第十一周实验报告

沈家成

2017 年 11 月 29 日

1　1602 Merge Sorted Array

1.1　数列归并

数列归并就是将两个有序数列组合成一个有序数列，算法思路就是每次挑两个数列头小的元素，组成新的数列。用队列实现即可。需要注意的就是，一个数列为空的时候，要单独处理。

1.2　数列为空时

数列为空时，直接将另一个数组接上即可。

2　3002 Ruko Sort

2.1　问题本质

问题的本质就是去重排序，方法很多。考虑到数据量很小，选择用堆排序。

2.2 流程

先查找这个元素是否已经存在，从而达到去重的目的。不存在就插入优先级队列（极小堆），全部插入完成后，就完成了去重，并准备好了排序。最后再出队，就完成了排序。

3 1069 二哥的硬币

3.1 主要难点

这是一个动态规划的问题，主要难点在于理清递推关系，从而分而治之，降低难度。

3.2 初步思路

设计一个二维数组 dp[i][j]，表示前 i 种硬币能否配成 j, 这样，递推关系就可以表示成：

$$dp[i][j] = dp[i-1][j - kA_i] \ (\exists k, 0 \leq k \leq m)$$

这样，在 $n = 2, m = 5, A_1 = 1, A_2 = 4, C_1 = 2, C_2 = 1$ 的情况下，产生的动态规划二维数组如下：

$$\begin{array}{cccccccc} i/j & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 1 & 1 \end{array} \tag{1}$$

统计最后一行 1 的个数，就可以得到答案了。

但是，这种方法的时间复杂度为 $O(m\sum_i C_i)$，经过 OnlineJudge 的测试，超时，无法通过。

3.3 算法改进

反思之前的算法，每次只得出一个布尔值，损失了很多有用的信息，所以效率不高，想要优化，就要充分利用信息。

重新发掘 dp 的用处，将 dp[i][j] 定义为前 i 种硬币组成 j 时，第 i 种硬币还剩下多少，这样就能更精准地应对各种情况。

这样，递推关系就变成了：

$$dp[i][j] = \begin{cases} -1 & j < A_i, or\ dp[i][j-A_i], \\ C_i & dp[i-1][j] \geq 0, \\ dp[i][j-A_i] - 1 & else. \end{cases}$$

-1 表示前 i 种硬币不能组成 j，有两种情况。一是面值比 j 还大，只要加了 1 个就超过了 j；二是在配更小的数的时候，硬币已经用光了。

那么其他情况下，就是对于组成 $j - A_i$ 情况下，再用掉一个硬币。

这样，在 $n = 2, m = 5, A_1 = 1, A_2 = 4, C_1 = 2, C_2 = 1$ 的情况下，产生的动态规划二维数组如下：

$$\begin{array}{cccccccc} i/j & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & -1 & -1 & -1 & -1 & -1 \\ 1 & 2 & 1 & 0 & -1 & -1 & -1 \\ 2 & 1 & 1 & 1 & -1 & 0 & 0 \end{array} \tag{2}$$

但是这种情况下，二维数组从布尔类型变成了整型，这就意味着所需要的存储空间增加了 16 倍，造成了内存不够。所以还需要改进。

3.4 继续改进

时间上的问题解决了，接下来要解决空间上的问题。再观察递推公式：

$$dp[i][j] = \begin{cases} -1 & j < A_i, or\ dp[i][j - A_i], \\ C_i & dp[i-1][j] \geq 0, \\ dp[i][j - A_i] - 1 & else. \end{cases}$$

值得注意的是，每次递推只和 $dp[i][j - A_i], dp[i-1][j]$ 有关，因此摒弃二维数组，只需要保存一行的数组，这样就节省了空间。

4 1272 写数游戏

4.1 数据结构的选择

总数据量不超过 100，并且有排序的需求，因此选择散列表。

4.2 具体实现

设置一个规模为 100 的散列表，表示第 i 个数字是否存在。这样通过从大到小遍历就完成了排序，而且插入新元素对于排序并没有影响。

这样，每次从大到小遍历，如果商不存在则增加元素，并将标志位计 true，数组规模计数变量加一。当标志位在一次遍历中都没有被置 true，就说明已经没有新元素了，输出数组规模即可。