第三章作业

程远 2234412848

目录

1 最长单调递增子序列

1.1 算法设计

本问题类似于公共子序列问题。假设序列长度为 n，序列存储在数组 $a[]$ 中。定义数组 $l[]$ 和 $b[]$ 并初始化所有元素为-1。$l[i]$ 表示以 $a[i]$ 结尾的最长单调递增子序列长度，$b[i]$ 用于记录第 i 个元素前面的元素，以实现构造子序列。

算法流程如下：进行双重循环，外层循环遍历从 1 到 $n-1$ 的元素，记当前下标为 i。对于每一个 i，内层循环遍历下标从 0 到 $i-1$ 的元素，记当前下标为 j。若满足 $a[i] > a[j]$，说明 $a[i]$ 可以跟随在 $a[j]$ 后构成更长的子序列。此时更新

$$l[i] = \max(l[i], l[j] + 1)$$

并将 $b[i]$ 更新为 $a[j]$ 的值，表示 $a[i]$ 的前一元素为 $a[j]$。

最终，数组 $l[]$ 中的最大值即为最长单调递增子序列的长度，记为 $l[k]$。为了构造序列，从 k 开始逐步从 $b[]$ 中追溯前一个元素，直到遇到 $b[k] = -1$。反转构造出的向量即为所求的递增子序列。

1.2 复杂度分析

时间复杂度：主要复杂度来源于双重循环，外层循环遍历数组中的每一个元素，共执行 n 次操作。在每次外层循环中，内层循环从头遍历到当前元素之前的所有元素，最多需 n 次操作，则总的时间复杂度为：

$$O\left(\sum_{i=1}^{n} i\right) = O\left(\frac{n(n-1)}{2}\right) = O(n^2)$$

该时间复杂度来源于逐步更新每个位置的最长递增子序列长度和构造数组。

空间复杂度：该算法使用了两个辅助数组 $l[]$ 和 $b[]$，分别用于存储到每个位置为止的最长递增子序列长度以及追溯序列的信息。这些数组的长度均为 n，因此空间复杂度为 $O(n)$。

1.3 优化方法

该算法可以通过二分查找优化。步骤如下：1. 使用一个数组 $tails$，记录不同长度递增子序列的最小可能结尾值。2. 遍历每个元素，使用二分查找找到当前元素应插入的位置。3. 若当前元素大于 $tails$ 中的所有元素，则追加该元素；否则，将当前元素替换至相应位置。

这样，时间复杂度为 $O(n \log n)$，空间复杂度为 $O(n)$。

1.4 优化后复杂度分析

时间复杂度：通过二分查找将原算法的内层循环替换为 $O(\log n)$ 的查找过程，从而降低时间复杂度。优化后每个元素都只需执行一次二分查找，复杂度为 $O(\log n)$，则总的时间复杂度降为：

$$O(n \log n)$$

因为尾数组 $tails$ 的维护，每个元素最多被访问一次。

空间复杂度：优化后的算法仍需使用数组 $tails$ 记录不同长度递增子序列的最小结尾元素，$tails$ 长度最多为 n，因此空间复杂度仍为 $O(n)$。

2 漂亮打印

2.1 算法设计

假设总单词数为 n，单词长度存于数组 $word[]$ 中。定义 $extra[i][j]$ 为排版从第 i 个到第 j 个单词在一行后的多余空格数，$cost[i][j]$ 表示这种排版的代价。状态转移公式为：

$$cost[i][j] = \begin{cases} \infty, & \text{if } extra[i][j] < 0 \\ 0, & \text{if } j \text{ is the last word} \\ (extra[i][j])^3, & \text{otherwise} \end{cases}$$

定义 $dp[i]$ 为从第 1 个到第 i 个单词的最小排版代价，$break[]$ 记录换行位置。状态转移方程：

$$dp[i] = \min_{1 \le j \le i} (dp[j-1] + cost[j][i])$$

算法流程为：1. 外层循环从 1 到 n，内层循环从当前单词到第 n 个单词。2. 更新 $extra[i][j]$，计算 $cost[i][j]$，并根据方程更新 $dp[i]$ 和 $break[]$。最终 $dp[n]$ 即为最小代价，利用 $break[]$ 可以追溯构造漂亮打印方案。

2.2 复杂度分析

时间复杂度：时间复杂度涉及三个部分：多余空格数的计算，每行代价的计算，最优解的动态规划求解。算法包括三层嵌套循环。

1. 计算 $extra[i][j]$：首先计算单词 i 到单词 j 排成一行时的多余空格数。通过一个双重循环填充 $extra[i][j]$，$extra[i][j]$ 需要 $O(1)$ 时间计算（包含单词和空格总长度的简单运算），则构建整个 $extra$ 表的时间复杂度为：

$$O(n^2)$$

2. 计算 $cost[i][j]$：$cost[i][j]$ 的计算基于 $extra[i][j]$，需要检查每一对 i 和 j 以确定其代价，同样为：

$$O(n^2)$$

3. 计算最优解 $dp[i]$：最小代价求解需要通过每个 i 向前回溯，检查每一行的可能性。因为包含的内层循环需要 $O(n^2)$ 的复杂度，则该部分的时间复杂度为：

$$O(n^3)$$

则算法的总时间复杂度为 $O(n^3)$。

空间复杂度：算法需要使用 $extra[i][j]$ 和 $cost[i][j]$ 两个二维数组，每个数组的大小为 $n \times n$。同时，动态规划表 $dp[i]$ 和换行位置表 $break[i]$ 各需 $O(n)$ 空间。因此，空间复杂度为：

$$O(n^2)$$