# 库存规划问题

## 问题描述

公司生产一种设备，每个月的需求都在变化，因此公司希望设计一种从策略赖规划生产，需求是给定的，即它虽然是波动的，但是可预测的。公司希望设计接下来的n个月的生产计划。对第i个月，公司知道需求di，即该月能够销售出去的设备的数量。公司雇佣的全职员工，可以一个月制造m台设备，若希望制造超过m台设备，可以额外雇佣兼职劳动力，雇佣成本为每制造一台设备付出C元。而且如果月末有设备尚未售出，公司还要付出库存成本，保存j台设备的成本为h(j)，h(j)=5\*j，为单调非递减函数。请安排每个月的生产计划，使得满足需求的前提下最小化成本。

## 问题分析

当待求解的问题具有以下三个特点的时候，适用动态规划方法解决：

最优子结构：父问题的最优解是由小问题的最优解构成的。

无后效性：如果给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响。

重叠子问题：相同的子问题可能被重复求解多次。

本题适合使用递归，将问题分成数个子问题并分别求出最优解。显而易见，本题在动态规划算法的适用范围内。

本题目中任意一月生产和本月销售，上月结余以及本月结余均有关。若使用贪心算法则难以兼顾上月结余和本月为未来的结余。

1) 要满足n个月的总需求，那么n个月总共要生产D台设备；

2) 要满足前i个月 (1≤i<n)的需求，那么前i个月生产的设备数量不能少于前i个月的需求之和。

## 动态规划算法求解思路

1. 设置cost矩阵，cost[i][j]表示前i个月总共生产j台设备的前提下的最小成本。理论上j的取值范围为0到D，i的取值范围为1到n。
2. 设置p矩阵，p[i][j]表示前i个月生产设备总数为j的情况下，第i个月选择生产的设备数量。
3. 设置D列表，长度为n，D[i]表示前i个月总需求。于是，假设第i个月生产k台设备，则前i-1个月要生产j-k台设备，第i个月库存数量为j-D[i]。
4. 核心结构：本算法核心在于cost[i][j]矩阵和p[i][j]矩阵的赋值。根据题目要求，对于前i个月的生产数量j而言，因为j至少要满足前i个月的生产需求D[i]，故j取值范围为D[i]到D[n]。若j大于D[i]，则多生产的设备需要花费5\*(j- D[i])来储存。这样设计数据结构的好处是每种情况下的保存成本都可以很容易地得到，非常便利。雇佣成本的计算需要分别讨论。当i为1，也就是第一个月时，j代表前1个月共生产j台设备，因而对于第一个月，p矩阵的p[i][j]存储的就是j。且由于不存在之前多生产的设备，故第一个月的生产数量只能大于等于当月需求。而对于i>1的情况，设当月生产k台设备，则k的取值范围为0台到j-D[i-1]台。则花费就相当于前i-1个月生产j-k台设备的花费加上本月的花费，即cost[i][j]等于cost[i-1][j-k]加本月的花费。遍历k的取值，如果cost[i][j]小于之前的cost[i][j]，则更新cost[i][j]取值，p[i][j]更新为k。

核心代码如下：

for i in range(n):  
 for j in range(D[i], D[n-1]+1):  
 if(i == 0):  
 cost[i][j] = 5\*(j-D[0])  
 if(j > m):  
 cost[i][j] = cost[i][j]+c\*(j-m)  
 p[i][j] = j  
 else:  
 cost[i][j] = 9999  
 for k in range(0, j-D[i-1]+1):  
 t = cost[i-1][j-k] + 5\*(j-D[i])  
 if(k > m):  
 t = t+c\*(k-m)  
 if(t < cost[i][j]):  
 cost[i][j] = t  
 p[i][j] = k

1. 递归遍历

def func1(p, i, j, sss):

if(i >= 0):

a = int(j-p[i][j])

func1(p, i-1, a, sss)

sss.append(p[i][j])

如以上代码所示，

## 递推公式，数学抽象

设生产计划总共n个月，D[i]表示前i个月总需求，第i个月最小花费：

,

其中表示前i个月总共生产j台设备的前提下的最小成本，表示前i-1个月总共生产k台设备的前提下的最小成本， 表示第i个月生产的花费，表示在k遍历后得到的最小成本。

设第i个月生产x台设备，前i个月的库存为，每月公司最多自主生产m台。

若，则；

若，则

## 最优子结构证明

设长度n数组result为它存储n个月生产规划问题最优解，则result[1:n]表示，它为前n-1个月生产规划问题最优解。

假设result1才是前n-1个月生产规划问题最优解，根据公式

,

可得 生产花费小于 ，即

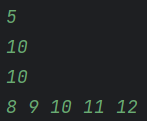
与题目设定 为最优解相矛盾。由上，问题得证。

## 时空复杂度

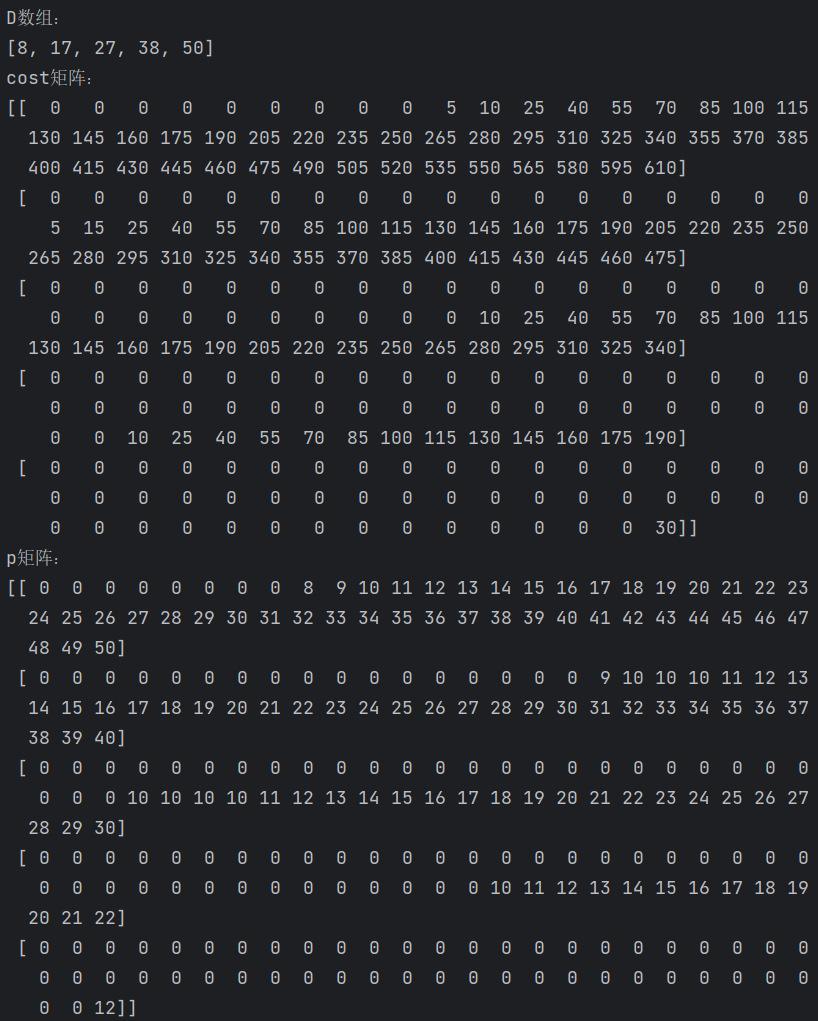
1. 时间复杂度：两层嵌套循环，共O(nD)；递归迭代需要O(D)；时间复杂度总共O(nD²)
2. 空间复杂度：两个n行D列的矩阵，共O(nD)

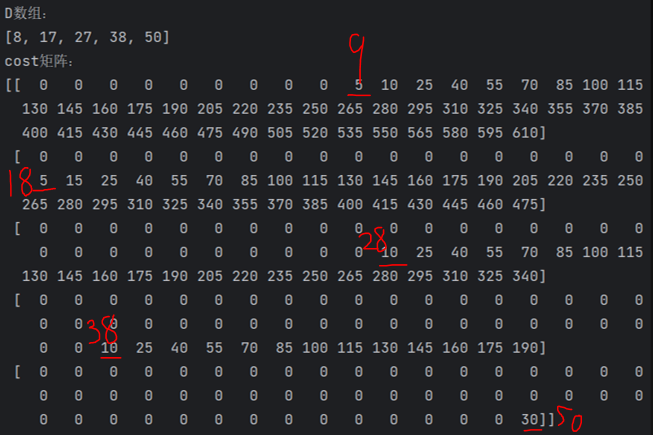
## 例子

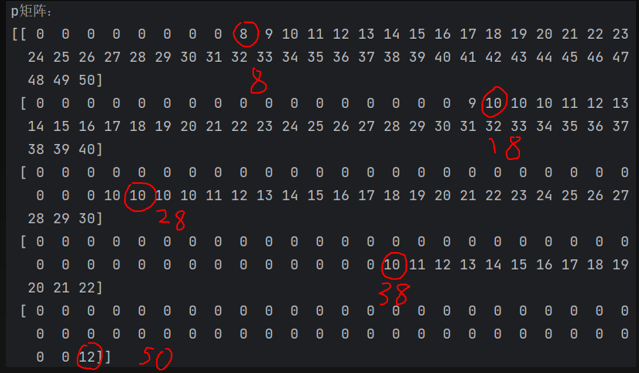
输入：



各数组情况：







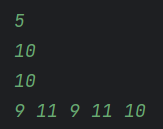
以上是根据D数组进行的选择结果

输出结果：

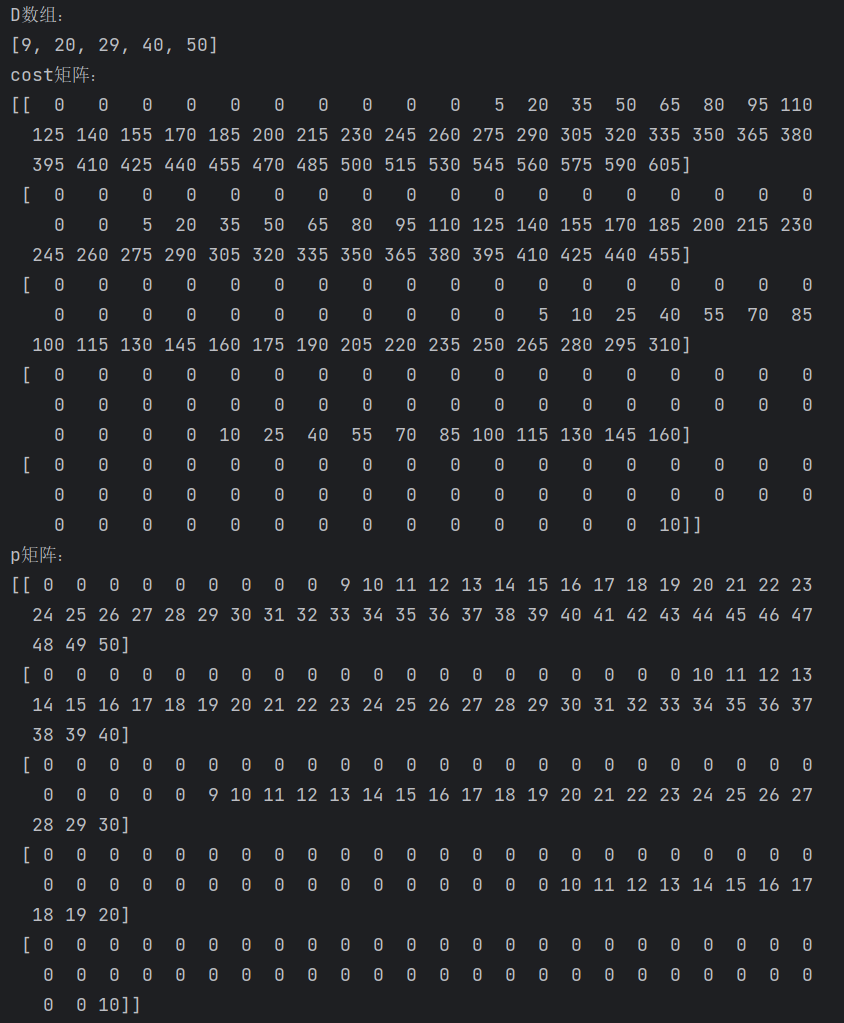


例子二：

输入：



各数组情况：



输出结果：



## 代码展现

import numpy as np

import sys

def func(n, d, m, c):

D = [0] \* n

cost = np.zeros((n, sum(d)+1),dtype = int)

p = np.zeros((n, sum(d)+1),dtype = int)

D[0] = d[0]

for i in range(1, n):

D[i] = D[i - 1] + d[i]

for i in range(n):

for j in range(D[i], D[n-1]+1):

if(i == 0):

cost[i][j] = 5\*(j-D[0])

if(j > m):

cost[i][j] = cost[i][j]+c\*(j-m)

p[i][j] = j

else:

cost[i][j] = 9999

for k in range(0, j-D[i-1]+1):

t = cost[i-1][j-k] + 5\*(j-D[i])

if(k > m):

t = t+c\*(k-m)

if(t < cost[i][j]):

cost[i][j] = t

p[i][j] = k

return p

def func1(p, i, j, sss):

if(i >= 0):

a = int(j-p[i][j])

func1(p, i-1, a, sss)

sss.append(p[i][j])

n = int(sys.stdin.readline())

m = int(sys.stdin.readline())

c = int(sys.stdin.readline())

d = list(map(int, sys.stdin.readline().split(" ")))

p = func(n, d, m, c)

sss = []

func1(p, n-1, sum(d), sss)

for i in range(len(sss)):

if(i != len(sss) - 1):

sys.stdout.write(str(sss[i]))

sys.stdout.write(" ")

else:

sys.stdout.write(str(sss[i]))