算法设计与分析 Week 8

第八次作业-solution

Lecturer: 杨启哲 Last modified: 2025 年 11 月 24 日

1. 对于任意给定的正整数 M, 请给出三个矩阵相乘的例子，它们的一种乘法顺序所需要的数量乘法的次数至少是另一种顺序的 M 倍。

   解答. 考察如下三个矩阵:

   $$A_{1\times M},\ B_{M\times 1},\ C_{1\times M}$$

   则我们有:

   - $A(BC)$ 的乘法次数为 $1\times M\times M+M\times 1\times M=2M^2$
   - $(AB)C$ 的乘法次数为 $1\times M\times 1+1\times 1\times M=2M$

   两种顺序刚好差 M 倍的乘法计算次数。 □

2. 回顾贪心算法中我们讲到的货币兑换问题。有一个货币系统，它有 n 种硬币，面值分别为 $v_1=1, v_2, \ldots, v_n$。现在需要兑换价值为 y 的钱，使得硬币的数量最少。之前提到过这个问题贪心算法不一定能给出最优解，现在请设计一个高效的算法解决这个问题。

   解答. 我们利用动态规划的思想来解决这个问题。记 $C[i][y]$ 为用前 i 种硬币兑换价值为 y 的钱所需要的最少硬币数，则我们有如下递推关系:

   $$C[i][y]=\min\{C[i][y-v_i]+1, C[i-1][y]\}$$

   注意到硬币是可以重复使用的，所以我们并不需要考虑 $C[i][y-v_i]$ 是否已经用过第 i 枚硬币。具体算法如下:

   兑换钱币

   输入: 面值列表 $v_1, \ldots, v_n$，兑换的钱数 y
   输出: 最少的硬币数

   ```
   for i = 1 to n do
       C[i][0] ← 0
   end for
   for j = 1 to y do
       C[0][j] ← ∞
   end for
   for i = 1 to n do
       for j = 1 to y do
   ```

```
        if j < v_i then
            C[i][j] ← C[i − 1][j]
        else
            C[i][j] ← min{C[i][j − v_i] + 1, C[i − 1][j]}
        end if
    end for
end for
return C[n][y]
```

算法需要一个 $n \times y$ 的表格来存储中间结果并对其进行遍历，所以时间复杂度何空间复杂度均为 $O(ny)$。 □

Remark 0.1

这实际上就是一个多重背包问题，事实上我们也可以对其做空间的优化。具体来说，我们可以只用一个长度为 y 的数组来存储中间结果，具体算法如下：

兑换钱币

输入: 面值列表 $v_1, \ldots, v_n$，兑换的钱数 y
输出: 最少的硬币数

```
for j = 1 to y do
    C[j] ← ∞
end for
C[0] ← 0
for i = 1 to n do
    for j = v_i to y do
        C[j] ← min{C[j − v_i] + 1, C[j]}
    end for
end for
return C[y]
```

3. 假设你现在有 n 枚硬币，其正面朝上的概率分别为 $p_1, p_2, \ldots, p_n \in [0, 1]$。请设计一个 $O(n^2)$ 时间的算法，计算正面朝上的硬币数恰好为 k 的概率（硬币之间是相互独立的），这里假设对两个 $[0, 1]$ 区间内的数进行加法和乘法的运算时间为 $O(1)$。

解答. 我们考虑如下的子问题，

子问题: 设 $P[i][j]$ 为前 i 枚硬币中正面朝上的硬币数恰好为 j 的概率。则我们有如下递推关系：

$$P[i][j] = P[i-1][j-1] \cdot p_i + P[i-1][j] \cdot (1 - p_i)$$

具体算法如下：

正面朝上硬币数

输入: 硬币数 n，正面朝上的概率 $p_1, \ldots, p_n$，正面朝上的硬币数 k
输出: 正面朝上的硬币数恰好为 k 的概率

```
for i = 0 to n do
    P[i][0] ← ∏_{t=1}^{i} (1 − p_t)
end for
for j = 1 to k do
    P[0][j] ← 0
end for
for i = 1 to n do
    for j = 1 to min{i, k} do
        P[i][j] ← P[i − 1][j − 1] · p_i + P[i − 1][j] · (1 − p_i)
    end for
end for
return P[n][k]
```

算法需要一个 $n \times k$ 的表格来存储中间结果并对其进行遍历，所以时间复杂度为 $O(nk)$。由于 $k \leqslant n$，所以时间复杂度为 $O(n^2)$。 □

4. 为了降低背包问题运行的时间界限 $O(nW)$，我们尝试用这样的思路，我们使用一个大数 K 将容量 W 和每件物品的体积 w_i 都缩小 K 倍。具体来说，令 $W \leftarrow \lceil \frac{W}{K} \rceil$，$w_i \leftarrow \lceil \frac{w_i}{K} \rceil$；再在这个背包问题的新实例上运行我们提出的解决背包问题的算法。

   (1) 现在算法的时间复杂度是多少?
   (2) 这样子的策略其实并不能总得到原实例的最优解，请给出一个反例。

(1) 现在算法的时间复杂度是多少?
(2) 这样子的策略其实并不能总得到原实例的最优解，请给出一个反例。

解答.

- 回顾一下背包问题的算法:

背包问题

输入: n 个物品，重量分别为 $w_1, \ldots, w_n$，价值分别为 $v_1, \ldots, v_n$，背包容量为 W
输出: 最大价值

```
for j = 0 to W do
    V[j] ← 0
end for
for i = 1 to n do
    for j = W to w_i do
```

```
        if j ⩾ w_i then
            V[j] ← max{V[j], V[j − w_i] + v_i}
        end if
    end for
end for
return V[W]
```

如果将 $W \leftarrow \lceil \frac{W}{K} \rceil$，$w_i \leftarrow \lceil \frac{w_i}{K} \rceil$，不难发现算法中的循环总共执行了 $\lceil nW/K \rceil$ 次，从而算法的时间复杂度为 $O(nW/K)$。

- 考虑如下的反例：
  - $W = 10$，物品重量为 $w_1 = 9, w_2 = 1$，价值分别为 $v_1 = 1, v_2 = 10$。

  如果 $K = 2$，则我们有：
  - $W = 5$，物品重量为 $w_1 = 5, w_2 = 1$，价值分别为 $v_1 = 1, v_2 = 10$。

  在原始问题中，最优解是两个物品全取，总价值为 11。但在更改后的问题中，最优解是只取第二个物品，总价值为 10。

□

> Remark 0.2
>
> 需要注意的是,K 并不是一个常数，所以计算复杂性的时候不能像省略常数一样省略。

5. 这个问题是对贪心算法解决背包问题的一个探索：

   (1) 请给出一个反例，说明按照价值排序然后贪心选择并不能总得到最优解。

   (2) 请给出一个反例，说明按照单位重量价值排序然后贪心选择并不能总得到最优解，并且该解与最优解的比值可以任意大。

   解答.

   (1) 考虑如下的反例：
   - 背包容量 $W = 10$，物品重量分别为 $w_1 = 9, w_2 = 8, w_3 = 2$，价值分别为 $v_1 = 10, v_2 = 9, v_3 = 2$。

   按照价值排序后，贪心算法会先选择物品 1，然后背包就满了，总价值为 10。但最优解是选择物品 2 和物品 3，总价值为 11。

   (2) 考虑如下的反例：
   - 背包容量 $W = X$，物品重量分别为 $w_1 = 2, w_2 = X - 1$，价值分别为 $v_1 = 3, v_2 = X - 1$。

   按照单位重量价值排序后，贪心算法会先选择物品 1，然后就无法选择第二件物品，总价值为 3。但最优解是选择物品 2，总价值为 $X - 1$。并且注意到 X 是任意的，所以当 X 趋近于无穷大时，最优解与贪心解的比值趋近于无穷大。

□

Remark 0.3

这个例子说明，无论是按价值的贪心，还是按单位重量价值的贪心，都不能保证得到背包问题的最优解，甚至与最优解的比值可以任意大。