



高级编程技术实验报告

实验一：空间奶牛传输

数据科学与计算机学院 17大数据与人工智能

17341015 陈鸿峥

一、问题描述及求解思路

1. Part A: Transporting Cows Across Space

(i) 读入数据

先创建一个空字典，然后利用python的with...as...语句读入文件，有效避免文件不存在或忘记关闭等情况（会自动报错）。

然后直接枚举文件的每一行，通过split对字符串进行分割，存在元组里，进而创建字典项。

```
res = {} # create an empty dictionary
with open(filename,"r") as file: # avoid exception
    for line in file:
        (name,num) = line.split(',')
        res[name] = int(num) # remember to take int
```

(ii) 贪心算法

先对读入的字典按照value进行降序排列，利用python的sorted函数可以很方便做到。同时sorted会返回一个新的列表，避免对原字典的破坏。这里还利用了lambda表达式，指明是以value为key排序。

```
cow_sorted = sorted(cows.items(),key=lambda item:item[1],reverse=True)
```

然后每次遍历cow_sorted，从大到小依次添加不会超过limit的表项，每添加一项就从列表中删除该项。每执行完一次遍历，就将当前trip添加到结果数组res中。然后重新开始新一轮遍历，直到cow_sorted为空。代码如下，已将原题注释删除。

```
def greedy_cow_transport(cows,limit=10):
    # Firstly sort the dict by value
    # The code below will not ruin the original dict
    cow_sorted = sorted(cows.items(),key=lambda item:item[1],reverse=True)
    res = []
    # the input must ensure the biggest cow <= limit
    while (len(cow_sorted) > 0):
        curr_weight = 0
        one_trip = []
```

```

    for item in cow_sorted[:]: # copy out!
        if (curr_weight + item[1] <= limit):
            curr_weight += item[1]
            one_trip.append(item[0]) # append the name
            cow_sorted.remove(item)
    res.append(one_trip) # add the last
return res

```

(iii) 暴力算法

暴力算法十分直接，就是枚举每一个partition，判断该partition是否合法，即partition中每一个trip的总重不超过limit。如果不合法，直接跳到下一个partition；合法，则判断是否为当前轮数最少，如果是，则更新最小轮数。代码如下，已将原题注释删除。

```

def brute_force_cow_transport(cows, limit=10):
    lst = cows.items() # will not ruin the original dict
    min_num_trip = len(cows)
    min_trip = []
    for partition in get_partitions(lst):
        flag = False # use for test if all trips valid
        for trip in partition:
            weights = sum([item[1] for item in trip]) # sum all weights in one trip
            if (weights > limit):
                flag = True
                break
        if flag:
            continue
        elif (len(partition) < min_num_trip): # find the min trip
            min_num_trip = len(partition)
            min_trip = [[item[0] for item in trip] for trip in partition] # take
                                ↪ out all the names
    return min_trip

```

(iv) 比较时长

将两种方法分别放在两个time.time()中即可测定运行时间。代码见ps1a.py，运行结果见图1。

(v) 分析

本部分为Problem A.5:Writeup的内容。

1. 结果见图1，除了一组原始测试样例外，还给出了自己的测试样例。从图1中可以看出，贪心算法明显比暴力算法快几个数量级。因为贪心算法只需对原字典进行一次排序，然后从大到小依次遍历插入即可，而且插入的项在原表项中可以直接删除，越到后面遍历的元素越少，最坏情况下也是平方级时间复杂度。而暴力算法的瓶颈在于划分get_partitions，

需要将所有可能的排列划分全部枚举出来，这是指数级时间复杂度。故明显贪心比暴力要快。

2. 贪心算法不一定能返回最优解，因为它之关心当前轮次是否有办法尽可能多地运满，而不考虑之后地轮次，故这是一种鼠目寸光，只在乎眼前的策略，好的情况可以达到最优解（如我给的测试样例），但坏的情况就不行了（如原题测试样例）。
3. 暴力算法一定能返回最优解，因为它相当于枚举了所有可能的情况，然后从中挑选出一个最好的，这当然是全局最优解。

2. Part B: Dynamic Programming: Hatching a Plan

(i) 动态规划

本题为典型的背包问题，对于每一个目标重量，枚举所有可能的鸡蛋重量（由于有无限鸡蛋），然后判断是否应该将该鸡蛋放入，放入总数量加1，不放则不做处理。

对于某一个目标重量 w_t ，其最优的鸡蛋数目 $f(w_t)$ 满足下列动态转移方程

$$f(w_t) = \begin{cases} 0 & w_t = 1 \\ \min\{1 + f(w_t - w_e)\}, \forall w_e & w_t \geq 1 \end{cases}$$

其中 w_e 为每一个鸡蛋的重量。

进而可以使用bottom-up的方法，对不同 $f(w_t)$ 进行计算，最终得到 $f(target_weight)$ 。代码如下，原题注释已删减。

```
def dp_make_weight(egg_weights, target_weight, memo = {}):
    memo[0] = 0
    for target in range(1, target_weight+1):
        for weight in egg_weights:
            if (target >= weight):
                # DP transition equation
                memo[target] = min(memo.get(target, target), 1+memo[target-weight])
    return memo[target_weight]
```

(ii) 分析

本部分为Problem B.2:Writeup的内容。

1. 注意这是一个无限背包问题，这相当于解下面的不定方程

$$w_1x_1 + w_2x_2 + \cdots + w_{30}x_{30} = W, x_i \geq 0, i \in \{1, 2, \dots, 30\}$$

其中 w_i 为每个鸡蛋的重量， W 为目标重量， x_i 为要求解的每个鸡蛋的数目。由于每个 x_i 都

是非负数，故要进行暴力枚举，最差的情况要进行 $(W+1)^{30}$ 种排列组合¹，这个数字太过庞大，显然是不可能实现的。

2. 最开始我就采用了如下的贪心算法，但后来我发现并不能求到最优解。

```
if (len(egg_weights) == 0):
    return 0
num_eggs = target_weight // egg_weights[-1]
return num_eggs + dp_make_weight(egg_weights[:-1], target_weight - num_eggs *
    ↪ egg_weights[-1])
```

目标函数即为

$$f(w_t) = w_t \text{ div } w_e + f(w_t - w_e * (w_t \text{ div } w_e))$$

其中 w_e 已经按照降序排列，边界情况见程序。限制条件即鸡蛋是否已经由大到小被挑选完。策略同目标函数，每次选择最重鸡蛋，然后尽可能多地用最重填，填不满的部分再用轻的鸡蛋。

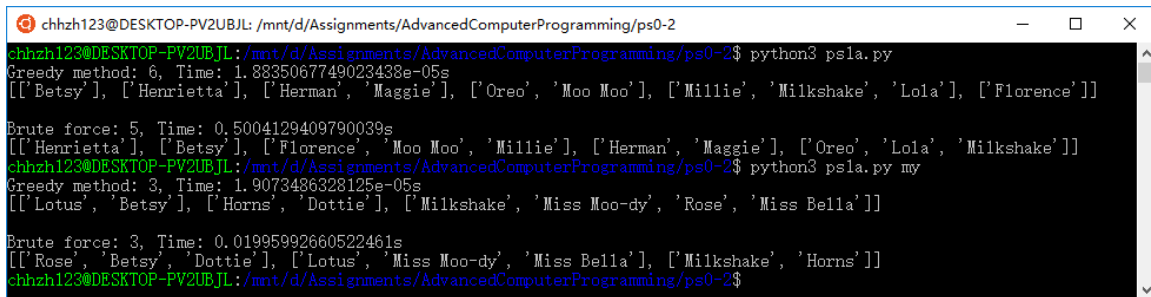
3. 不一定，如图2中给的第二个测试样例。鸡蛋重量分别为(1, 5, 11)，目标重量为15。如果用贪心算法，先选1个11，剩下4没得选，只好填4个1，共5个鸡蛋。但采用动态规划，可以求得只用3个5单位重量的鸡蛋。

二、代码

代码实施及注释请见附件 `ps1a.py`、`ps1b.py`。

三、运行截图

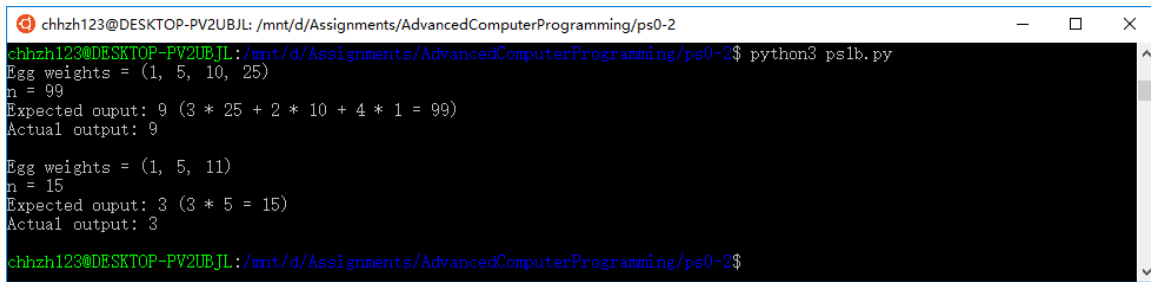
实验运行结果如下面几幅图片所示，注意给出了多个自己写的测试样例，确保结果的正确性和鲁棒性。



```
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2$ python3 ps1a.py
Greedy method: 6, Time: 1.8835067749023438e-05s
[['Betsy'], ['Henrietta'], ['Herman', 'Maggie'], ['Oreo', 'Moo Moo'], ['Millie', 'Milkshake', 'Lola'], ['Florence']]
Brute force: 5, Time: 0.5004129409790039s
[['Henrietta'], ['Betsy'], ['Florence', 'Moo Moo', 'Millie'], ['Herman', 'Maggie'], ['Oreo', 'Lola', 'Milkshake']]
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2$ python3 ps1a.py my
Greedy method: 3, Time: 1.9073486328125e-05s
[['Lotus', 'Betsy'], ['Horns', 'Dottie'], ['Milkshake', 'Miss Moo-dy', 'Rose', 'Miss Bella']]
Brute force: 3, Time: 0.01995992660522461s
[['Rose', 'Betsy', 'Dottie'], ['Lotus', 'Miss Moo-dy', 'Miss Bella'], ['Milkshake', 'Horns']]
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2$
```

图 1: Part A 结果，给出测试样例和自己写的样例输出

¹这只是一个粗略估计，并未考虑 w_i 的顺序、大小等问题



```
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2$ python3 ps1b.py
Egg weights = (1, 5, 10, 25)
n = 99
Expected output: 9 (3 * 25 + 2 * 10 + 4 * 1 = 99)
Actual output: 9

Egg weights = (1, 5, 11)
n = 15
Expected output: 3 (3 * 5 = 15)
Actual output: 3

chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/AdvancedComputerProgramming/ps0-2$
```

图 2: Part B结果，给出测试样例和自己写的样例输出