Assignment #5: Greedy 穷举 Implementation

Updated 1939 GMT+8 Oct 21, 2024

2024 fall, Complied by <mark>同学的姓名、院系</mark>

姓名：李彦臻
学号：2300010821
学院：数学科学学院

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：建立一个含有 21252 个 0 的列表，用于记录 d 天之后的 21252 天的状态；列表里的第 i 个天代表第 d+1+i 天。对于列表里的每一天，其每是一个高峰就+1；判断是否是高峰只需考虑是否与 a、b、c 关于 23/28/33 同余即可。最后，只需找列表里 3 的位置即可！

代码：

```python
```

```
def main(a,b,c,d):
    days=[0]*21252#用于记录 d 天之后的 21252 天的状态；列表里的第 i 个天代表第 d+1+i 天
    for i in range(21252):
        if (d+1+i-a) % 23 == 0:#若第 i 天与 a 关于 23 同余，则说明第 i 天是体力高峰
            days[i] += 1
```

```
        if (d+1+i-b) % 28 == 0:#若第 i 天与 b 关于 28 同余，则说明第 i 天是情感高峰
            days[i] += 1
        if (d+1+i-c) % 33 == 0:#若第 i 天与 a 关于 33 同余，则说明第 i 天是智力高峰
            days[i] += 1
    return days.index(3)+1#因为 23/28/33 两两互素，因此一个周期内只会出现一个共同的高峰~
#小心：验证 a+b 是否为 c 的倍数时，不要写为 if a+b % c == 0！！
#这是因为%的优先级比+高，因此系统会识别为 a+(b%c) == 0；因此，要给前面的 a-b 加括号！

i=1#记录这是第几组数据
while True:
    data=list(map(int,input().split()))
    a,b,c,d=data[0],data[1],data[2],data[3]
    if a == -1 and b == -1 and c == -1 and d == -1:
        break
    else:
        k=main(a,b,c,d)
        print(f"Case {i}: the next triple peak occurs in {k} days.")
        i += 1
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

18211: 军备竞赛

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

思路：先分析一下策略：为了使每时每刻手上的钱尽量多，前期应以“做-卖-做-卖-...”的方式积累原始资金；不仅如此，因为对于一个便宜武器而言，现在买和以后买没有本质上的区别，因此，为了让每时每刻手里的钱尽量多（防止购买不了武器），每一轮制作的武器都应当是当下最便宜的武器~并且，每一轮卖的都应当是当下最贵的武器！
可以将每一轮交易都视为一次积累资金；每积累完一次资金，都检查一次当下够购买多少个武器 w。最终的答案便是每一轮可购买的武器个数 w 的最大值！
如果某一轮交易过后手里的资金足够买剩下的所有武器时，便可以结束~

代码：

```python
```
```
p=int(input())
price=list(map(int,input().split()))
price.sort()
n=len(price)#设为“可变的量”，记录每一次操作后所剩的武器数
weapons=[0]#每一轮资金积累后的最大可购买武器数
while True:
    if p >= sum(price):#如果手里的资金足够买剩下的所有武器时，便可直接结束~
        weapons.append(n)#小心：上一轮的循环未把这种情况的 i 添加进列表里！
        #并且，如果这就是第一个循环，就会直接输出 0；所以必须在这里再添加一次 i~
        break
    elif p < price[0]:#如果连最便宜的那个武器都买不起，也可以结束了
        break
    else:#如果不是上述两种极端情况，则进行一轮交易！
    #p.s.这一轮交易一定是能完整进行的，因为如果只剩一个货物并且 p 足够购买它，则属于第一种情况！
        p -= price[0]
        p += price[n-1]
        price.remove(price[0])
        price.remove(price[-1])#用完了之后记得扔掉
        n -= 2#武器个数减二
        if n == 0:#检查此时武器是否已经售罄
            break
        else:#否则，检查当下至多够购买多少个武器 i，并把 i 添加进 weapons 列表内
            total=0#变量名不要用 sum！！会引起混淆
```

```
            for i in range(n):
                total += price[i]
                if total > p:
                    weapons.append(i)
                    break
            #注意：如果 p 大于此时所有剩下的武器的价格之和，则不会把 i 添加进去
            #但没关系，因为下一个循环的开头会把这种情况的 i 添加进去~
print(max(weapons))
```

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：这题较简单，只需简单分析一下即可得出最优策略：设某种排列下第 1~n 个同学的做实验时间为 t1~tn，则等待时间分别为 0 t1 t1+t2 t1+t2+t3 ...
此时，总等待时间为 S=(n-1)t1+(n-2)t2+...+tn-1，只用让 S 尽量少即可~又因为 ti 的权重随

i 变大而变小，因此要让 t1 最小，tn 最大！
除了思路以外，这题帮我复习了一个知识点，就是保留小数点后 k 位的方法：
print("{:.kf}".format(_))！

代码：

```python
```
```
n=int(input())
time=list(map(int,input().split()))
queue={}#创建一个字典，把每个学生的序号与时间对应起来
for i in range(n):
    queue[i+1]=time[i]
good_queue=sorted(queue.items(),key=lambda item:item[1])
#小心：此时 good_queue 是一个由元组组成的列表，而不是字典！
indexs=[]#记录排好后的队列里每个学生原本的序号
times=[]#记录排好后的队列里每个学生的时间
for i in range(n):
    indexs.append(good_queue[i][0])
    times.append(good_queue[i][1])
total=0#记录总等待时间
for i in range(n):
    total += (n-1-i)*times[i]
average=total/n
print(" ".join(map(str,indexs)))
print("{:.2f}".format(average))#保留小数点后两位的方法！
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

01008: Maya Calendar

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

思路：这题本质的思路是比较简单的，无非就是两个步骤：先把第一个日历的日期转化为世界开始的第 t 天，再利用第二个日历的性质换算出这一天在第二个日历中的表示方法。
主题思路是比较容易想到的，但需要注意一些+1 或者-1 的小的细节，以及别忘了输出的时候得先把组数打印出来！！

代码：

```python

```

months=['pop','no','zip','zotz','tzec','xul','yoxkin','mol','chen']

```
months.extend(['yax','zac','ceh','mac','kankin','muan','pax','koyab','cumhu','uayet'])
days=['imix','ik','akbal','kan','chicchan','cimi','manik','lamat']
days.extend(['muluk','ok','chuen','eb','ben','ix','mem','cib','caban','eznab','canac','ahau'])

def transfer():#把第一个日历的日期先转化为世界开始的第几天
    date=list(map(str,input().split()))
    day_list=list(map(str,date[0]))
    day_list.remove(day_list[-1])#把日期末尾的.给去掉
    day=int("".join(map(str,day_list)))#把日期求出来
    #p.s.日期起始于0，因此要把世界开始的那一天假设为第0天
    month=months.index(date[1])#求月份
    year=int(date[2])#求年份
    t=day + month*20 + year*365#找出这是世界开始的第几天

    #把t求出来之后，再把这一天转化为Tzolkin日历的日期
    y=t // 260#先把年份求出来
    k=t % 260#再求出这是该年的第几天
    d=(k % 13) + 1#求“日期余数”时别忘了+1，因为“日期余数”起始于1！！
    day_number=k % 20
    m=days[day_number]#最后再把“日期名字”求出来
    print(f"{d} {m} {y}")

n=int(input())
print(n)#别忘了这一步！！得先把组数打印出来~
for i in range(n):
    transfer()
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

545C. Woodcutters

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

思路：这道题的 greedy 模式其实很类似于 BerSU_ball 那道题：如果从左边的树开始考虑的话，应当让靠近左边的树尽量往左边倒，这样可以尽量预留空隙给右边的树，让他们拥有尽量多的选择和可倒空间！
可以证明，这种方法就是最优解；因为哪怕有一个树 A 在某个方法下可以倒，但在这个方法下没有倒，那么就说明 A 右边的的空隙不够他倒，并且左边的也被占据了导致空间不够他倒；换句话说，A 的长度加上他左边那棵树 B 的长度大于这个他们两之间空隙的长度！
因此此时，如果要让他倒，就一定会牺牲 B，因为 A 要倒就必须往左边倒，然后空隙就不够 B 倒了；而 B 也不能往他自己的左边倒，因为根据这个方法的原理可知，B 就是因为他自己的左边空间不够了才往右边倒的。。
所以如果 B 非要倒，又会让 B 的左边的树没法倒；如此下去，不难看出，要让 A 倒，一定会牺牲一个树！证毕。

代码：

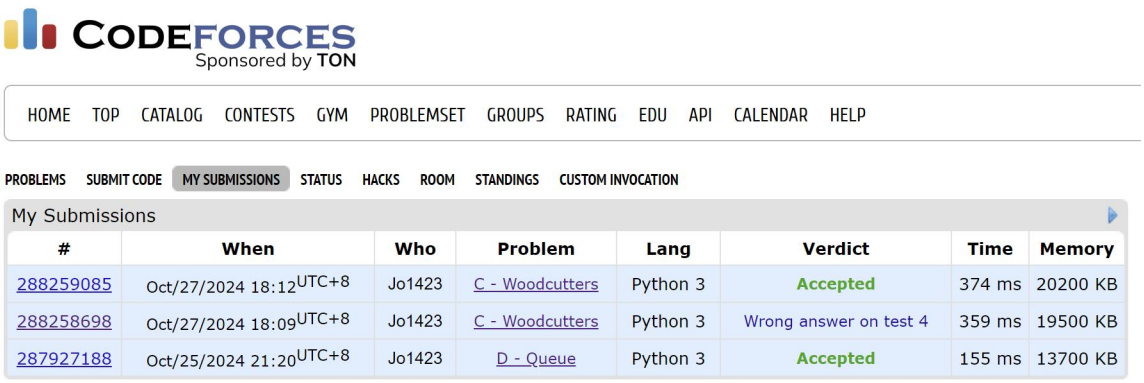
```python

```
```
n=int(input())
trees=[]
for i in range(n):
    tree=list(map(int,input().split()))
    trees.append(tree)

status=[0]*n#记录每个树往哪边倒；0 代表左，1 代表右，2 表示没倒
for i in range(1,n-1):#只用考虑第 2~n-1 个树（第 1、n 个树直接往左、右倒就行了，自动记为 0、1）
    #因为最后只需检查有几个树为 2（没有倒），所以最后一个数干脆记为 0 也没问题~
    #接下来，根据左边的那个树往哪边倒来分类讨论即可~
    if status[i-1] == 0 or status[i-1] == 2:
        if trees[i][1] < trees[i][0] - trees[i-1][0]:
            status[i]=0
        elif trees[i][1] < trees[i+1][0] - trees[i][0]:
            status[i]=1
        else:
            status[i]=2
    else:#只有左边那个数往右倒时需要单独讨论~
        if trees[i][1] < trees[i][0] - trees[i-1][0] - trees[i-1][1]:
            status[i]=0
        elif trees[i][1] < trees[i+1][0] - trees[i][0]:
            status[i]=1
        else:
            status[i]=2

print(n - status.count(2))#最后，用 n 减去 status 里 2 的个数（没倒的树）即可~
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



01328: Radar Installation

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

思路：先对每个岛进行“从左到右”的排序；注意，这里的从左到右不是对横坐标进行排序，而是对 xi 排序！（其中 xi 是对第 i 个岛而言，能覆盖到他的最右边的雷达的横坐标）
现在，从左往右布置雷达；在覆盖到“当前最左的岛屿”的情况下，应当使每个雷达尽量往右布置，这样就可以覆盖到更多的还未被覆盖到的岛屿~不难想象，这种布置雷达的方式显然是最优的（可以使每个雷达的利用率最大化）

代码：

```python
```
```
def number(n,d):
    islands=[]
    for i in range(n):
        islands.append(list(map(int,input().split())))
    location={}#用字典记录每个岛屿和它对应的 xi
    k=0#预先给 k 赋一个值，否则下下步无法检查 k 是否为-1
    for island in islands:
        x,y=island[0],island[1]
        if y > d:#第一种情况：如果有岛屿不可能被覆盖，则直接把 k 设为-1
            k=-1#这里 k 就是函数最终输出的结果
        else:
            location[(x,y)]=x + (d**2-y**2)**0.5
            #注意！！这里要把岛屿以元组的形式储存进字典，不要用列表！
            #否则一会对字典的值进行排序时，无法把列表放进结果元组内！！！
    if k != -1:#确定雷达方案存在后再进行下一步（对岛屿按照 xi 排序）
        m=len(location)
        #一定要小心，有可能有重复的岛屿！！!那样会导致字典实际长度 m 小于 n，因此要把 m 视为新的 n！！
        sorted_dic=sorted(location.items(),key=lambda item:item[1])
        sorted_islands=[]#记录按 xi 排序后的岛屿
        sorted_radar=[]#记录按 xi 排序后的每个岛屿对应的 xi
        for i in range(m):
            sorted_islands.append(sorted_dic[i][0])
            sorted_radar.append(sorted_dic[i][1])
        radar=[sorted_radar[0]]#用于记录每个最终每个雷达的横坐标；x0 可以先直接扔
```

进去

```
        i=1#从第二个岛屿开始考虑即可
        while i < m:
          if (sorted_islands[i][0] - radar[-1])**2 + sorted_islands[i][1]**2 > d**2:
                radar.append(sorted_radar[i])
                #如果目前已有的最右边的雷达无法覆盖当下的岛屿，就把当下这个岛屿对应的雷达安装了
             i += 1
        k=len(radar)
    return k

t=1#记录这是第几组case
result=[]
while True:
    data=list(map(int,input().split()))
    n,d=data[0],data[1]
    if n == 0 and d == 0:
        break
    else:
        k=number(n,d)
        result.append(f"Case {t}: {k}")
        input()#别忘了把两个案例之间那一行空格给扔了~
        t += 1
for case in result:
    print(case)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

2. 学习总结和收获

<mark>如果作业题目简单，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。</mark>

最近在做每日选做的过程中，我遇到一些比较有意思或者比较难的题的时候，除了自己总结一些方法以外，我还会一边体会群里一些同学做题的方法，一边跟着 ai 学习一些算法和思维；比如我的 dp 和双指针都是这么学会的（虽然 o(n) 的 dp 写起来还不是很熟练，但至少已经明白原理了，并且已经拿来解决了不少题了，蛮有成就感的~）。

因此，在做这次作业的过程中，虽然里面已经有几道略有难度的题了，但是我还是基本上每道题都能在半个小时之内 ac，那两三道比较简单的题也都是十分钟内解决的，还是蛮有成就感的~我现在做这些 1100~1200 的 greedy 或者 dp 题的时候，思路基本上很快都能想出来，反而遇到的问题主要都是一些小的细节问题，比如程序里的一些小逻辑问题，或者在一些+1 或者-1 的细节上会 debug 很长时间之后才能发现原来这里有一个这么小的错误（笑）；希望自己能在不断做题和总结经验的过程中，以后逐渐少犯这些小错误，避免浪费无谓的时间吧~

这次作业我觉得主要比较有难度的就是第二题（军备竞赛），策略比较难想，但想出来了之后代码还是比较好写的；还有就是第六题比较难，除了思路以外，在编程的过程中我遇到了一个

细节点，就是在把岛屿（列表）与一个数对应起来放进列表里时，要把岛屿以元组的形式储存进字典，不要用列表！因为否则一会对字典的值进行排序时，无法把列表放进字典排序好后的结果元组内！！！（因为列表是不可哈希的）；不仅如此，这道题里还有两个坑点：
第一个是一定要小心，有可能有重复的岛屿！那样会导致字典实际长度 m 小于 n，容易出 bug!!
还有就是在输入数据时一定要多写一个 input()，别忘了把两个案例之间那一行空格给扔了，否则就会 RE！！

在做某一道每日选做题的过程中，因为运用到了列表和元组的性质，并且也涉及到了判断两个列表是否相等，因此我学到了一些与列表和元组有关的知识，比如系统判断两个列表是否相同的时候会考虑顺序；（十分方便！）

python能否判断两个列表是否相等

是的，Python 可以直接判断两个列表是否相等。你可以使用 `==` 运算符来比较两个列表的内容和顺序。如果两个列表的元素、顺序都相同，则它们被视为相等。

示例代码

python 复制代码

```python
list1 = [1, 2, 3]
list2 = [1, 2, 3]
list3 = [3, 2, 1]

print(list1 == list2)  # 输出: True，内容和顺序相同
print(list1 == list3)  # 输出: False，顺序不同
```

注意

- **元素顺序**：`==` 比较时，不仅要求列表内容相同，还要求顺序一致。
- **嵌套列表**：即使是嵌套列表，`==` 也能正确判断两者是否完全相等。

同样，元组也是有序的~但是元组可以存入 set，列表不行；所以要判断当下输入的每个二元数组之前是否已经输入过，可以把二元数组储存为元组！

元组是有序的吗（顺序不一样认为是相同的吗）

是的，**元组是有序的**，这意味着元组中的元素按定义的顺序排列，且顺序不同的两个元组被视为不同的对象。

有序性示例

python 复制代码

```python
tuple1 = (1, 2, 3)
tuple2 = (3, 2, 1)

print(tuple1 == tuple2)  # 输出: False
```

在上面的示例中，`tuple1` 和 `tuple2` 的元素内容相同，但顺序不同，因此 Python 认为它们不是相同的元组。

我还顺便自学了一下可哈希和不可哈希的概念的定义，并了解到了：集合和元组内只能有可哈希的量，比如 int/float/complex/str/tuple！

什么叫不可哈希

在 Python 中，**哈希（hash）** 是一种将对象映射为一个固定大小的数值（称为哈希值）的方法。具有哈希值的对象可以通过哈希值快速地存储和查找，因此可以用作集合（`set`）和字典（`dict`）的键。**不可哈希**的对象则没有固定的哈希值，无法直接用作集合或字典的键。

可哈希对象的特性

一个对象是可哈希的需要满足以下条件：

1. **哈希值固定**：对象的哈希值不会随时间变化，因此在数据结构中可以高效查找。
2. **不可变性**：可哈希对象的内容是不可改变的（不可变类型）。例如，整数、浮点数、字符串和元组等都是不可变的，因而是可哈希的。
3. **实现 `__hash__()` 和 `__eq__()` 方法**：可哈希对象必须实现 `__hash__()` 方法来生成哈希值，同时也要实现 `__eq__()` 方法确保相等比较的逻辑。

常见的可哈希和不可哈希类型

- **可哈希类型**（不可变）：整数（`int`）、浮点数（`float`）、字符串（`str`）、元组（`tuple`）等。
- **不可哈希类型**（可变）：列表（`list`）、字典（`dict`）、集合（`set`）等。