Assignment #5: Greedy穷举 Implementation

Updated 1939 GMT+8 Oct 21, 2024

2024 fall, Complied by 胡杨 元培学院

说明：

1）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

3）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

4）如果不能在截止前提交作业，请写明原因。

1. 题目

04148: 生理周期

brute force, http://cs101.openjudge.cn/practice/04148

思路：假设到三个生理周期重合p，e，i分别需要经过x,y,z天，先考虑p,e两个的周期何时重叠，找到p+23x=e+28y的时间，再看此时是不是和i的周期重叠，如果不是，则继续考虑p,e的下一个重叠的周期（直接x+=28,y+=23），直到找到该日期

以及刚刚学了一下双指针的思想，尝试了一下好像确实耗时很短……！

大致用时：做题四五十分钟，改bug迷惑一晚上（细节好容易错啊）

题解启示：我的做法相当于是从p，e，i的当前天数往后加周期看哪一天可以重合。有两个题解的思路相当于先从d往后定一个天数再看它是不是周期重合的那一天。感觉题解的想法更简单好想！（但是当时确实只想到了从前往后找）

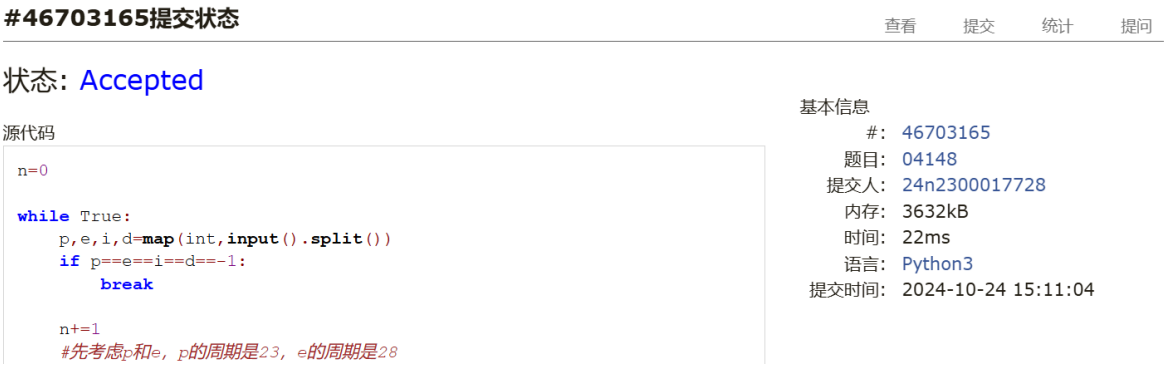
还可以直接遍历1~21252，思路更简单了

代码：

```
n=0

while True:
    p,e,i,d=map(int,input().split())
    if p==e==i==d==-1:
        break

    n+=1
    #先考虑p和e，p的周期是23，e的周期是28
    x=0
    y=0
```

```
    while p+23*x<=21252+d and e+28*y<=21252+d:
        if p+23*x<e+28*y:
            x+=1
        elif p+23*x>e+28*y:
            y+=1
        else: #此时p和e的周期重合，考虑i的情况，其周期是33
            if (p+23*x-i)%33!=0:
                x+=28
                y+=23
                continue #此时i的周期不重合，则继续考虑下一个重叠的情况
            elif (p+23*x-i)%33==0: #此时三个周期都重合了！检验其是否大于d
                day=p+23*x
                if day>d:
                    print(f"Case {n}: the next triple peak occurs in {day-d}
days.")
                else:
                    print(f"Case {n}: the next triple peak occurs in {day+21252-
d} days.")
                break #退出循环
```

代码运行截图 （至少包含有"Accepted"）



18211: 军备竞赛

greedy, two pointers, http://cs101.openjudge.cn/practice/18211

思路：先从最便宜的武器开始做，等到做不起的时候就把最贵的武器卖掉。每一次卖掉武器都要保证卖掉它之后的钱至少可以再做一件便宜武器，不然就亏了。通过样例发现了两个数组越界的bug，改之。

ps.最初认为卖掉一件要至少在造两件才行，但是忽略了其实可以卖两个造三个也是赚了。所以就把追求每次卖都要收获比对手更多的思想改成了卖一个能造一个不亏就行。

大概用时：原初调试30min，WA后又改了30min

题解启示：排序过后卖一个贵的肯定可以造一个便宜的啊！！根本不用纠结卖了一个能不能回本的问题，双指针+有钱就造没钱就卖，保证more>=0并且在只剩一个的时候break就行了。我是傻逼……)

代码：

```
p=int(input())
n=list(map(int,input().split()))
```

```
n.sort()
more=0 #初始化最初武器差值

#先把一件武器都做不起的情况解决
if p<n[0]:
    print(0)

#接下来考虑可以开始做武器-卖武器循环的情况
elif p>=n[0]:
    while more>=0:
        if len(n)==0:#当武器用光的时候，退出循环
            break
        else:
            if p>=n[0]:
                p-=n[0]
                more+=1
                if len(n)==1:
                    break
                else:
                    n=n[1:]
            else: #当造不起新武器的时候，需要考虑卖掉图纸后的钱能不能至少造1件武器
                #为了排除下面的表达式p+n[-1]>=n[0]把一张图纸又卖又造的问题，当列表元素个数
<=1时，直接break
                if len(n)<=1:
                    break

                else:
                    if p+n[-1]>=n[0]:
                        p += n[-1]
                        more -= 1
                        n = n[:-1]
                    else:
                        break

    print(more)
```

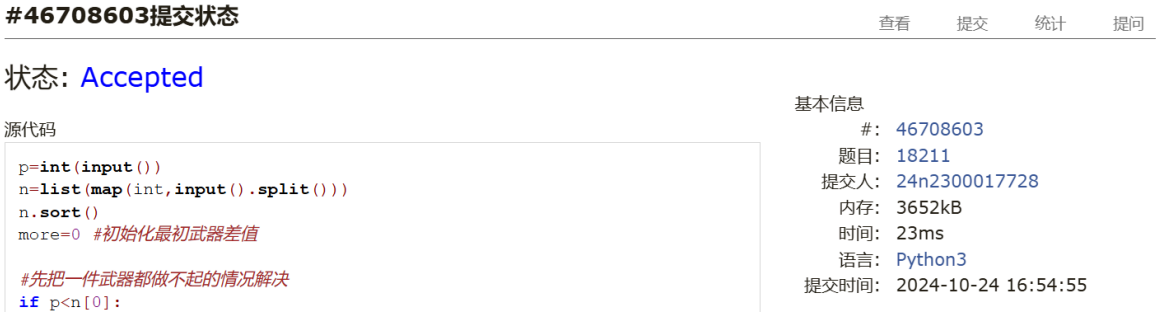
发现我是傻逼过后的代码重制版：

```
p=int(input())
n=list(map(int,input().split()))
n.sort()
more=0 #初始化最初武器差值

if p<n[0]:
    print(0)

else:
    while more>=0:
        if len(n)==1:
            if p>=n[0]:
                more+=1
                break
            else:
```

```
                break
        else:
            if p>=n[0]:
                p-=n[0]
                more+=1
                n=n[1:]
            else:
                p+=n[-1]
                more-=1
                n=n[:-1]

    print(more)
```

试图尽量保持原状，但是发现还是**不要一边遍历一遍改列表，设置双指针从两边遍历虽然思想一致但是可以避免数据越界等等困扰，也更好表达。**

代码运行截图 （至少包含有"Accepted"）



21554: 排队做实验

greedy, http://cs101.openjudge.cn/practice/21554

思路：要让平均等待时间最短，即让做的最快的同学先做实验，做的慢的后做。怎么保留两位小数先去查一查（查一查）。

大致用时：加上查资料1h

本题思路简单但是用到了很多不会的语法，总结如下：

启示

（一）保留x位小数

1.使用 `round()` 函数

`round()` 函数可以用来四舍五入到指定的小数位数。

```
number = 3.141592653589793
rounded_number = round(number, 2)
print(rounded_number)  # 输出 3.14
```

需要注意的是，`round()` 是根据四舍五入规则进行的，如果第三位小数是 5 或以上，则会进一位。

2.使字符串格式化

你可以使用字符串格式化方法来显示浮点数为保留两位小数的字符串形式。

使用 `format()` 方法

```
number = 3.141592653589793
formatted_number = "{:.2f}".format(number)
print(formatted_number)  # 输出 "3.14"
```

使用 f-string

```
number = 3.141592653589793
formatted_number = f"{number:.2f}"
print(formatted_number)  # 输出 "3.14"
```

3. 使用 `Decimal` 类型（精确计算）

如果你需要更精确地控制数字，并且避免浮点数运算带来的精度问题，可以使用 `decimal` 模块中的 `Decimal` 类型。

```
from decimal import Decimal, getcontext

# 设置全局精度
getcontext().prec = 4  # 总共四位，包括整数部分和小数部分

number = Decimal('3.141592653589793')
rounded_number = round(number, 2)
print(rounded_number)  # 输出 Decimal('3.14')
```

（二）字典推导式创建字典

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
my_dict = {k: v for k, v in zip(keys, values)}
```

（三）lambda函数

在Python中，`lambda` 关键字用于创建匿名函数，即没有名字的函数。`lambda` 函数可以**接受任意多个参数，但是只能有一个表达式，不能包含复杂的逻辑或多条语句。**`lambda` 函数通常用于需要一个简单函数的地方，比如排序或过滤操作。

基本语法

`lambda` 函数的基本语法如下：

```
lambda 参数1, 参数2, ... : 表达式
```

这里，`参数1, 参数2, ...` 是 `lambda` 函数的输入参数，而 `表达式` 是使用这些参数计算的结果，该结果会作为 `lambda` 函数的返回值。

也**可以没有形参，直接跟表达式**

示例

1.简单的加法:

```
add = lambda x, y: x + y
print(add(5, 3))  # 输出 8
```

2.排序列表中的字典: 如果你有一个由字典组成的列表，并且想要根据某个键对它们进行排序，你可以使用 `lambda` 函数来指定排序依据：

```
people = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]
sorted_people = sorted(people, key=lambda person: person['age'])
print(sorted_people)
# 输出 [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name':
'Charlie', 'age': 35}]
```

---

sorted()

`sorted()` 用于对序列进行**排序**，可以通过 `key` 参数自定义排序**规则**，通过 `reverse` 参数控制排序**方向**（**默认升序**）

基本语法

```
sorted(iterable, key=None, reverse=False)
```

- `iterable`：要排序的可迭代对象。
- `key`：一个函数，用来指定一个元素的排序关键字。
- `reverse`：一个布尔值，如果设置为 `True`，则排序结果为降序，默认为 `False`（升序）。

  示例：如按字符串长度排序

  ```
  words = ["apple", "banana", "cherry", "date"]
  sorted_words = sorted(words, key=len)
  print(sorted_words)  # 输出: ['date', 'apple', 'cherry', 'banana']
  ```

  在这个例子中，`key=len` 指定了排序的关键字为字符串的长度，因此列表中的字符串首先按照它们的长度进行了排序。

  如果我们想要降序排序，我们可以设置 `reverse=True`：

  ```
  sorted_words_desc = sorted(words, key=len, reverse=True)
  print(sorted_words_desc)  # 输出: ['banana', 'cherry', 'apple', 'date']
  ```

代码：

```python
n=int(input())
l=[i for i in range(1,n+1)]
time=list(map(int,input().split()))
time_dict={i:t for i,t in zip(l,time)} #创建编号-时间字典
time_list=sorted(time_dict.items(),key=lambda x:x[1]) #按时间排序

#输出做实验的顺序
shun_xv=[str(item[0]) for item in time_list]
print(' '.join(shun_xv))

#计算平均等待时间
waiting_time=sum(item[1]*(n-1-index) for index,item in enumerate(time_list))
avg_waiting_time=waiting_time/n
print(f'{avg_waiting_time:.2f}')
```

代码运行截图 （至少包含有"Accepted"）

01008: Maya Calendar

implementation, http://cs101.openjudge.cn/practice/01008/

思路：可以按照Haab日历计算出这一天是从0年开始的第几天。//260即为东青年的年份。其余数分别%13和%20即可得到数字和名称。

发现读取Haab日历可能用到正则表达式，现学了亿下。

大致用时：1h

启示

（一）正则表达式

正则表达式（Regular Expressions）是一种强大的文本处理工具，可以帮助你匹配、查找、替换字符串中的特定模式。

1.导入 `re` 模块

首先，你需要导入 Python 的 `re` 模块：

```
import re
```

2.基本概念

- **模式**：你要搜索的字符串模式。
- **目标字符串**：你想要在其中进行搜索的实际字符串。
- **匹配**：如果模式与目标字符串中的某部分完全一致，就称为“匹配”。

3. 常用函数

下面是一些常用的 `re` 函数及其简单示例：

3.1 `re.search()`

这个函数会在整个目标字符串中搜索第一个匹配的子串，并返回一个 `match` 对象。如果没有找到匹配，则返回 `None`。

```
import re

# 目标字符串
text = "Hello, my email is example@example.com"

# 模式
pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

# 搜索
match = re.search(pattern, text)

if match:
    print("找到了匹配:", match.group())  # 输出: 找到了匹配: example@example.com
else:
    print("没有找到匹配")
```

3.2 `re.findall()`

这个函数会找到所有匹配的子串，并返回一个列表。

```
import re

# 目标字符串
text = "Numbers: 123, 456, 789"

# 模式
pattern = r'\d+'

# 查找所有数字
matches = re.findall(pattern, text)

print("找到的所有数字:", matches)  # 输出: 找到的所有数字: ['123', '456', '789']
```

3.3 `re.sub()`

这个函数可以用来替换字符串中的匹配项。

```
import re

# 目标字符串
text = "The price is $100.00"

# 模式和替换内容
pattern = r'\$\d+\.\d{2}'
replacement = "€50.00"

# 替换
new_text = re.sub(pattern, replacement, text)

print("替换后的字符串:", new_text)  # 输出: 替换后的字符串: The price is €50.00
```

理解正则表达式确实需要一些时间，但我会尽量用更具体的例子和步骤来帮助你。我们从基础开始，一步步构建正则表达式。

4. 基本元字符

首先，了解一些基本的正则表达式元字符：

- `.`：匹配任何单个字符（除了换行符）。
- `*`：匹配前面的字符零次或多次。
- `+`：匹配前面的字符一次或多次。
- `?`：匹配前面的字符零次或一次。
- `{n}`：匹配前面的字符恰好 n 次。
- `{n,}`：匹配前面的字符至少 n 次。
- `{n,m}`：匹配前面的字符至少 n 次，但不超过 m 次。
- `^`：匹配字符串的开始。
- `$`：匹配字符串的结束。
- `[abc]`：匹配方括号内的任意一个字符。
- `[^abc]`：匹配不在方括号内的任意一个字符。
- `()`：用于分组，可以对一组字符应用量词。
- `\d`：匹配任何数字（等价于 `[0-9]`）。
- `\w`：匹配任何字母、数字或下划线（等价于 `[a-zA-Z0-9_]`）。
- `\s`：匹配任何空白字符（包括空格、制表符、换页符等）。
- `\b`：匹配单词边界。

5. 具体示例

示例 1: 匹配电话号码

假设我们要匹配格式为 `xxx-xxx-xxxx` 的电话号码。

```python
import re

# 目标字符串
text = "Contact us at 123-456-7890 or 456-789-0123"

# 模式
pattern = r'\d{3}-\d{3}-\d{4}'

# 查找所有电话号码
phone_numbers = re.findall(pattern, text)

print("找到的电话号码:", phone_numbers)  # 输出: 找到的电话号码: ['123-456-7890', '456-789-0123']
```

解释:

- `\d{3}`: 匹配三个数字。
- `-`: 匹配连字符 `-`。
- `\d{3}`: 再匹配三个数字。
- `-`: 再匹配连字符 `-`。
- `\d{4}`: 最后匹配四个数字。

示例 2: 匹配电子邮件地址

假设我们要匹配电子邮件地址。

```python
import re

# 目标字符串
text = "Hello, my email is example@example.com and another one is test@domain.co.uk"

# 模式
pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

# 查找所有电子邮件地址
emails = re.findall(pattern, text)

print("找到的电子邮件地址:", emails)  # 输出: 找到的电子邮件地址: ['example@example.com', 'test@domain.co.uk']
```

解释:

- `\b`: 匹配单词边界。
- `[A-Za-z0-9._%+-]+`: 匹配一个或多个字母、数字或特殊字符（如 `.`、`_`、`%`、`+`、`-`）。
- `@`: 匹配 `@` 符号。
- `[A-Za-z0-9.-]+`: 匹配一个或多个字母、数字、`.` 或 `-`。
- `\.`: 匹配点 `.`。
- `[A-Z|a-z]{2,}`: 匹配两个或更多字母。
- `\b`: 匹配单词边界。

示例 3: 提取日期

假设我们要从文本中提取日期（格式为 `YYYY-MM-DD`）。

```
import re

# 目标字符串
text = "Today's date is 2023-10-27 and tomorrow's date is 2023-10-28."

# 模式
pattern = r'\b\d{4}-\d{2}-\d{2}\b'

# 查找所有日期
dates = re.findall(pattern, text)

print("找到的日期:", dates)  # 输出: 找到的日期: ['2023-10-27', '2023-10-28']
```

解释：

- `\b`：匹配单词边界。
- `\d{4}`：匹配四位数字。
- `-`：匹配连字符 `-`。
- `\d{2}`：匹配两位数字。
- `-`：匹配连字符 `-`。
- `\d{2}`：匹配两位数字。
- `\b`：匹配单词边界。

6. 分步构建正则表达式

如果你需要构建一个复杂的正则表达式，可以一步一步来：

1. **确定你要匹配的内容**：明确你要从文本中提取什么信息。
2. **分解成小部分**：将整个模式分解成更小的部分。
3. **逐步构建**：从简单的部分开始，逐步添加更多的细节。
4. **测试和调整**：使用实际数据进行测试，并根据结果进行调整。

7. 小结

- 使用 `re` 模块来处理正则表达式。
- `re.search()` 用于查找第一个匹配。
- `re.findall()` 用于查找所有匹配。
- `re.sub()` 用于替换匹配的部分。
- 了解基本的正则表达式语法，如 `.`、`*`、`+`、`?`、`{n}` 等。

（二）rstrip（）

`rstrip()` 是 Python 中字符串的一个方法，用于**删除字符串末尾的指定字符。默认**情况下，它会删除字符串末尾的**所有空白**字符（包括空格、制表符 `\t`、换行符 `\n` 等）。你也可以**指定要删除的字符。**

注意，`rstrip()`方法**并不支持删除末尾指定个数的字符**!

1. 删除末尾的所有空白字符

```
text = "Hello, World!   \n\t"
stripped_text = text.rstrip()
print(repr(stripped_text))  # 输出: 'Hello, World!'
```

在这个例子中，`rstrip()` 删除了字符串末尾的**所有空白**字符（包括空格、制表符和换行符）。

2. 删除指定的字符

你可以传递一个参数给 `rstrip()` 方法，指定要删除的字符。

```
text = "Hello, World!***"
stripped_text = text.rstrip('*')
print(repr(stripped_text))  # 输出: 'Hello, World!'
```

在这个例子中，`rstrip('*')` 删除了字符串末尾的**所有**星号 `*`。

3. 删除多个指定的字符

你可以传递一个包含多个字符的字符串，`rstrip()` 会删除这些字符中的任何一个在字符串末尾出现的情况。

```
text = "Hello, World!***---"
stripped_text = text.rstrip('*-')
print(repr(stripped_text))  # 输出: 'Hello, World!'
```

在这个例子中，`rstrip('*-')` 删除了字符串末尾的**所有**星号 `*` 和破折号 `-`。

4. 处理没有指定字符的情况

如果字符串末尾没有指定的字符，`rstrip()` **不会做任何改变。**

```
text = "Hello, World!"
stripped_text = text.rstrip('*-')
print(repr(stripped_text))  # 输出: 'Hello, World!'
```

在这个例子中，因为字符串末尾没有星号 `*` 或破折号 `-`，所以 `rstrip('*-')` 没有改变字符串。

总结

- `rstrip()` 用于删除字符串末尾的指定字符。
- 默认情况下，它删除**所有空白**字符。
- 你可以通过传递一个字符串参数来**指定**要删除的字符。
- 如果字符串末尾没有指定的字符，`rstrip()` 不会改变字符串。

（三）`global`声明全局变量（不常用）

在 Python 中，`global` 关键字用于**在函数内部声明一个全局变量**。这意味着你可以在函数内部修改全局作用域中的变量。**默认**情况下，当你在函数内部赋值给一个变量时，Python 会认为这是一个**局部变量**，除非你明确使用 `global` 关键字声明它是一个全局变量。

基本用法

1. 声明和使用全局变量

```
x = 10  # 全局变量

def my_function():
    global x  # 声明 x 是全局变量
    print(f"Inside function, before change: x = {x}")
    x = 20  # 修改全局变量 x
    print(f"Inside function, after change: x = {x}")

print(f"Outside function, before call: x = {x}")
my_function()
print(f"Outside function, after call: x = {x}")
```

输出：

```
Outside function, before call: x = 10
Inside function, before change: x = 10
Inside function, after change: x = 20
Outside function, after call: x = 20
```

在这个例子中，`global x` 声明了 `x` 是一个全局变量，因此在 `my_function` 内部对 `x` 的修改会影响到全局作用域中的 `x`。

2. 不使用 `global` 关键字

如果你不使用 `global` 关键字，函数内部的 `x` 会被视为一个新的局部变量：

```
x = 10  # 全局变量

def my_function():
    print(f"Inside function, before change: x = {x}")
    x = 20  # 这里会创建一个新的局部变量 x
    print(f"Inside function, after change: x = {x}")

print(f"Outside function, before call: x = {x}")
my_function()
print(f"Outside function, after call: x = {x}")
```

输出：

```
Outside function, before call: x = 10
Traceback (most recent call last):
  File "example.py", line 9, in <module>
    my_function()
  File "example.py", line 4, in my_function
    print(f"Inside function, before change: x = {x}")
UnboundLocalError: local variable 'x' referenced before assignment
```

在这个例子中，由于没有使用 `global` 关键字，Python 认为 `x` 是一个局部变量。当尝试在赋值前访问 `x` 时，会引发 `UnboundLocalError`，因为**局部变量 `x` 在赋值之前还没有被定义。**

使用 `global` 的注意事项

1. **避免滥用**：**尽量减少对全局变量的使用**，特别是在大型项目中。过多地使用全局变量可能会导致代码难以维护和调试。
2. **命名冲突**：使用全局变量时要注意避免命名冲突。如果多个函数都使用同一个全局变量，可能会导致意外的行为。
3. **可读性**：使用全局变量可能会影响代码的可读性和模块化。尽量将数据封装在函数或类中，通过参数传递数据。

示例：全局变量和局部变量的混合使用

```python
# 定义全局变量
x = 10

def modify_global():
    global x  # 声明 x 是全局变量
    x = 20  # 修改全局变量 x
    print(f"Inside modify_global, x = {x}")

def use_local():
    x = 30  # 创建一个新的局部变量 x
    print(f"Inside use_local, x = {x}")

print(f"Outside functions, before calls: x = {x}")
modify_global()
use_local()
print(f"Outside functions, after calls: x = {x}")
```

输出：

```
Outside functions, before calls: x = 10
Inside modify_global, x = 20
Inside use_local, x = 30
Outside functions, after calls: x = 20
```

在这个例子中：

- `modify_global` 函数修改了全局变量 `x`。
- `use_local` 函数创建了一个新的局部变量 `x`，不影响全局变量 `x`。

代码：

```
import re

def H_to_T(day,month,year):
    haab_month_list=['pop', 'no', 'zip', 'zotz', 'tzec', 'xul', 'yoxkin', 'mol',
'chen', 'yax', 'zac', 'ceh', 'mac', 'kankin', 'muan', 'pax', 'koyab',
'cumhu','uayet']
    t_month_list=['imix', 'ik', 'akbal', 'kan', 'chicchan', 'cimi', 'manik',
'lamat', 'muluk', 'ok', 'chuen', 'eb', 'ben', 'ix', 'mem', 'cib', 'caban',
'eznab', 'canac', 'ahau']
    #计算一共多少天
    days=year*365+haab_month_list.index(month)*20+day

    #计算T的日期
    t_year=days//260
    rest=days%260
    t_num=rest%13+1 #计算第一个数字
    t_month_index=rest%20 #计算月份的索引
    t_month=t_month_list[t_month_index] #获取月份的名称
    return f'{t_num} {t_month} {t_year}'


n=int(input())
print(n)

for _ in range(n):
    text=input()
    pattern=r'\.'
    replacement=' '
    text_new=re.sub(pattern,replacement,text)

    test_list=list(text_new.split()) #处理Haab年份文本

    day=int(test_list[0])
    month=test_list[1]
    year=int(test_list[2]) #获取打，month，year的信息

    output=H_to_T(day,month,year)
    print(output)
```

代码运行截图 （至少包含有"Accepted"）

545C. Woodcutters

dp, greedy, 1500, https://codeforces.com/problemset/problem/545/C

思路：首先首尾两棵树肯定可以砍。我们从左往右遍历这个一维直线，如果一棵树可以向左倒就向左倒（这样不会妨碍下一棵树），如果不能向左倒考虑向右倒，如果可以向右倒且不会碰到下一棵树就向右倒，否则不砍。

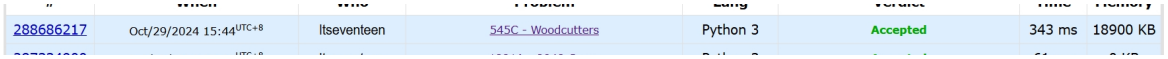
可以向左倒：这棵树和左边一棵树x的差值严格>h

大概用时：40min（参考了题解的思路）

代码：

```
n=int(input())
#建立树的列表，其中每棵树的坐标和高度用一个list表示
tree=[list(map(int,input().split())) for _ in range(n)]

count=2 #初始化计数器,有两棵以上的树的时候至少可以砍首尾，并且最右边的树往右砍是特殊情况！

if n==1: #如果只有一棵树，则直接输出
    print(1)

else:
    for i in range(1,n-1):
        if tree[i][0]-tree[i-1][0]>tree[i][1]: #这棵树可以向左砍
            count+=1
        else: #这棵树不能向左砍，考虑向右砍树的情况
            if tree[i+1][0]-tree[i][0]>tree[i][1]: #向右砍树
                count+=1
                tree[i][0]+=tree[i][1] #向右砍树会占用右边的树左边的位置，不妨将这棵树目前的坐标向右移
            else:
                continue
    print(count)
```

代码运行截图（至少包含有"Accepted"）



288686217	Oct/29/2024 15:44 UTC+8	Itseventeen	545C - Woodcutters	Python 3	Accepted	343 ms	18900 KB

01328: Radar Installation

greedy, http://cs101.openjudge.cn/practice/01328/

思路：

代码：

```
```

代码运行截图（至少包含有"Accepted"）

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“计概2024fall每日选做”、CF、LeetCode、洛谷等网站题目。

好难啊做得我快三了，今日脑子不在思考状态，先提交一个，雷达有时间再看X(

先说我代码的问题，有思路但不知道如何用计算机语言表达的情况依然存在，可以随时用**草稿纸理清思绪**，把用人脑子解决的问题想办法抽象成数字的问题用计算机解决。其次经常犯**细节性错误**，包括但不限于数组越界（使用双指针可以避免一部分越界且思路更清晰）、变量的字母对应出错、+-1的问题、能不能取等于的问题……最后有时候总会把简单的问题想复杂，但是写代码需要**把复杂的问题简单化**才对。可能还是需要更多练习，但是写作业就已经让我半死不活，其他学科作业也可多了比如高数上周留了五十多道作业:(

写作业和看题解的过程中也学到了过量的以前不会的语法，见“启示”部分，大脑已宕机……

每日选做进度已丢失，讲义也有很多内容要做，不知道该从何处跟进了。是先做讲义还是先做作业呢，每日选做又从哪里开始补呢:(

苦痛的期中季