Problem Set 1 实验报告

17340027 姚洁倩

A．

1.题目描述

题目假设了一个场景：使用宇宙飞船运载牛，其中宇宙飞船能够运载的重量有限，每一头牛都有自己的重量，将这些牛全部运走，最小化使得旅程的次数。

2.解题思路

文档中将题目分成了三个部分，第一部分加载牛的数据，第二部分使用暴力解法解题，第三部分使用贪心算法解题。

第一部分，加载牛的数据。这部分的工作主要是读取文件，并且将相对应的数据放到一个字典中，并作为结果返回。首先建立一个空的字典，接下来一行行读取文件，使用 for 循环，每次读取文件中的一行，读完之后将结尾的换行符去掉，并将这一行字符串以逗号作为分隔符，分割成 list，索引为 0 的即是牛的名字，索引 1 是牛的重量，将牛的名字作为 key，将重量作为对应的映射，存储进字典中。

第二部分，使用贪心算法解题。按照题目的要求，总是选取重量最大的且在飞船还能运载的重量之内的牛。在这里使用了一个列表 my_list 存储尚且未被运输的牛。在初始化列表的时候，将字典中的牛及它们的重量，转成一个二元组存储在这个列表中。接下来，以牛的重量作为比较的值，使用 sort 函数对列表进行从大到小的排序。排好序之后就可以开始计算运输的方案了。方法是使用一个 while 循环，当剩下的牛的列表,,非空时，循环就会继续。有一个 index 索引指示在剩余牛列表中应当检查哪一只牛符不符合上飞船的条件。若发现飞船剩余的容量还能容纳下这只牛，那么则将这只牛加入这趟旅程，并且从剩余牛列表中删去这只牛，否则就检查下一个索引。每次循环尾部都会判断索引是否到了剩余牛列表的末尾，若到了，则说明已经遍历过一遍这个列表且再找不到可以加入这趟旅程的牛了，这时候就能够确定下当前的旅程所要运载的牛数量。将开启一趟新的旅程并将索引置为零。在最后返回所有的旅程列表即可。

第三部分，使用蛮力方法解题。借助已经给出的 get_partition 函数，能够得到所有的划分情况。对于每一种划分的情况，计算其中每次的旅程是否超出宇宙飞船的重量，若超出，则说明这种划分方案不可行。若不超出，则可以看一下这种划分方案需要的旅程数，若小于目前记录的最小旅程数，则将这种划分方案记录下来，并更新最小旅程数。在遍历完所有的划分方案之后就能够得到最小旅程的方案了。

第四部分，比较两种运输的方案。使用 time 库中的 time 函数，分别计算两种方案所得出解的时间即可。

第五部分，writeup，写在了本报告最后的部分。

3.部分测例及运行截图

```
PS C:\coding\Python\17340027-姚洁倩-p1> python .\ps1a.py
Greedy:Time spent: 0.01513361930847168
minimun number of trip: 6
solution: [['Betsy'], ['Henrietta'], ['Herman', 'Maggie'], ['Oreo', 'Moo Moo'], ['Millie', 'Milkshake',
'Lola'], ['Florence']]
Brute force:Time spent: 0.8234908580780029
minimun number of trip: 5
solution: [['Henrietta'], ['Betsy'], ['Millie', 'Florence', 'Moo Moo'], ['Oreo', 'Lola', 'Milkshake'], [
'Maggie', 'Herman']]
```

```
PS C:\coding\Python\17340027-姚洁倩-p1> python .\ps1a.py
Greedy:Time spent: 0.025257587432861328
minimun number of trip: 5
solution: [['Lotus'], ['Horns'], ['Dottie', 'Milkshake'], ['Betsy', 'Miss Moo-dy', 'Miss Bella'], ['Rose']]
Brute force:Time spent: 0.07874131202697754
minimun number of trip: 5
solution: [['Dottie', 'Rose'], ['Lotus'], ['Miss Bella', 'Miss Moo-dy', 'Milkshake'], ['Betsy'], ['Horns']]
```

这里测试了两个 txt 文档的数据

B．

1.题目描述

题目假设了一个场景，有几种不同重量的蛋（数量无上限），大小一样。要运输一定重量的蛋，因为每个蛋所占空间大小一样，求一个最优化方案，即要运输的最少的蛋的数量。

2.解题思路

第一部分，动态规划解题。题目要求使用动态规划进行解题。首先明确这是一个完全背包问题，有几种不同重量的蛋，蛋的数量是无限个，给出一个目标重量，求出达到目标重量的要运输的最小蛋的数量。我们可以根据这个来列一个状态转移方程。使用一个 dp_egg 数组来记录达到某重量的最小蛋个数，即 dp_egg[W]：在目标重量为 W 的时候，所需要的最小蛋的数目。不妨记这几种不同重量的鸡蛋为 egg1，egg2，egg3……eggn。于是可以将大的问题分解成更小的子问题，状态转移方程为 $dp\_egg[W] = \min_{eggi<=W}\{dp\_egg[W-eggi]+1\}$，即对重量 W，它必定是由 W-eggi+eggi 得到的，就是说，在重量为 W 的情况若是最优解，那么取出一个鸡蛋之后的 W-eggi 也必定是最优解，那么我们只要找到放入这个蛋之前的重量所对应的最小数量，加上 1 就是重量为 W 对应的最小数量了。那么由此可以写出 for 循环，自底向上计算从重量 1 到重量 W 的所需要的最小运载蛋数即可。

第二部分，writeup 写在了本文档的后部分。

3.部分测例及运行截图

```
PS C:\coding\Python\17340027-姚洁倩-p1> python .\ps1b.py
Egg weights = (1, 5, 10, 25)
n = 99
Expected ouput: 9 (3 * 25 + 2 * 10 + 4 * 1 = 99)
Actual output: 9

Egg weights = (1, 50, 95)
n = 100
Expected ouput: 2 (2 * 50 = 100)
Actual output: 2

Egg weights = (1, 9, 25, 30, 35, 40, 45, 92)
n = 98
Expected ouput: 4 (9 * 2 + 35 + 45 = 98)
Actual output: 4
```

此处测试了三个样例，都成功通过了。

Writeup 部分

Problem A.5:Writeup

1.结果显示，贪心算法运行的时间比蛮力解法运行时间要短很多。 原因是蛮力方法考虑了所有的组合情况，集合的划分个数与集合的元素个数是指数级的关系，当 n 越大，所花费的时间呈指数级增长。而对于贪心的算法，将牛按照重量进行排序，排序时间不妨认为是O(nlogn)（如果是快速排序的话）。接下来，只要遍历这个排好序的序列，从中取出符合条件的牛，遍历这个序列需要 O(n)时间，而遍历的次数在最坏情况下为 n 次，那么贪心算法所花费的时间为 O(n^2)，在时间上显然比蛮力穷举的方法要好。

2.贪心算法并不能得到最优解，因为贪心方法每次取的都是所能够装得下的最大重量的牛，但是最佳的组合方案也许并不是这样的。例如说，限制的重量为 10，而牛的重量分别为 6、5、3、2、2、2，使用贪心算法解得的是[6,3][5,2,2][2]，但实际上有更优的方案[6,2,2][5,3,2]。

3.蛮力方法可以得到最优解。因为它考虑了所有的组合情况，确保了不会有遗漏。并且一个个都计算了，那么只要在其中选取符合条件且是最小的旅程数的就可以了。

Problem B.2:Writeup

1.因为蛮力解法需要考虑所有的组合。当有 30 种不同重量的鸡蛋时，需要考虑这个集合所有的子集，除去空集之外总共有 2^30-1 种情况。这个子集代表的意思是使用这些重量的鸡蛋得到目的的总重。在一个子集之中，还要考虑各种鸡蛋使用的不同数量来凑齐目的的重量。因此，蛮力解法的时间复杂度是指数级的，耗时非常高。

2.如果使用贪心的解法，限制的条件是目的总重量。需要找到达到总重量的最少数目的鸡蛋。那么，此时的策略将会是不断地选取最大重量的鸡蛋而不超过限制的总重。由于总是会有重量为 1 的鸡蛋，所以总是能够凑整。

3.贪心算法并不总是返回一个最优的解。比如蛋的重量有 1，50，95 这三种，限制的总重量为 100，那么按照贪心的策略，首先取 95 的，那么接下来会取 5 个 1 的，那么需要运载的数量就为 6；但是实际上，最优解是取两个 50 的，运载的数量为 2。