs101.openjudge.cn/practice/03670/>

解题思路：先确定这不需要保护圈，不过肯定要一个两层的 for 循环。每行比较容易，每列就麻烦了，虽然我们常说不要替计算机工作，但有时候最简单粗暴的方法就是先帮计算机处理一部分难点，所以就有了下面的第5行。

```
1 mx = [[*map(int, input().split())] for _ in range(5)]
2 for i in range(5):
3     r_max = max(mx[i])
4     for j in range(5):
5         c_min = min((mx[0][j], mx[1][j], mx[2][j], mx[3][j], mx[4][j]))
6         if mx[i][j] == r_max == c_min:
7             print(i+1, j+1, mx[i][j]) ; break
8         if mx[i][j] == r_max == c_min: break
9     else: print('not found')
```

改良：当时想要是能**把矩阵转置**就方便了，网上一搜果然有，按我理解只要用**zip 函数和 * 解包号**就能实现。这里还多设了个变量 **saddle** 以表示是否为所求鞍点，看起来简洁一些。其余代码结构与原版基本一致。

```
1 mx = [[*map(int, input().split())] for _ in range(5)]
2 mx_t = [*zip(*mx)] # turn the matrix
3 for i in range(5):
4     r_max = max(mx[i])
5     for j in range(5):
6         c_min = min(mx_t[j])
7         saddle = mx[i][j] == r_max == c_min
8         if saddle:
9             print(i+1, j+1, mx[i][j]) ; break
10        if saddle: break
11    else: print('not found')
```

选做：19948:因材施教(greedy)

<http://cs101.openjudge.cn/practice/19948>

解题思路：又是一题奇妙的 greedy，原本的做法绕一大圈还是错的，看了题解思路才发现我想复杂了，关键就是最后一行 **sum(diff[m-1:])**。把 m-1 个最大的差去掉就好，因为按题意**一定可以把他们分到独立的班**（差为0）。

```
1 n, m = map(int, input().split())
2 rank = sorted(map(int, input().split()))
3 diff = []
4 if n==m or n<2:
5     print(0)
6 else:
7     for i in range(n-1, 0, -1):
8         diff += [rank[i] - rank[i-1]]
9     diff.sort(reverse=True)
10    print(sum(diff[m-1:]))
```

注：为了排版整齐把这题搬到前面了，月度开销就在下一页。

选做：04135:月度开销, binary search

<http://cs101.openjudge.cn/practice/04135>

解题思路：其实在《算法基础与在线实践》里见过这道练习题(可惜没参考答案)，就放在二分查找那一章。可是看了题一直没明白如何应用 bisect 库实现，后来才知道得自己写... 以下代码是我写了不知道几天都写不出来后去网上查的参考代码，我就把 C++ 翻译成了 Python，顺便学会一些基础的 C++ 语法。这里 3-13 行我是会写的，通过一个 for 循环接收数据，同时获取 **最大单日开销 max_exp** 和 **总开销 sum_exp**，可以避免遍历两次浪费时间，前面两个数据即为答案的 **下界low** 和 **上界upp**。14行起的 while 循环就是难点了，简而言之，**通过枚举答案，不断在题目限制下拉近上下界，直到重合时，答案就对了。**(这种题要是考试真遇到还真没把握能及时写出来...)

```
1 # https://blog.csdn.net/qq_51767234/article/details/118034450
2 # translated from C++ code, specialised binary search
3 N, M = map(int, input().split())
4 days = [0]*N
5 max_exp = sum_exp = 0
6 for d in range(N):
7     exp = int(input())
8     days[d] = exp
9     max_exp = max(max_exp, exp)
10    sum_exp += exp
11 low = max_exp
12 upp = sum_exp
13 mid = low + (upp-low)//2
14 while low != upp:
15     fajo = 1
16     sum_tmp = 0
17     for i in range(N):
18         sum_tmp += days[i]
19         if sum_tmp > mid:
20             fajo += 1
21             sum_tmp = days[i]
22         elif sum_tmp == mid:
23             fajo += 1
24             sum_tmp = 0
25     if fajo <= M:
26         upp = mid-1
27     else:
28         low = mid+1
29     mid = low + (upp-low)//2
30 print(mid)
```

注：有注意到老师发的参考答案是通过定义一个函数处理，好像整体过程短一些，不过思路是类似的。

20211202 cs101 2021 MockExam

这个模拟考真是极难... 可是我竟然提前刷过4题，所以有时间去研究马走日和 Jimmy。考的时候也把旧代码重新修缮了才提交的，不过还是明显快大家许多(我这可能算开挂了...)。后者确实无法研究明白就放弃了... 最终AC 5。

01852:Ants, math

<http://cs101.openjudge.cn/practice/01852/>

解题思路：这个理解方式很多种，重要前提是**把蚂蚁视作可以穿透彼此**，如此一来，**最短时间 min_time** 为：所有蚂蚁往自己靠近的一端边缘走，最靠中间的到了就结束。**最长时间 max_time** 为：最靠左的一路走到右，和最靠右的一路走到左，两只蚂蚁耗时最久的(就看看是“左的更左”还是“右的更右”)。

```
1 ans=[]
2 for _ in range(int(input())):
3     L, n = map(int, input().split())
4     ant = [*map(int, input().split())]
5     min_time = int(max([L/2 - abs(x-L/2) for x in ant]))
6     max_time = max(L-min(ant), max(ant))
7     ans.append([min_time, max_time])
8 for a in ans: print(*a)
```

04110: 圣诞老人的礼物-Santa Clau's Gifts, greedy

<http://cs101.openjudge.cn/practice/04110/>

解题思路：这是很早就刷过的题，用一个 **cp** 列表存储数据(所谓的cp值)，排序之后遍历取出答案即可。一个坑是**不能用字典存储，不然 cp 可能一样就覆盖了**，也是这题让我深刻地记住字典有这样一个缺陷的。

```
1 n, wx = map(int, input().split())
2 cp=[]
3 for _ in range(n):
4     v, w = map(int, input().split())
5     cp.append([v/w, v, w])
6 cp.sort(reverse=1); ans=0
7 for x in cp:
8     if x[2] <= wx:
9         ans += x[1]
10        wx -= x[2]
11    else:
12        ans += wx*x[0]
13        break
14 print('%.1f'%ans)
```

注：听说有同学用dp做，个人觉得那真是没有办法的办法，而这题要是能仔细读好题意不是太难的 greedy。

12558:岛屿周长(matrix)

<http://cs101.openjudge.cn/practice/12558/>

解题思路：上保护圈，**每个1上下左右有几个0 周长就得加多少**。在第8行转换为一个数学公式，如果四个位置都是0那么ans就+4了，每出现一个1就扣，遍历找到每块陆地(1)计算累计，这题就做完了。

```
1 ans=0
2 n, m = map(int,input().split())
3 edge = [0]*(m+2)
4 imap = [edge, *[ [0,*[ *map(int,input().split()),0]for _ in range(n)], edge]
5 for r in range(1, n+1):
6     for c in range(1, m+1):
7         if imap[r][c]:
8             ans += 4 -imap[r-1][c] -imap[r+1][c] -imap[r][c-1] -imap[r][c+1]
9 print(ans)
```

12757:阿尔法星人翻译官, string

<http://cs101.openjudge.cn/practice/12757/>

解题思路：第一步肯定是乖乖准备好翻译用的字典。关于负数处理，我只用了一个变量 **neg 存储布尔值**，然后直接把 negative 从文字里去掉(用完就扔)免得妨碍后续步骤，最后利用 **False = 0 及 True = 1 的等价关系** 就可以极简地处理好这个繁琐的问题。最核心的是第10行 for 循环，我利用两个变量 x y 处理数字，**注意必须先特别处理“百万”、“千”、“百”这三个词，因为会有乘法关系，其余的就是加法关系。**

```
1 trn = {'zero':0,'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,
2       'eight':8,'nine':9,'ten':10,'eleven':11,'twelve':12,'thirteen':13,
3       'fourteen':14,'fifteen':15,'sixteen':16,'seventeen':17,'eighteen':18,
4       'nineteen':19,'twenty':20,'thirty':30,'forty':40,'fifty':50,'sixty':60,
5       'seventy':70,'eighty':80,'ninety':90,'thousand':1000,'million':1000000}
6 raw = input().split()
7 neg = raw[0]=='negative'
8 if neg: del raw[0]
9 x = y = 0
10 for i in raw:
11     if i in {'million', 'thousand'}:
12         x += y*trn[i]
13         y = 0
14     elif i == 'hundred':
15         y *= 100
16     else:
17         y += trn[i]
18 ans = x + y
19 print([ans,-ans][neg])
```

注：发现题解里的代码是一步一步慢慢来处理的，我的处理方式类似走捷径就快了好几步，所以代码更短一些。一个细节是字典里没存 hundred，其实不存 million 和 thousand 也可以，因为反正要搞特殊，不妨就后边处理。

04123:马走日,dfs

<http://cs101.openjudge.cn/practice/04123/>

解题思路：这题是因为考之前研究了老师课上的ppt，加上考的时候看了一眼才写出来的。我定义函数时对于参数的处理比较麻烦，因为按我理解，**写一个函数就是为了方便调用嘛，应该要把必要的变量纳入参数考虑。**这样的好处是下次复制到别的地方可能更容易适用，坏处显然是代码看起来更长一些。有几点值得注意：

1. **stp 是关键变量**：内涵为**马走了几步**，要走完棋盘显然要走**长x宽 步**，所以一旦走完一遍就往答案累计+1。
2. 第11-14行：**dfs 核心部分**，一并处理了**走没走过的判定及边界条件**，也同时体现了**递归与回溯**。
3. 第17行：**为什么是10？**按我理解，那便是**棋盘大小的上限**，这样搭配输入的长宽限制处理起来更方便。

```
1 ans = [] ; a = 0
2 ri_8 = ((2,-1),(2,1),(-2,-1),(-2,1),(-1,-2),(1,-2),(1,2),(-1,2))
3 def dfs(mtx, r, c, max_r, max_c, stp):
4     if stp == max_r*max_c:
5         global a
6         a += 1
7         return
8     for i in range(8):
9         dr = r + ri_8[i][0]
10        dc = c + ri_8[i][1]
11        if not mtx[dr][dc] and 0 <= dr < max_r and 0 <= dc < max_c:
12            mtx[dr][dc] = 1
13            dfs(mtx, dr, dc, max_r, max_c, stp+1)
14            mtx[dr][dc] = 0
15    for _ in range(int(input())):
16        n, m, x, y = map(int, input().split())
17        board = [[0]*10 for _ in range(10)]
18        a = 0 ; board[x][y] = 1
19        dfs(board, x, y, n, m, 1)
20        ans.append(a)
21    print('\n'.join(map(str, ans)))
```

选做：01661:帮助 Jimmy, dp/dfs

<http://cs101.openjudge.cn/practice/01661/>

解题思路：这题我肯定不会... 以下代码我是看着题解代码进行了模仿与个人风格化后写出来的。看了网上各种版本的解析后整体思路都明白了，就是代码写不出来。模仿完略懂一点，但没彻底搞懂代码实现过程的每个细节，只能算是局部理解。我能做的就是**把抽象的代码用存储变量的方式具象化**，因为Python的语法大多贴近英语，看着变量名我大概能知道怎么回事。这也是我刷了那么多题，看了那么多比我优秀的真大佬 和较晚开窍的零基础同学写的代码后悟出的心法：**面对复杂的难题，适当地多设几个变量，起几个好名字，代码可读性将大幅提升**。所以这题我就抱着学习课外超纲知识的心态去做的，就不期望掌握整个过程，应该把时间花在巩固其他课内的基础上，比如**dp、二分查找、dfs 等经典算法**，这才是更好的选择。(以及多花点时间学习其他巨难的科目啊！QAQ)

注：这题好像还有纯 dp 的解法，遗憾的是我不太熟悉 C++ 语法，网上代码里好几个地方是复杂语法我无法翻译...

```

1 def dfs(i:int, a:int, x:int):
2     if dp[i][a] != -1:
3         return dp[i][a]
4     high = plat[i][2]
5     left = right = 40000001
6     safe = True
7     for j in range(i+1, N+1):
8         fall = high - plat[j][2]
9         if fall > MAX:
10             safe = False ; break
11         le_j = plat[j][0]
12         ri_j = plat[j][1]
13         if le_j <= x <= ri_j:
14             left = dfs(j, 0, le_j) + fall + x - le_j
15             right = dfs(j, 1, ri_j) + fall + ri_j - x
16             safe = False ; break
17     dp[i][a] = min(left, right)
18     if safe and high <= MAX: dp[i][a] = high
19     return dp[i][a]
20
21 ans=[]
22 for _ in range(int(input())):
23     N, X, Y, MAX = map(int, input().split())
24     dp = [[-1, -1] for _ in range(1010)]
25     plat = [[X, X, Y]]
26     for _ in range(N):
27         plat.append([*map(int, input().split())])
28     plat.sort(key = lambda x:x[2], reverse=True)
29     ans.append(dfs(0, 1, X))
30 print('\n'.join(map(str, ans)))

```

总结：这两周模拟考做完更多的是针对考试的一些感想与策略：

1. **先做对，再做好：**考试的时候哪有那么多时间想最优解？能过的解法就是好解法，事后再琢磨改良也不迟。
2. **时间少，合理分：**硬磕一道题或多尝试几题，开始时先看过每道题就做决策，以求 $\min(\text{time}) \& \max(\text{AC})$ 。
3. **读了题，试着写：**不太理解题意没关系，能写出一段代码实现就好，不写不知道，一写猜对了，岂不很赚？
4. **写不来，再读题：**反复读题是必要的，须抓取关键信息，登山、统计单词数、圣诞礼物等都是很好的例子。
5. **善搜索，用资源：**所谓“开卷考试”，在不作弊前提下能获取到什么资源也是考核的一部分吧，老师的ppt，历年题解，参考书籍，自己的模板，google 百度，需要时可以穷尽一切资料来源。所以我自己尝试后总会再去参考题解，毕竟那里头是代代相承的精华，含金量常常会比我自己的脑子高许多，必定值得研究消化。

我可能和别的同学不同，计概是我的专业基础必修课里我（唯一）比较有把握的地方，预计期末保5冲6，这是大一的我离100分最近的科目了。作业一定会完完整整地连同选做一起交，就是有时候得晚点交，真的太难的题就是学习与模仿，慢慢累积总会有所进步的。因为刷题量刚好累积破了整数（以及期末季的压力）所以现在每天刷题适当减少了，忙的时候最多做一道 CF 的基础语法题，希望可以一直保持状态到 12.23 当天的下午5.00。

~>>> will_continue_hardworking = True

~>>> solved_CF>200 ; solved_OJ>100

Assignment#D：陈勇2100092701

文档撰写日期：2021年12月14~15日（星期二~三）

前言：最后一份作业了，时间过得好快啊。这应该也算是难度最大的一份作业了，好在其中几题我提前做过，且选做题基本都依赖了现成的代码作为参考，我才能比较快地完成。

21608: 你和你比较熟悉的同学， bfs/dfs and similar

<http://cs101.openjudge.cn/practice/21608>

解题思路：个人觉得 **递归式的dfs** 还是比较容易想的，就是针对图(树)中每个同学遍历，再对其各自的子节点遍历。遍历完一个循环就能取得一个顶点个数，**一直维护最大值直到遍历所有人** 即可。注意：若连接到 -1，**实际上是没有连接子节点的**，所以需要**忽略**(代码中以删除实现)。这题代码(及许多类似题型)就是明显地分**三部分**：

一、遍历搜索的函数(可能需要沿途计算或赋值)。二、创建图(树)。三、利用'一'的数据维护或计算答案。

变量注释：tree就是树，stud指同学，seen是访问过的同学(见过的)，know是熟悉的同学(即子节点)，link是每个节点的关系(连接)，maxi是最终要维护的最大值(maximum)。这些变量名对我自己来说一目了然，很有帮助。

```
1 # dfs - recursion method
2 # recursion is usually easier to be understood (for me)
3 def dfs_rec(tree:dict, stud:str, seen:list):
4     if stud not in seen:
5         seen.append(stud)
6         if stud in tree:
7             for know in tree[stud]:
8                 dfs_rec(tree, know, seen)
9     return seen
10
11 tree = {}
12 for _ in range(int(input())):
13     link = input().split()
14     tree[link[0]] = link[2:]
15
16 maxi = 0
17 for stud in tree:
18     seen = dfs_rec(tree, stud, [])
19     if '-1' in seen:
20         seen.remove('-1')
21     maxi = max(maxi, len(seen))
22 print(maxi)
```

注：各版本的代码一开始写其实结构都不太优，虽然测试数据过了但代码不好看，于是参考题解做了修缮。写完后许久才想到其实seen中的元素是不会重复的，不妨用set，或许效率更高，只是真是太后知后觉了就没再改了。

另一写法：递归式dfs函数还可以这么写，实际运行基本没差，但看起来给人不一样的感觉，可能有助理解。

```
1 # another pattern with more return and less indentation
2 def dfs_rec(tree:dict, stud:str, seen:list):
3     if stud in seen: return seen
4     seen.append(stud)
5     if stud not in tree: return seen
6     for know in tree[stud]:
7         seen = dfs_rec(tree, know, seen)
8     return seen
```

用栈处理：虽然stack可以优化dfs的运行，但写起来不太容易，不同题型有各种变化所以可能混淆。特别的，因为没有了递归的特性，记录用的seen不必再作为参数。但需要初始值(我取名init，意指initial)。最重要的是注意每步的逻辑和顺序，拿纸笔列出来帮助很大。pop的部分容易把人搞乱，因为要优先考虑深入嘛，所以一旦还可以再深入哪怕一步都不能先pop，为此需要引入布尔值变量(我取名do_pop)辅助判定。

```
1 # dfs - using stack method (non-recursion)
2 # stack is nice, but its usage is harder to be understood
3 def dfs_stk(tree:dict, init:str):
4     seen = [init]
5     stack = [init]
6     while stack:
7         stud = stack[-1]
8         if stud not in seen:
9             seen.append(stud)
10        if stud not in tree:
11            stack.pop()
12            continue
13        do_pop = True
14        for know in tree[stud]:
15            if know not in seen:
16                stack.append(know)
17                do_pop = False
18                break
19        if do_pop: stack.pop()
20    return seen
21
22 tree = {}
23 for _ in range(int(input())):
24     link = input().split()
25     tree[link[0]] = link[2:]
26
27 maxi = 0
28 for stud in tree:
29     seen = dfs_stk(tree, stud)
30     if '-1' in seen:
31         seen.remove('-1')
32     maxi = max(maxi, len(seen))
33 print(maxi)
```

初尝试bfs：知道要用queue来代替stack，可又是“道理我都懂，代码写不好”，看了一眼题解就都明白了。

```

1 # bfs - using queue method (extra, kinda at the top tier of syllabus)
2 # queue is nice too, differ from stack, but usage is similar concept
3 def bfs_que(tree:dict, init:str):
4     seen = []
5     queue = [init]
6     while queue:
7         stud = queue.pop(0)
8         if stud not in seen:
9             seen.append(stud)
10            if stud in tree:
11                for know in tree[stud]:
12                    queue.append(know)
13    return seen
14
15 tree = {}
16 for _ in range(int(input())):
17     link = input().split()
18     tree[link[0]] = link[2:]
19
20 maxi = 0
21 for stud in tree:
22     seen = bfs_que(tree, stud)
23     if '-1' in seen:
24         seen.remove('-1')
25     maxi = max(maxi, len(seen))
26 print(maxi)

```

02711: 合唱队形, dp

<http://cs101.openjudge.cn/practice/02711>

解题思路：这，不就是登山吗？合唱队里的人即为景点，请人出列就是“**把景点移除**”，而且如果有登山的最优解是最多能去几个景点，换句话就是“**最少能不去几个**”。如此推理，题目所求即——**若想使得：满足前半段递增与后半段递减的最长子序列，是每项相连的，需要移除几项？**代码与登山只差最后一行，**总人数减去最优解即可。**

```

1 n = int(input())
2 stud = [*map(int, input().split())]
3 dp_up = [1]*n
4 for i in range(n):
5     for j in range(i):
6         if stud[j] < stud[i]:
7             dp_up[i] = max(dp_up[i], dp_up[j]+1)
8 stud.reverse()
9 dp_down = [1]*n
10 for i in range(n):
11     for j in range(i):
12         if stud[j] < stud[i]:
13             dp_down[i] = max(dp_down[i], dp_down[j]+1)
14 dp_down.reverse()
15 dp = [dp_up[x] + dp_down[x]-1 for x in range(n)]
16 print(n-max(dp))

```

注：虽说是复制粘贴，几个关键的变量名还是改一下，方便以后阅读，要不然时隔许久忘了就完了。

02694:波兰表达式，recursion

<http://cs101.openjudge.cn/practice/02694/> (做作业时还叫“逆”，这里是事后编辑，rpn 的 "r" 就不该有了)

解题思路：按题意，**递归需要从左到右取出表达式的元素**，直接在递归函数里设置变量可以实现索引控制，所以只要按部就班逐个判断肯定能完成，(编辑：但我不想，于是使出eval函数)。(其实是写了代码试运行后发现结果符合才大概明白怎么回事的，这个逆波兰表达式的逻辑还是有点绕，主要是题面和网上搜索的结果不符合的缘故)

```
1 statement = input().split()
2 def rpn():
3     item = statement.pop(0)
4     if item in '+-*':
5         return eval('%.8f%s%.8f'%(rpn(), item, rpn()))
6     else:
7         return float(item)
8 print('%.6f'%rpn())
```

感谢李文梁同学分享：可以直接用字符串格式递归来规避精度问题，最后输出时再以浮点数格式化调整即可。

```
1 statement = input().split()
2 def rpn():
3     item = statement.pop(0)
4     if item in '+-*':
5         return str(eval(rpn() + item + rpn()))
6     else:
7         return item
8 print('%.6f'%float(rpn()))
```

注：原本就是想到终于可以应用早就认识的 eval 函数，没想到被精度问题 WA 了不知道几次... 浮点数确实麻烦。这样短短一串代码看起来也挺简洁的，就是不想写那么多重复的语句，在老师和同学帮助下终于成功了。

21554:排队做实验 (greedy)

<http://cs101.openjudge.cn/practice/21554/>

解题思路：首先理解每个人等待的时间会有以下关系：显然先让耗时短的做最好，那就把每人耗时升序排，再用 for 循环控制 **乘积数 k**，为了取得未排序数组的索引不妨再设 **索引数 j**，在循环中取出或累计答案即可。

```
1 # odd code remastered
2 n = int(input())
3 queue = [*map(int, input().split())]
4 q_sort = sorted(queue)
5 q_indx = []
6 j = time = 0
7 for k in range(n-1, -1, -1):
8     time += q_sort[j]*k
9     i = queue.index(q_sort[j])
10    q_indx.append(i+1)
11    queue[i] = 0
12    j += 1
13 print(*q_indx)
14 print('%.2f'%(time/n))
```

改良版本：应用 **enumerate**函数，利用 其产生的二维数组作 for 的遍历，对乘积数 k 换了更简洁的1处理方式。虽然但是，我自己觉得我的两版代码看起来各有优劣，就是控制结构细节不同，可能就看个人习惯与喜好了。

```
1 n = int(input())
2 queue = enumerate(map(int, input().split()), start=1)
3 q_sort = sorted(queue, key = lambda x:x[1])
4 q_indx = []
5 time = 0
6 k = n-1
7 for stud in q_sort:
8     indx, wait = stud
9     time += k * wait
10    q_indx.append(indx)
11    k -= 1
12 print(*q_indx)
13 print('%.2f'%(time/n))
```

08210: 河中跳房子, binary search

<http://cs101.openjudge.cn/practice/08210>

解题思路：原本做月度开销没写判定函数，写得比较散，导致这题不能直接套用。于是先复习了月度开销的代码，理解了 **check** 函数的运作，经过几轮痛苦的 debug 才成功，结果也趋近了题解的代码。思路就是很直接地把代码用英文说一遍(近似月度开销)，我的关键词汇(变量和函数名)都有合适的名字，这里不再细说。**注意二分的关键：**因为 **太远时上限 upp 不能再超过 mid (则 取mid)**，而 **太近时下限 low 应该往上推一点点 (则 取mid+1)**。

```
1 L, N, M = map(int, input().split())
2 rock = [0, *[int(input()) for _ in range(N)], L]
3 def too_far(x):
4     num_rock = cur_rock = 0
5     for i in range(1, N+2):
6         if rock[i] - cur_rock < x:
7             num_rock += 1
8         else:
9             cur_rock = rock[i]
10        if num_rock > M:
11            return True
12        else: return False
13 low, upp = 0, L
14 while low != upp:
15     mid = (low + upp)//2
16     if too_far(mid):
17         upp = mid
18     else:
19         low = mid+1
20 print(low-1)
```

注：为什么上限叫 upp？英文有这词？其实就是为了统一同一系列的变量名使得整体看起来整齐，自创词汇。

选做：21577: 护林员盖房子

<http://cs101.openjudge.cn/practice/21577/>

解题思路：于我来说这算是矩阵题里很难的了，经过搜索得知其内涵叫做“**最大子矩阵问题**”，我粗浅的理解是结合了固定方向的矩阵遍历和dp，以**找到最符合要求的矩阵/矩形**的一套方法。我自己的思路是加上保护圈后各别计算空地的高(行)和宽(列)，就是遍历两次。然后把两套数据结合，再遍历一次并沿途维护最大面积。可是这怎么写啊？只写了前半段发现后面写不来，只好又去搜索，发现了**第三层for循环**的写法，也就是把**局部遍历嵌入在全局遍历里头**。前半段和我的思路相符，**把矩阵翻译成存储空地的宽度信息的样子**；后半段是我想法的简化，直接**以行数差得到高度，并以高度为优先**算面积。于是我按着这个写法又从C++翻译了一版Python代码。我特别简化了保护圈，只加在左上侧，因为**遍历方向是固定的，所以右下侧用不着保护**。（wdth 指宽度，原字 width）

```
1 # refer to https://blog.csdn.net/xuyang0905/article/details/109459022
2 # core function tranlated from C++
3 m, n = map(int, input().split())
4 land = [[0]*(n+2), *[0, *map(int, input().split())] for _ in range(m)]
5 for r in range(1, m+1):
6     for c in range(1, n+1):
7         if land[r][c]:
8             land[r][c] = 0
9         else:
10            land[r][c] = land[r][c-1] +1
11 area = 0
12 for r in range(1, m+1):
13     for c in range(1, n+1):
14         if land[r][c]:
15             wdth = land[r][c]
16             area = max(area, wdth)
17             for up_r in range(r-1, -1, -1):
18                 if land[up_r][c]:
19                     wdth = min(wdth, land[up_r][c])
20                     area = max(area, wdth*(r - up_r +1))
21                 else: break
22 print(area)
```

注：这题看起来不难，实际还是很烧脑，要是没有网络资源，我可能一直卡在“思路过了，代码不过”的阶段。

选做：18164: 剪绳子，greedy/huffman

<http://cs101.openjudge.cn/practice/18164/>

解题思路：看到老师的提示，我知道这题不是我能用已知的知识轻易完成的，果断先去学习了**heapq库**。有了初步概念后发现这题就是所谓的“典型题”，选对函数实现就好，于是跟着题解写出了基本一个样的代码。关于greedy，（编辑：我原本是想反了，不太对，应该想象从最小段的绳子拼回整条绳子，才是与heap相同的概念）。用**heapq**写的过程中感悟到，**这个库其实就是把greedy用特殊的数据结构实现了**，说到底是一个思想。

```

1 import heapq
2 N = int(input())
3 cut = [*map(int, input().split())]
4 heapq.heapify(cut)
5 ans = 0
6 for _ in range(N-1):
7     a = heapq.heappop(cut)
8     b = heapq.heappop(cut)
9     cost = a + b
10    heapq.heappush(cut, cost)
11    ans += cost
12 print(ans)

```

注：能力与时间有限，没能仔细研究 heapq 库，没有百分百把握处理类似但不同的题，可恶的期末季啊...

选做：580C.Kefa and Park, bfs and similar, 1500

<https://codeforces.com/contest/580/problem/C>

解题思路：已知对 bfs 更友好，且是双向图（但我还是管他叫tree），而这些对我来说超纲，所以：**看题解**。整个思路我理解了一遍，其实我只要把各个变量的内涵说明一下思路就自然出来了（我决定以注释形式直接写代码里）。以下是我取往届同学代码为本，参考网上资料及另一版本的题解，综合取优后按自己的理解写的版本。我只能相信考试不至于那么难了，如果老师说的都是真的，肯定也不至于这么难的。

```

1 # refer to code of 2020fall-cs101, Yiyun LIN
2 n, maxi = map(int, input().split())      # maxi 最大能容忍连续出现几只猫
3 c_01 = [*map(int, input().split())]       # c_01 全是0或1的记录列表，0无，1有(猫)
4 cats = { 1 : c_01[0] }                   # cats 每个节点是否有猫的辅助字典
5 tree = {}                                # tree 其实是一个graph，且是双向的
6 for _ in range(n-1):                      # 6-11 读入每个节点信息，try语句优化了结构
7     x, y = map(int, input().split())
8     try: tree[x].append(y)
9     except: tree[x] = [y]
10    try: tree[y].append(x)
11    except: tree[y] = [x]
12 path = 0                                  # path 最终所求答案，即有几条路可走到底(餐厅)
13 went = set()                             # went 去过的，记录访问过的节点
14 queue = [1]                               # queue 实现bfs的核心数据结构
15 while queue:                            # 它空了即代表搜索完毕
16     vrtx = queue.pop(0)                  # vrtx 简写自 vertex，顶点，和node(节点)类似
17     went.add(vrtx)                      # 记录该点为去过的
18     if cats[vrtx] > maxi: continue    # 猫太多不能走了，撤出(跳出循环)
19     more = False                         # more 布尔值，有没有"更多"相邻点
20     for k in tree[vrtx]:                # 遍历每个相邻的点
21         if k not in went:              # 若存在没去过的点
22             if c_01[k-1]:               # 且该点有猫，累计猫的数量+1
23                 cats[k] = cats[vrtx] + 1
24             else:                     # 没猫就断了连续，累计数归0
25                 cats[k] = 0
26             queue.append(k)           # 入队，bfs关键时刻
27             more = True              # 能跑到这一步意味着有"更多"点可以走
28     if not more: path += 1                # 没有"更多"了，已经走到底，一条路形成
29 print(path)

```

注：注释写着写着竟近乎逐行讲解，不敢说自己理解得完全正确，若发现注释的意思有误还请指正。值得注意的是这代码居然没有 def，要照搬应用就需要更多修改了，希望期末不会用得上。

总结：到这个时候了，挑一些比较普适性的事情列出来提醒我自己吧：

1. 考试备好纸笔，敲键盘之前要 **先画画**。手写一些 **伪代码** 也可能是个好主意。
2. 开好 **Leetcode** 方便跑数据；注意 **variable explorer** 容易看细节；挂着 **tutor** 以防有意外。
3. 考前帮电脑做个检查，**硬件软件绝对都不能挂**。还有 **WiFi 路由器**，一个可以决定线上考试成败的狠角色。

最后的总结，特别多好说，又不知道该着重说什么，其余留待考试前我统整好所有作业笔记后一并写吧。刷题数考试前肯定还会变，实际数据就记录在最终的那份文档里好了。我打算把我一学期创造的资料打包分享，想说既然难得有一门课我学得还不错（这句话背后的辛酸我就不分享了），就回馈一下课堂吧。学习这门课确实仰赖各方面的资源，希望我的学习成果可以作为日后选课同学的参考之一。我零基础起步，深知学习历程点点滴滴，期望以此能让与我有类似背景的同学，尤其是留学生，知道一件事：**零基础一样能学好计概B**。

```
~>>> will_fight_for_exam = True  
~>>> solved_CF > 200 ; solved_OJ > 100  
~>>> Final_Sum陈勇2021fall.pdf and Code_Pack陈勇2021fall.zip coming soon
```

注：最后决定不记录具体刷题数了，说不定还会加呢，借这个空位再次强调前言说过的几点须知：

一、任何题目请优先以老师维护的两份官方题解为准，请勿将此文档视作标准题解。

Important Reminder: Please regard teacher's answer collections as main references, DON'T regard this document as standard answer.

二、文档内容可能存在的问题包括但不限于：复制粘贴错误、错别字、OJ题号及链接改变（详情可咨询老师）。

三、若读者发现此文档有误，可以咨询老师、助教或班上同学，也可以直接咨询笔者（有空才能维护）。

四、强烈建议查阅此题解时与 Code_Pack_yeagle_ting2021fall.zip（此名称才是对的）中的相关内容对照。

五、笔者 Python 入门，目前也只对 Python 掌握，顶多能看懂小部分 C++ 基础语法，风格 Pythonic 勿见怪。

Assignment#E：陈勇2100092701

文档撰写日期：2021年12月28日（星期二）

前言：昨晚考完高数，带着不知会否及格的些许焦虑，今早考了另一门相对轻松的课。现在终于能在考试间隙，在寒假前，最后再体会一遍 **舒适的计概B**，这次是真正的最后一份作业了。内容包括三大部分：**期末考试经历分享**，**期末考试7道题（重做并已于cs101题库AC）**，以及我提供的**让老师收录于学习方法合集里的总结文**。

终于还是意外了 · AC5

那天午睡醒来接近2点，感觉神清气爽。早早设置好环境，我好像是第一个在 Teams 上线的。我习惯早做准备，如此可以趁早避开意外。临考前不慌不忙，还在群里帮了一位留学生老乡@黄靖扬，考题释出前又摸了会儿魔方，几次模拟考都没那么冷静过。

没有想太多，直接点进第一题（矩阵），看了看发现连模板来源都备好了，老师真是贴心。后就先犯了**读题不细的低级错误**，没注意到输出的格式，发现样例不对，浪费好一些时间把对的改成错的又改回来。接着才发现这是个D，不妨放一放，看看E。双十一，居然 WA 了，同样是**读题不细，每满200扣30 我看成 满200扣30**。当时傻傻的我并没有发现，没关系，稳住心态先去看小朋友。马上注意到缺了 12，马上判断样例错误，提交就马上 WA，完了，马上以为错的是我。于是开始研究样例二，一番头脑风暴与折腾后，发现样例变了，老师提醒样例有误。好吧那我再提交，还 WA？**AC0既视感，开始冒冷汗**。

好吧，或许M题容易些，看了看觉得老师简直太好了，在提示里道尽一切。然后平静了一阵，回去看双十一，发现是自己眼残了，于是接近4点才斩获第一个AC。**小朋友呢？**还是没算明白。这题描述写得妙，“**你很着急**”，简简单单四个字真是身临其境，贯穿了我的整场考试。当时真是脑袋发烟了都想不明白怎么回事，于是先快速地解决了两道M题，注意到矩阵题的输出格式后也对了。反复研究小朋友之后觉得我应该推倒重来，可惜还是 WA。于是去看 DLS，经过一小阵的尝试与一次 WA，在考试结束前不久 AC。

可是小朋友呢？停留在那四个字：“**你很着急**”（对，D和M加一起都不如E的一群春游的小朋友折腾我）。到了最后，听见老师也注意到我“出意外”了，我只能自顾遗憾，心里还有一丝愧疚。事后群里果然炸开了锅，看见也有几位和我一样的情况，我就征求一个AC代码。第一个看见的就是@黄靖扬同学，没明白我错在哪，但发现他应该取用了我考前分享的模板，果然互帮互助终究有回报的。于是最后一次小考告一段落，在群里的喧嚣中我没吱什么声，只是赶紧去备考当晚的地概。之后梳理群里的信息和代码就注意到问题了，**我没考虑重复的编号，我大意了**。

事后反省，虽然题目没说清楚，但我应该想到的。切身体会到出题不易，描述不好写，但又要考一考学生，考前又不能多做测试，出题确实比解题难啊。我个人觉得不妨把这种特殊情况放在样例让考生研究，可又好像太过明显，不知道有没有两全其美的办法。**考一次试不仅能难倒同学们，还能难倒老师和助教，真是辛苦大家了。**

教训：细读题，别着急。不要对自己太自信，也不要对题目太相信。

一个小建议：不妨这样设置考试题号：x1E、x2E、x3M、x4M、x5D、x6D，这样顺序就对了。（我观察到这题号首字符好像和 python 的变量名一样不能是数字，不知道这样行得通吗，供老师参考）

23554:小朋友春游 (implementation, sort)

<http://cs101.openjudge.cn/practice/23554/>

解题思路：简单题简单做，不要担心超时不要担心内存，乖乖实现 **implementation** 即可，两个方向：

排序解法: 题目要求升序输出, 接收时就确保数据是升序即可, 应用 `sorted` 函数。变量: `kids` 是剩余的小朋友, `lost` 是走丢的, `more` 是乱入的(多出来的), 这里用 **list comprehension 简化代码**。

```
1 n = int(input())
2 kids = sorted(map(int, input().split()))
3 lost = [i for i in range(1, n+1) if i not in kids]
4 more = [j for j in kids if j > n+1]
5 print(*lost)
6 print(*more)
```

暴力解法: 提示说道可以不必排序, 这点确实可以, 因为数据范围不大, 暴力遍历即可。关键的地方是怎么处理重复的编号, 我采用了 `count`, 就是一个暴力的思想。对, 我们不能忘记 **brute force 也算是一种算法**。

```
1 n = int(input())
2 *kids, = map(int, input().split())
3 lost, more = [], []
4 for i in range(1, n+1):
5     if i not in kids:
6         lost.append(i)
7 for j in range(n+2, 10001):
8     if j in kids:
9         k = kids.count(j)
10        more.extend([j]*k)
11 print(*lost)
12 print(*more)
```

注: 我没研究如何尽可能少使用额外空间, 猜测是删除元素?

考试(WA)代码之一: 凡是有可能出现重复元素的, 不要用 `set`。而且当时还以为用解包号输出有问题, 就多此一举地用了 `join` 和 `map`。唉, 属实是急坏了。

```
1 n = int(input())      # WA code
2 kids = set(map(int, input().split()))
3 mine = {i for i in range(1, n+1)}
4 lost = sorted(k for k in mine - kids if k <= n)
5 more = sorted(a for a in kids - mine)
6 print(' '.join(map(str, lost)))
7 print(' '.join(map(str, more)))
```

考试(WA)代码之二: 就差一点点, 就是还没考虑到重复元素。(以及第10行应该是 $k > n+1$)

```
1 n = int(input())      # WA code
2 kids = [*map(int, input().split())]
3 lost, more = [], []
4 for i in range(1, n+1):
5     if i not in kids: lost.append(i)
6 kids.reverse()
7 for k in kids:
8     if k > n: more.append(k)
9 more.reverse()
10 print(' '.join(map(str, lost)))
11 print(' '.join(map(str, more)))
```

23566:决战双十一 (implementation)

<http://cs101.openjudge.cn/practice/23566/>

解题思路: 名副其实的 implementation, 变量 **total** 累计金额。两个字典: **shop** 键=店铺, 值=商品, **sale** 键=最低价, 值=减价。用了三个for循环, 感觉一定还可以简化, 但没想到具体怎么做。值得注意的是, 若把第9行的处理后置就会 WA, 可能是可证明的 greedy? 或者纯属测试数据巧合? 不清楚, 不细究了。

```
1 n, m = map(int, input().split())
2 total = 0
3 shop, sale = {}, {}
4 for _ in range(n):
5     s, p = map(int, input().split())
6     try: shop[s].append(p)
7     except: shop[s] = [p]
8     total += p
9 total -= 30*(total//200)
10 for i in range(1, m+1):
11     q, x = map(int, input().split('-'))
12     sale[i] = (q, x)
13 for s in shop:
14     if sum(shop[s]) >= sale[s][0]:
15         total -= sale[s][1]
16 print(total)
```

注: 上述代码与考试AC代码并无太大区别, 考试时着急就把第9行分成两行来写了, 实际上没必要。

23564:数论 (number theory, math)

<http://cs101.openjudge.cn/practice/23564/>

解题思路: 提示给了函数, 这题就容易多了。**直接取用函数** (自己调对缩进后), **通过集合比较是否有重复元素**, 若无重复元素则还需 **判断质因数个数的奇偶性**, **取模** 就好。最后用了 comprehension, nice python style。

```
1 def pFactors(n):
2     pFact, limit, check, num = [], int(n**0.5) +1, 2, n
3     for check in range(2, limit):
4         while num % check == 0:
5             pFact.append(check)
6             num /= check
7         if num > 1: pFact.append(num)
8     return pFact
9 memo = pFactors(int(input()))
10 x, y = len(memo), len(set(memo))
11 print(0 if x-y else [1,-1][x%2])
```

考试 (AC) 代码: 粗糙, 急躁, 无脑复制粘贴, 排版不佳, 还有一堆乱七八糟多余的东西, 无论如何还是过了。

```
1 def pFactors(n):
2     """Finds the prime factors of 'n'"""
3     from math import sqrt
4     pFact, limit, check, num = [], int(sqrt(n)) + 1, 2, n
5     for check in range(2, limit):
6         while num % check == 0:
7             pFact.append(check)
8             num /= check
9         if num > 1:
10            pFact.append(num)
11    return pFact
12 n = int(input())
13 if n == 1: print(1)
14 else:
15     memo = pFactors(n)
16     x = len(memo)
17     if len(memo) != len(set(memo)): print(0)
18     else:
19         print([1, -1][x%2])
```

23556:小青蛙跳荷叶 (dp)

<http://cs101.openjudge.cn/practice/23556/>

解题思路: 小青蛙看着面善, 一路读到提示才发现, 原来是和斐波那契的兔子一样。第一片荷叶对应斐波那契数列的第二项, 在 0-based index 里刚好就是索引值了, 做一个前缀和取最后一项 (索引n) 就行。

```
1 n = int(input())
2 dp = [1, 1] + [0] * (n-1)
3 for i in range(2, n+1):
4     dp[i] = dp[i-1] + dp[i-2]
5 print(dp[n])      # same with dp[-1]
```

考试 (AC) 代码: 也没时间多想, 直接复制粘贴之前做过的题, 真是感谢当时的自己写了个函数, 方便地过了。其实斐波那契数列是我的dp启蒙, 当初花了好些时间消化理解, 现在回头看才知道自己成长了那么多。

```
1 def fib(n):
2     if n<3: return 1
3     memo = [1, 1] + [0] * (n-2)
4     for i in range(2, n):
5         memo[i] = memo[i-1] + memo[i-2]
6     return memo[n-1]
7 print(fib(int(input())+1))
```

23555:节省存储的矩阵乘法 (implementation, matrices)

<http://cs101.openjudge.cn/practice/23555/>

解题思路：这题比较需要处理的是输入和输出方式的特殊性，其实提前备好几个全为0的矩阵就方便了。就是按部就班做下去，这题也不会太难。代码中各变量名基本遵循题面描述。考试 AC 代码与除了多一行没意义的空格外与此完全相同，本来还想空格比较好看，后来觉得还是算了吧。

```
1 n, m1, m2 = map(int, input().split())
2 mtrx_1 = [[0]*n for _ in range(n)]
3 mtrx_2 = [[0]*n for _ in range(n)]
4 for _ in range(m1):
5     r, c, v = map(int, input().split())
6     mtrx_1[r][c] = v
7 for _ in range(m2):
8     r, c, v = map(int, input().split())
9     mtrx_2[r][c] = v
10 ans = [[0]*n for _ in range(n)]
11 for i in range(n):
12     for j in range(n):
13         for k in range(n):
14             ans[i][j] += mtrx_1[i][k] * mtrx_2[k][j]
15 for r in range(n):
16     for c in range(n):
17         if ans[r][c]: print(r, c, ans[r][c])
```

23558:有界的深度优先搜索 (dfs)

<http://cs101.openjudge.cn/practice/23558/>

解题思路：看了一会儿，发现这不就是 Kefa 的图 + 马走日的步数 吗？结合两者概念从模板中提取相关代码拼装，第一次WA。调整几个细节，尤其是 sort，就AC了。说什么思路其实也没有，就是善用模板，套用模板来源的两题思路融合，加上题目要求 尽可能升序 所以为子节点做排序，把最终的 seen 解开输出就完事了。

```
1 n, m, L = map(int, input().split())
2 def dls_rec(tree:dict, node:str, seen:list, dep:int):
3     if dep > L: return seen
4     if node not in seen:
5         seen.append(node)
6         if node in tree:
7             for near in tree[node]:
8                 dls_rec(tree, near, seen, dep+1)
9     return seen
10 tree = {}
11 for _ in range(m):
12     a, b = map(int, input().split())
13     try:
14         tree[a].append(b)
15         tree[a].sort()
16     except: tree[a] = [b]
17     try:
18         tree[b].append(a)
19         tree[b].sort()
20     except: tree[b] = [a]
21 start = int(input())
22 print(*dls_rec(tree, start, [], 0))
```

注：考试 AC 代码与此无异，只是当时没心思改变量名，以上是改好的版本。当时时间紧迫我就在 leetcode 上直接改，AC 之后甚至忘了拷贝下来保存（使劲去算小朋友了），还好在考试的提交那里还能找回来。

23568: 幸福的寒假生活 (dp)

<http://cs101.openjudge.cn/practice/23568/>

解题思路：三件事要办：**映射日期、数据预处理、dp转换方程**。日期由自定义的 **conv** (convert指转换) 函数解决；预处理是以结束时间排序(形同充实寒假)，用 **sort** 解决；至于 **dp**，我只知道要取 **不只一次的 max**，思维框架是有的，代码实现是无了，一直连样例都过不去，甚至还会 RE。为了保留时间给之后的考试和论文，我直接去参考了期末考当天 **丁一尘同学在群里发的代码**，感叹我和8班同学的等级还是差得多。研究了一阵子我理解了，但考试考这个我大概率就卡在第16行了... 很妙的是通过 '**+1**' 及 **dp数组开到 46** 就可以避开边界会越界的问题。

```
1 # refer to code of classmate Ding-Yichen
2 def conv(date:str):
3     m, d = map(int, date.split('.'))
4     if m == 1: return d-7
5     else: return 24+d
6 plan = []
7 for _ in range(int(input())):
8     start, end, love = input().split()
9     s, e = conv(start), conv(end)
10    if e > 44: continue
11    plan.append([s, e, int(love)])
12 plan.sort(key = lambda x:x[1])
13 dp = [0]*46
14 for p in plan:
15     s, e, love = p
16     dp[e+1] = max(dp[e+1], max(dp[:s+1])+love)
17 print(max(dp))
```

注：透过这题我明白自己只是靠着一个留学生的 bonus 才有 AC5 的，把我摆到8班里面可能也就中上的程度。一考完就看见大家说这题有坑，很庆幸自己不必在考试时做这题，但考试后还是可以做一做的，当作学习了。

学习方法总结

我大略看了同学们写的内容，排除重复性，有一些事是我作为 **零基础+留学生** 可以分享的：

学习需要好的打开方式。 最初填问卷的时候我好像说我是“几乎零基础”，因为自学过一点html皮毛。Markdown 和 Programming 还是有大差别的，但如果学习编程之前能对电脑熟悉，帮助是肯定有的。我业余爱好打游戏，入学后直接成了心底一道白月光，可能寒假期间才能放心去玩了。幸运的是我学编程的过程中体会到了类似的乐趣。很久之前我就接触过编程相关的游戏，或者从我初识电脑时兴趣就已经产生了吧，**感性认识是一切学习的开端。**

推荐一个挺有意思的小游戏，不知境内能否访问此网址：<https://www.w3schools.com/codegame/index.html>

在班里千万别压力或焦虑。 不管你是零基础还是学过一点，总有一些竞赛生的水平是不可企及的。群里热闹讨论，有人觉得容易有人觉得难，我们一定要从中 **萃取精华**，使自己成长。也别妄想能把一切都学到，我们必须做取舍，掌握一些自己会用的，好用的技巧就行了。在 **整体** 掌握上，要多和自己比较，别和同学们比较，这是建立信心与学习节奏；在代码 **细节** 上，要和不同时期的自己比较，更要和同学们比较，这是提升编程水平的好途径。所以凡是从群里感到一点点的不安，都一定可以转化成进步的养分。

模板思维很重要。 我见过那种用一些模块拼凑的编程方式（不知该怎么形容，类似上面那个小游戏吧），比起经典的编程方式是更形象的，我个人以为那会是编程普及化的重要途径。那就是一套一套的模板，学会套用并在其基础上开发新的模板，也算是学习编程的精髓之一。**从学期初就开始准备属于自己的模板，随课程进展不断修订补充，这是绝对有帮助的。** 若你对整理模板和代码没有概念，我留了 True_Code_Pack_yeagle_ting2021fall.zip 给老师，也是受之前几位比我更厉害的学长姐启发，可以参考其中的资料整理方式。

（下述内容修订自 True_Final_Sum陈勇2021fall.pdf 的总结部分，这个 pdf 也是我留给老师的一份大笔记）

我对编程的兴趣是浓厚的，这也是我那么认真的最大原因。兴趣使然，凌驾于一切学习理由。我不会尝试在此说服学弟妹们编程为什么值得我们爱好，仅提供几个可能有指导作用的方法论作为学习方式的参考：

1. **时间精力分妥当。** 曾老师再次强调，课下投入是必须的，请不要担心付出没回报，更不要盲目对着难题耗。
2. **双向交流都要有。** 请教与赐教都别吝啬，老师助教同学都是交流的对象，不一定多交流，但肯定要交流。
3. **看题解与善搜索。** 题解自有黄金屋，一定要看；网上资讯无穷多，应该瞧瞧。时间点见仁见智，有就行。
4. **先理解而后预测。** 归纳现象并分析规律，知道怎么回事了，下次就能警觉，和所谓的科学探究方法一样。
5. **试模仿能学创造。** 凭空想象不易，尝试过认真模仿现成代码会明白过程中的思考，那是自我提升的好途径。

最后说说学术以外的：作为线上的同学，我是按教务推荐选的课，也只有一班线上班的样子。我想我是极幸运的，**老师对线上学习同学和留学生都很友好**，所以我才用“舒适”形容这门课。希望学弟妹们不必再为线上选课烦恼，若不幸还是需要，那么闫老师的课是一个非常好的选择。再次感谢闫老师对我的赏识与抬举，很大程度上支持我在高压的学习环境中保持积极的心态，**我相信老师对每位选课同学的欣赏也是优秀率那么高的关键之一**。本来还想问数据结构与算法的课能让线上同学选课吗，后来知道老师不会开课（好可惜+1）。我想我还会再找机会继续学习编程及cs相关科目，兴趣所在，希望能有更多收获。

for days in the_future:

```
will_learn_more_programming = True
```

for appreciation in class_GMyhf:

```
give_back = True_Final_Sum陈勇2021fall.pdf + True_Code_Pack_yeagle_ting2021fall.zip
```

总结 (True Final Sum)

True_Code_Pack_yeagle_ting2021fall.zip 有一个 EXAM 文件夹，内容是笔者整理的一些模板 (xxx_template.py)，以及用于装载期末考 7 题代码的 .py 档，还有一个是跑测试用的程序 .test.py。这份文档未能完成的，详细且针对代码及算法的笔记，就以 template (模板) 的形式提供予各位。详情请参考 zip 中的 READ_ME.txt。

或许读者已经发现，笔者对编程的兴趣是浓厚的，这可能也是这份文档能够成型的最大原因。兴趣使然，凌驾于一切学习理由。笔者不会尝试在此说服读者编程为什么值得我们爱好，仅提供几个可能有指导作用的方法论作为学习方式的参考：

1. **时间精力分妥当。** 替老师再次强调，课下投入是必须的，请不要担心付出没回报，更不要盲目对着难题耗。
2. **双向交流都要有。** 请教与赐教都别吝啬，老师助教同学都是交流的对象，不一定多交流，但肯定要交流。
3. **看题解与善搜索。** 题解自有黄金屋，一定要看的；网上资讯无穷多，应该瞧一眼。时间点见仁见智，只建议两件事要做。
4. **先理解而后预测。** 归纳现象并分析规律，知道是怎么回事了，下次就能够警觉，和所谓的科学探究方法一个意思。
5. **试模仿能学创造。** 凭空想象不易，尝试过认真模仿现成代码的人会明白过程中的思考，那也是自我提升的关键途径。

祝愿读者学习编程之路充满收获，True_Final_Sum 陈勇 2021fall 完。

陈勇

2021fall-cs101 || 13 班线上学生

马来西亚留学生 || 环境科学与工程学院 21 级本科生

2021 年 12 月 28 日 (星期二)