Report For Investment Problem

09019204
曹邹颖

1. Problem description / demand analysis 问题描述

You are granted a project in which you need to do some archaeological excavation. There are four different places that you can dig, however you only have 10 days for the excavation. The expected rewards for each day are shown in the following table.

Place Days	1	2	3	4
0	0	0	0	0
1	10	3	4	1
2	15	9	6	10
3	16	17	9	13
4	17	26	13	16
5	30	28	15	20

(1) Design an efficient algorithm to plan the number of days to spend on each place to maximize the reward. The output should be the following.
Place 1, spend X days
Place 2, spend X days
Place 3, spend X days
Place 4, spend X days
Total profit = XX2

(2) Let P number of places, and D the number of total days. What is the complexity time of the algorithm?

2. System structure / algorithm idea 算法设计

(1) basic idea 基本思想

我将这个问题看作一个分组背包问题，其中 the number of total days 看作背包的容量，Days 看作物品的 weight，挖到的 reward 看作物品的 value，Place 看作物品的分组，每个 Place 可以选择挖的天数从 0 到+∞，由于挖 0 天的 reward 为 0 以及挖＞5 天的 reward 与挖 5 天的 reward 一致，本着最大化 reward 的目的，即将问题简化为每个 Place 可以选择挖的天数从 1-5，看作每组有 5 个物品，选择一个放入背包，最大化背包的价值问题。

(2) system framework 算法框架

算法分为背包物品选择与路径打印两部分。

背包物品选择部分还是与 01 背包相似，这里我采用一维数组，状态转移方程还是 profit[j] = max(profit[j - weight[i]] + reward[i] , profit[j])，只不过需要去枚举第 i 组物品，考虑选哪一个物品时最优的(或者不选)。

路径打印部分，采用先组内编号再组外编号的方法把物品标号，用状态矩阵path[i][j]记录状态 j 下物品 i 是否被选中。path[i][j] = 1 表示被选，初始化为全 0。打印时，从最后一个状态往前找，这样就可以得到每个 Place 所挖的天数。

(3) the functions and relationships of each module 模块的功能和关系

路径需要在背包物品选择部分记录下来，即枚举第 i 组物品，选择最优的一个物品时在他的序号位置下置 path=1，而打印部分是与背包物品选择模块分开的。

3. Function module design 功能模块设计

(1) module design idea

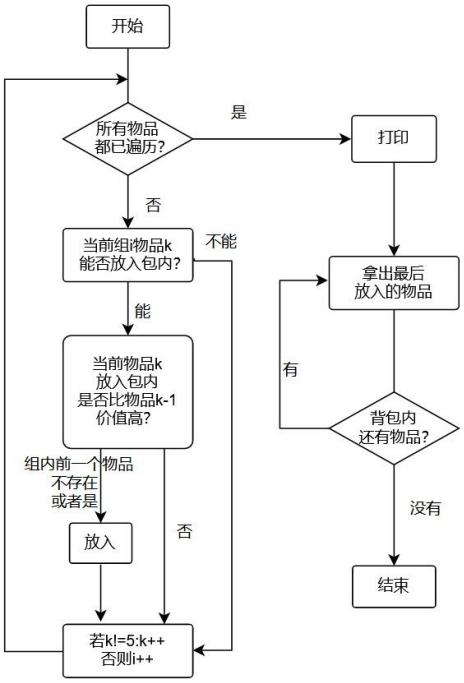
背包物品选择部分(包含记录路径)：

```
for 所有的组 i
    for j=d...0
        for 所有的 k 属于组 i
            if j>=k+1
                profit[j]=max{profit[j],f[v-(k+1)]+reward[5*i+k]}
                choose=5*i+k
        path[choose][j]=1
```

路径打印部分(用 spend 数组记录每个 Place 挖的天数)：

```
初始 w 为总共能挖的天数；
while 所剩挖的天数 w 不为 0
    for 所有的组 i（倒序）
        for 所有的 j 属于组 i（组内序号）
            if path[i*5+j][w]==1: spend[i] = j + 1;w -= j + 1;break;
```

(2) flow chart



(3) algorithm complexity analysis

设 P: number of places，D: the number of total days

根据算法中最复杂的为三层 for 循环（具体见 Report 4./6.），可见算法复杂度为 O(5PD)=O(PD)。

4. Test results and analysis 测试结果和分析

(1) test data selection or generation method 测试数据选择/生成方法

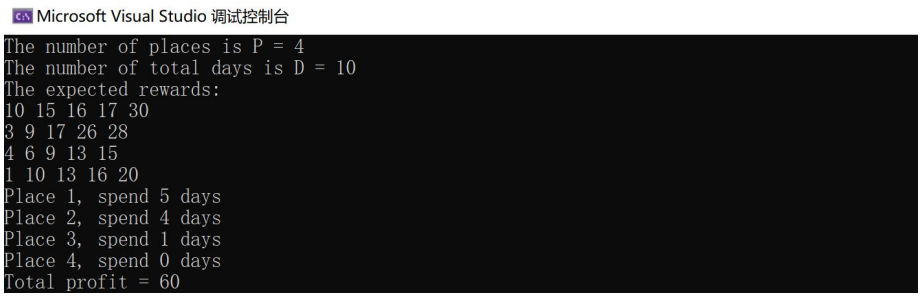
P: number of places, and D: the number of total days

本题第一小问：P=4，D=10

Reward:

Place Days	1	2	3	4
0	0	0	0	0
1	10	3	4	1
2	15	9	6	10
3	16	17	9	13
4	17	26	13	16
5	30	28	15	20

(2) operation result screenshot 运行结果截屏

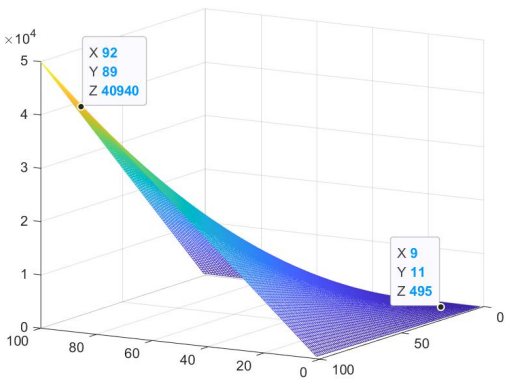


(3) performance diagram 性能图

本题第二小问：Let P number of places, and D the number of total days. What is the complexity time of the algorithm?

答：O(5PD)=O(PD)

```
>> [P,D]=meshgrid(1:1:100,1:1:100);
>> Z=5*P.*D;
>> mesh(P,D,Z)
```



5. Experimental summary 实验总结

(1) the problems encountered 遇到的问题

问题：

一开始将其看作简单 01 背包问题求解，会发现求出的方案 Place 中有挖两次（两次天数累积>5）的情况。

(2) the problem-solving process 解决方案

解决方案：

将其看作分组背包问题求解，每个 Place 看作一个组，每组只选择 1-5 天数中的一个 weight。

(3) summarize the experimental experience 实验总结

通过这一实际问题，将其转化为背包问题，掌握动态规划的基本思想，了解到动态规划的背包问题多种类型，并能设计相应的算法，同时掌握动念规划时间空间复杂度分析，以及问题复杂性分析方法。但是，针对背包问题这一典型的组合优化问题，在计算理论中属于 NP-完全问题，传统上的动态规划求解方法复杂度还是很高的，我并没有尝试其他非传统方法来提高解决问题的效率这是有遗憾的。

6. Source code 源代码

```
#include<iostream>
using namespace std;
int main()
{
    int p,d,choose;                    // P:number of places,D:the number of total days,
                                       // choose:the index of choosed item
    cout << "The number of places is P = ";cin >> p;
    cout << "The number of total days is D = ";  cin >> d;
    int* spend = new int[p];           //  最后结果：存放每个 place 挖的天数
    int* reward = new int[p * 5];      //  每个 place 挖的天数对应的 reward:物品的 value
    int* profit = new int[d];          // profit[j]:总共挖 j 天获得的最大利润
    int** path = new int* [(p * 5)];   // path[i][j]:背包容量为 j 时物品 i 是否放入（0/1）
    for (int i = 0; i < p * 5; i++)
        path[i] = new int[(d + 1)];
    cout << "The expected rewards: " << endl;
    for (int i = 0; i < p * 5; i++)    cin >> reward[i];
    for (int i = 0; i < p; i++)  spend[i] = 0;
    for (int i = 0; i < d; i++)  profit[i] = 0;
    for (int i = 0; i < p * 5; i++)
        for (int j = 0; j <= d; j++)
            path[i][j] = 0;
    for (int i = 0; i < p; i++)                  // p 组物品
        for (int j = d; j >=0; j--){             //  背包容量 j
            for (int k = 0; k < 5; k++)          //  每组 5 个物品，枚举
                if (j >= k + 1)
```

```
                if (profit[j] < (profit[(j - k - 1)] + reward[(5 * i + k)])) {
                    profit[j] = profit[(j - k - 1)] + reward[(5 * i + k)];
                    choose = 5 * i + k;
                }
            path[choose][j] = 1;
        }
    int w = d;
    while (w)                                   // 背包容量 w：剩余挖的天数
        for (int i = p - 1; i >= 0; i--)        // 物品的组号
            for (int j = 0; j < 5; j++)         // 物品组内编号
                if (path[i * 5 + j][w])         // ==1：组内选的物品标号
                {
                    spend[i] = j + 1;           // Place i+1 所挖天数
                    w -= j + 1;
                    break;
                }
    for (int i = 0; i < p; i++)
        cout << "Place " << i + 1 << ", spend " << spend[i] << " days\n";
    cout << "Total profit = " << profit[d] << endl;
    for (int i = 0; i < p * 5; i++)
        delete[] path[i];
    return 0;
}
```