算法设计与分析 LECTURE 9

第十周作业-solution

LECTURER: 杨启哲

LAST MODIFIED: 2023 年 12 月 25 日

1. (教材习题 6.11) 请给出三个矩阵的例子，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 100 倍。

   解答. 考察如下三个矩阵：

   $$A_{10\times 1000},\ B_{1000\times 5},\ C_{5\times 500}$$

   则我们有：

   - $A(BC)$ 的乘法次数为 $10\times 1000\times 500+1000\times 5\times 500=7500000$
   - $(AB)C$ 的乘法次数为 $10\times 1000\times 5+10\times 5\times 500=75000$

   两种顺序刚好差 100 倍的乘法计算次数。 □

2. (教材习题 6.29) 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. (最长回文子序列) 如果一个子序列从左向右和从右向左读都一样，则称之为回文。例如，序列：

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

有很多回文子序列，如 A, C, G, C, A 和 A, A, A, A。请设计一个 $O(n^2)$ 时间的算法，对于输入一个长度为 n 的序列，找出其最长回文子序列的长度。

解答. 我们可以利用动态规划的思想来解决这个问题。x 需要注意的是，因为是回文子序列，所以并不一定连续。因此记 $L[i][j]$ 为序列 $A[i], \ldots, A[j]$ 的最长回文子序列的长度，则我们有如下递推关系：

$$L[i][j] = \begin{cases} 1 & i = j \\ 2 & i = j - 1, A[i] = A[j] \\ L[i+1][j-1] + 2 & i < j - 1, A[i] = A[j] \\ \max\{L[i+1][j], L[i][j-1]\} & i < j, A[i] \neq A[j] \end{cases}$$

根据上述递推关系可以构造如下算法：

(1) 初始化 $L[i][i] = 1$ 和所有的 $L[i][i+1]$。
(2) 根据 $L[i][j]$ 中 $j - i$ 的大小从 2 到 $n - 1$ 开始循环，依次根据上述关系更新每组的 $L[i][j]$。

简单的伪代码如下：

最长回文子序列

输入: 序列 $A[1,\ldots,n]$
输出: 最长回文子序列的长度

```
1: for i = 1 to n do
2:     L[i][i] ← 1
3: end for
4: for i = 1 to n − 1 do
5:     if A[i] = A[i + 1] then
6:         L[i][i + 1] ← 2
7:     else
8:         L[i][i + 1] ← 1
9:     end if
10: end for
11: for k = 2 to n − 1 do
12:     for i = 1 to n − k do
13:         j ← i + k
14:         if A[i] = A[j] then
15:             L[i][j] ← L[i + 1][j − 1] + 2
16:         else
17:             L[i][j] ← max{L[i + 1][j], L[i][j − 1]}
18:         end if
19:     end for
20: end for
21: return L[1][n]
```

算法需要进行 $n \times n$ 的一个循环，每次循环的操作次数是常数次，所以时间复杂度为 $O(n^2)$。 □