problem set 1实验报告

16337183 孟衍璋

Problem A.1

首先需要将txt文件中的数据读入，用with open将文件打开，再用readlines命令将文件中每行的数据存入一个列表中，用split命令将每一行中的名字和重量分开，分别存入dictionary中。实现如下：

```
def load_cows(filename):
    with open(filename) as f:
        lines = f.readlines()  # 逐行读取txt文件，返回的lines为一个list
        dictionary = {}
        # lines的每一项都是一个包含奶牛名称和重量的字符串，通过split函数将他们分开后传入dictionary中
        for i in range(len(lines)):
            lines[i] = lines[i].replace('\n', '')
            temp = lines[i].split(',')
            dictionary[temp[0]] = int(temp[1])
        # print(dictionary)
    return dictionary
```

Problem A.2

第二部分是使用贪心算法，计算在有一定重量限制的船上运送指定重量的牛，运送次数最少的运输方案。实现代码如下：

```
def greedy_cow_transport(cows,limit=10):
    # 将字典按照value值的大小排序
    sorted_by_weight = sorted(cows.items(), key = lambda item:item[1], reverse = True)
    # print(sorted_by_weight)
    transport = []  # 用来存储最终的运输方案
```

```
    transported = []   # 用于判断奶牛是否已经被运输
    for _ in range(len(sorted_by_weight)):
        transported.append(0)

    while all(transported) == 0:  # 如果所有的奶牛都已经被运输，则退出循环
        transport_once = []  # 存储每一次运输的奶牛的名字
        already_token = 0  # 目前已经运输的奶牛的重量
        for i in range(len(sorted_by_weight)):
            if already_token + sorted_by_weight[i][1] <= limit and
transported[i] == 0:
                already_token += sorted_by_weight[i][1]
                transported[i] = 1  # 代表这只牛已经被运输
                # print(sorted_by_weight[i][0])
                transport_once.append(sorted_by_weight[i][0])
        transport.append(transport_once)
    # print(transport)
    return transport
```

Problem A.3

第三部分需要使用暴力搜索算法，运用给的生成所有partitions的函数，遍历每一种情况，选出其中合法的部分，再计算出其中运输次数最少的。实现代码如下：

```
def brute_force_cow_transport(cows,limit=10):
    valid_transport = []
    partitions = get_partitions(cows)
    for partition in partitions:  # partition是某种分配的方式
        flag = 1  # 判断该种分配方式是否符合重量规范
        for transport_once in partition:  # transport_once是每种分配方式中运
输一趟搭载的奶牛
            total_weight = 0  # 计算每趟运输总重量
            for cow in transport_once:
                total_weight += cows[cow]
                # total_weight += cow
            if total_weight > limit:  # 如果重量大于限制，则不行
                flag = 0
        if flag:  # 如果符合条件，则存储这种分配方案
            valid_transport.append(partition)
    # valid_transport里的元素仍是列表，要选出其中长度最短的列表，其长度就是最少的旅途
数
```

```
    return min(valid_transport, key = len)
```

Problem A.4

第四部分主要是算出上述两个算法所使用的时间，并输出进行比较。实现如下：

```
def compare_cow_transport_algorithms():
    cows = load_cows(r"./ps1_cow_data.txt")

    # 贪心算法花费的时间
    start = time.time()
    greedy = greedy_cow_transport(cows)
    end = time.time()
    print("greedy =", end - start)
    print("number of trips:", len(greedy))
    # print(greedy)

    # 暴力搜索所花费的时间
    start = time.time()
    brute = brute_force_cow_transport(cows)
    end = time.time()
    print("brute =", end - start)
    print("number of trips:", len(brute))
    # print(brute)
```

Problem A.5

1. 在上一个问题中，输出为：

可以看出，贪心算法用的时间几乎可以忽略，而暴力搜索需要较多的时间，是因为暴力搜索需要遍历所有的情况，再选出一个最优的，所以花的时间较长。

2. 贪心算法不一定会返回最优的结果，因为它有可能会陷入局部最优。

3. 暴力搜索算法一定会返回最优的结果，因为它遍历了所有情况，再从中选出一个最优的。

Problem B.1

这部分需要使用动态规划算法，计算要凑出指定重量的蛋，需要的最少的给定重量的蛋。实现如下：

```
def dp_make_weight(egg_weights, target_weight, memo = {}):
    list(egg_weights).sort() # 按照重量给蛋排序
    dictionary = {0:0}  # 字典中的key表示需要的重量，value为对应的重量需要的蛋数目
    for weight in range(1, target_weight + 1):
        dictionary[weight] = float('inf')
        for egg in egg_weights:
            if egg <= weight:  # 如果蛋的重量小于当前需要的重量，则可以将其加入
                dictionary[weight] = min(dictionary[weight],
dictionary[weight - egg] + 1)
    if dictionary[target_weight] == float('inf'):
        return -1
    else:
        return dictionary[target_weight]
```

Problem B.2

1. 因为暴力搜索算法的时间复杂度是成指定函数增加的，在有限时间内完成是不现实的。
2. 如果使用贪心算法，目标函数就是凑出指定重量的蛋的策略，约束是给定的蛋的质量，贪心算法的策略是选择满足条件的最大的蛋加入。
3. 贪心算法并不一定返回最优的结果，比如需要总重量为100，而现在拥有的蛋的重量为95，50，1。最优的结果应该是两个50，但贪心算法会返回1个95加5个1的结果。