Assignment #5: 20251009 cs101 Mock Exam寒露第二天

Updated 1651 GMT+8 Oct 9, 2025

2025 fall, Complied by **韩旭 元培学院**

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

1. 题目

E29895: 分解因数

implementation, http://cs101.openjudge.cn/practice/29895/

用时：10min （15:20AC）

思路

这道题最早考虑的是直接从n/2到sqrt(n)筛选最大因数，发现超时，意识到每个大因数都对应一个小因数，并且随着小因数的增加，大因数的减小明显更快，所以按小因数来筛选是更有效率的方法。从2到sqrt(n)筛选最小的因数，然后输出对应的最大因数即可。

代码

```python
import math

n=int(input())
low=int(math.sqrt(n))
for i in range(low):
    if n%(i+2)==0:
        print(int(n/(i+2)))
        break
```

代码运行截图

E29940: 机器猫斗恶龙

greedy, http://cs101.openjudge.cn/practice/29940/

用时：5min（15:25AC）

思路

假设初始血量为0，然后按顺序闯关，记录过程中血量最低达到过多少，这个负值的绝对值加一（因为血量最低不能为0）即为所至少需要的血量。

代码

```python
n=int(input())
L=list(map(int,input().split()))
ip=0
min_ip=0
for i in L:
    ip+=i
    min_ip=min(ip,min_ip)
print(abs(min_ip)+1)
```

代码运行截图

#50273666提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
n=int(input())
L=list(map(int,input().split()))
ip=0
min_ip=0
for i in L:
    ip+=i
    min_ip=min(ip,min_ip)
print(abs(min_ip)+1)
```

基本信息

#: 50273666
题目: E29940
提交人: 22n2200017737
内存: 14804kB
时间: 63ms
语言: Python3
提交时间: 2025-10-09 15:25:28

English 帮助 关于

M29917: 牛顿迭代法

implementation, http://cs101.openjudge.cn/practice/29917/

用时：45min（16:44AC）

思路

这道题逻辑并不复杂，但需要的知识点比较多。

（1）一开始我有点忘了牛顿迭代法是怎么回事了，主要是没想清楚公式里面的$f(x)$是什么意思。后来做完第四、五题之后回来想起来牛顿迭代法是求$f(x)=0$的根的方法，于是要不使用sqrt()函数构造出一个类似的$f(x)$，就想到了$f(x)=x^2-n$，于是迭代过程就解决了。

（2）这里没有看清楚输入的n可能是一个整数或一个小数，而把n用int函数处理了，导致RE（因为n可能为0无法收敛）。

（3）由于不知道如何处理不定行输入，花费了比较多的时间。这里总结一下不定行输入的三种常见处理：

- 使用 `try...except` 捕获输入结束（如 EOFError，End-of-file error，）

```
#例子1
lines = []
try:
    while True:
        line = input()
        lines.append(line)
except EOFError:
    pass

print(lines)
```

- 利用 `sys.stdin` 逐行读取，`line.strip()` 函数可以去除字符串开头和结尾的空白字符，比如回车（`\n`）。

```
#例子2
import sys

for line in sys.stdin:
    print(line.strip())
```

- 使用 `sys.stdin.read()` 一次性读取所有输入

```
#例子3
import sys

data = sys.stdin.read()
lines = data.strip().split('\n')
print(lines)
```

代码

```
import sys
for line in sys.stdin:
    if not line.strip():
        continue
    n=float(line.strip())
    old=0
    new=1
    count=0
    while abs(new-old)>1e-6:
        old=new
        new=old-(old**2-n)/(2*old)
        count+=1
    print(count,f"{new:.2f}",sep=" ")
```

代码运行截图

#50276234提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys
for line in sys.stdin:
    if not line.strip():
        continue
    n=float(line.strip())
    old=0
    new=1
    count=0
    while abs(new-old)>1e-6:
        old=new
        new=old-(old**2-n)/(2*old)
        count+=1
    print(count,f"{new:.2f}",sep=" ")
```

基本信息

#: 50276234
题目: M29917
提交人: 22n2200017737
内存: 3544kB
时间: 19ms
语言: Python3
提交时间: 2025-10-09 16:43:48

English 帮助 关于

M29918: 求亲和数

implementation, http://cs101.openjudge.cn/practice/29918/

用时：15min（15:51AC，做完第二题后想了10min牛顿迭代法没太明白，然后就跳过来做这个题）

思路

这道题我先写了一个函数找出某个数除本身以外的因数之和，这样对于某个数i，把`find_sum_facotr(i)`的结果S代入回这个函数，检查是否仍为i，即可筛选出亲和数。这里首先要注意去重（比如a和b，b和a这种），以及因数和大于范围N 的情况。

但即使这样做，如果从1到N筛选，仍然会出现超时。这里我用了一个“歪招”，即先用写好的算法把所有100000以内的亲和数对打印出来，发现20000到60000之间并不存在亲和数，所以在筛选的时候跳过了这部分，避免了超时的问题。

代码

```
import math

def find_sum_factor(n):
    factor = [1]
    upper = int(math.sqrt(n)) + 1
    for i in range(upper):
        if i<=1:
            continue
        if n%i==0:
            factor.append(i)
            if int(n/i)!=i:
                factor.append(int(n/i))
    return sum(factor)

N=int(input())
store=[]
for i in range(N):
    if i+1>=20000 and i+1<=60000:
        continue
    if i+1 not in store:
        S=find_sum_factor(i+1)
        if find_sum_factor(S)==i+1 and i+1!=S and S<=N:
            print(i+1,S,sep=" ")
            store.append(i+1)
            store.append(S)
```

代码运行截图

#50274382提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
import math

def find_sum_factor(n):
    factor = [1]
    upper = int(math.sqrt(n)) + 1
    for i in range(upper):
        if i<=1:
            continue
        if n%i==0:
            factor.append(i)
            if int(n/i)!=i:
                factor.append(int(n/i))
    return sum(factor)

N=int(input())
store=[]
for i in range(N):
    if i+1>=20000 and i+1<=60000:
        continue
    if i+1 not in store:
        S=find_sum_factor(i+1)
        if find_sum_factor(S)==i+1 and i+1!=S and S<=N:
            print(i+1,S,sep=" ")
            store.append(i+1)
            store.append(S)
```

基本信息

#: 50274382
题目: M29918
提交人: 22n2200017737
内存: 3648kB
时间: 1571ms
语言: Python3
提交时间: 2025-10-09 15:51:00

M29949: 贪婪的哥布林

greedy, http://cs101.openjudge.cn/practice/29949/

用时：20min（16:11AC）

思路

这道题的思路是经典的贪心，但需要排序的是矿石的单位价值，因为在总重量一定的情况下，优先拿的应该是单位价值最大的矿石。计算单位价值的时候由于涉及到除法，如果再乘以w可能无法得到原来的v，所以我用了字典保存单位价值对应的重量和总价值。

再总结一下f-string的用法：基本用法是在{ }内写出任意的表达式，比如`print(f"My name is {name}, and I am {age} years old.")`。我们可以在{ }内以冒号分隔使用格式说明符，例如：

```
pi = 3.1415926
print(f"Pi ≈ {pi:.2f}")      # 保留两位小数
print(f"Pi ≈ {pi:10.3f}")    # 宽度10，保留3位小数
print(f"{1000:,}")           # 千位分隔符
```

代码

```python
N,M=map(int,input().split())
L=[]
weight_dict={}
value_dict={}
for i in range(N):
    v,w=map(int,input().split())
    vpw=v/w
    a=weight_dict.get(vpw,0)
    b=value_dict.get(vpw,0)
    weight_dict[vpw]=a+w
    value_dict[vpw]=b+v
    L.append(vpw)
L.sort(reverse=True)
index=0
total_value=0
total_weight=0
while total_weight<M and index<N:
    if total_weight+weight_dict[L[index]]<=M:
        total_value += value_dict[L[index]]
        total_weight+=weight_dict[L[index]]
    else:
        total_value += L[index]*(M-total_weight)
        break
    index+=1
print(f"{total_value:.2f}")
```

代码运行截图

#50274992提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```python
N,M=map(int,input().split())
L=[]
weight_dict={}
value_dict={}
for i in range(N):
    v,w=map(int,input().split())
    vpw=v/w
    a=weight_dict.get(vpw,0)
    b=value_dict.get(vpw,0)
    weight_dict[vpw]=a+w
    value_dict[vpw]=b+v
    L.append(vpw)
L.sort(reverse=True)
index=0
total_value=0
total_weight=0
while total_weight<M and index<N:
    if total_weight+weight_dict[L[index]]<=M:
        total_value += value_dict[L[index]]
        total_weight+=weight_dict[L[index]]
    else:
        total_value += L[index]*(M-total_weight)
        break
    index+=1
print(f"{total_value:.2f}")
```

基本信息

#:	50274992
题目:	M29949
提交人:	22n2200017737
内存:	3648kB
时间:	20ms
语言:	Python3
提交时间:	2025-10-09 16:11:04

T29947:校门外的树又来了（选做）

http://cs101.openjudge.cn/practice/29947/

用时：30min

思路

最开始依然使用list标记的方法，发现空间过大，想了一些简单的节省空间的方法并不奏效，于是考虑区间合并，但考试的时候时间已经来不及做了，回来重新做比较快就通过了。主要方法是区间合并，即先按数组的起点排序，并且标记此前最大的终点，计算总区间长度。如果后面的起点比最大终点大，则加上完整的区间长度；否则加上多出来的区间长度。按区间起点排序的好处在于，新的起点和此前最大重点之间的树肯定都被砍掉过了。需要注意的一个细节是flag标记要从-1开始。

代码

```
L,M=map(int,input().split())
starts=[]
ends={}
flag=-1
ans=0
for i in range(M):
    start,end=map(int,input().split())
    starts.append(start)
    tmp=ends.get(start,-1)
    ends[start]=max(end,tmp)
starts.sort()
for start in starts:
    if start<=flag:
        ans+=max(0,ends[start]-flag)
    else:
        ans+=ends[start]-start+1
    flag=max(ends[start],flag)
print(L+1-ans)
```

代码运行截图

#50347305提交状态

查看　提交　统计　提问

状态：Accepted

源代码

```
L,M=map(int,input().split())
starts=[]
ends={}
flag=-1
ans=0
for i in range(M):
    start,end=map(int,input().split())
    starts.append(start)
    tmp=ends.get(start,-1)
    ends[start]=max(end,tmp)
starts.sort()
for start in starts:
    if start<=flag:
        ans+=max(0,ends[start]-flag)
    else:
        ans+=ends[start]-start+1
    flag=max(ends[start],flag)
print(L+1-ans)
```

基本信息

#：50347305
题目：29947
提交人：22n2200017737
内存：3640kB
时间：21ms
语言：Python3
提交时间：2025-10-13 21:49:26

English　帮助　关于

2. 学习总结和收获

这次考试题目虽然总体难度不高，但暴露出了我在知识点和编程思维细节上的一些问题。

- **输入输出处理的经验积累：**
  在做牛顿迭代法时，我发现我对于输入输出语法的掌握还有所欠缺。于是我重新整理了 Python 处理不定行输入的三种方法：`try...except` 捕获 EOF、`sys.stdin` 逐行读取、`sys.stdin.read()` 一次性读取，并且学习了标准输入流的工作原理。同时，在输出方面重新复习了 `f-string` 的多种用法。
- **算法理解与实现的平衡：**
  比如牛顿迭代法中，从忘记原理到重新推理出公式用法，让我再次体会了算法不是记忆，而是推理的结果。而机器猫和哥布林两题则是对贪心思想的自然运用，这两个题我做得比较顺利，说明在这部分算法的原理上没有特别大的问题。
- **区间排序与空间优化：**
  校门外的树是之前作业里面的原题，所以我优先使用了之前掌握的方法，但是空间优化的要求又进一步提升了算法的效率，这提醒我在完成作业的时候除了AC也需要考虑到能否进一步优化的问题，尤其是解法中出现了大数组或者大量循环的时候。区间合并思路刚想到的时候会感觉比较复杂，这时候借助指针能够比较好地梳理清楚逻辑，并且在debug的时候总结出两个关键细节：　(1) `flag` 初值应为 `-1`，以避免首区间漏算；　(2) 每次更新 `flag` 时用 `max(flag, ends[start])`，防止倒退覆盖范围。这些边界和细节的修改，体现了对算法边界条件和逻辑正确性的理解。
- **时间优化**:
  在做求亲和数的时候，也遇到了超时的问题。虽然最终用"预观察+跳过无亲和数区间"的取巧方式绕过去，但AI建议标准的做法是类似**埃拉托斯特尼筛**，把每个因数（小于等于`N//2`的）直接加到其倍数上，从而得到一个所有数对应的因数和的列表（复杂度大概是$O(NlogN)$，如果每次都调用函数复杂度则会达到$O(N^{\frac{3}{2}})$。使用类似埃拉托斯特尼筛的保存方式就避免了每次遇到一个数都要重新遍历计算，使得时间复杂度超时的问题。如果缓存因数和函数的结果，可能也可以达到类似的节省时间的效果。