1. 问题理解与算法思想

问题核心：在 N×N 的棋盘上放置 N 个皇后，要求任何两个皇后都不能处于同一行、同一列或同一对角线上。

核心算法：回溯法

对于 N 皇后问题，我的思路是：

逐行放置：从第 0 行开始，尝试为每一行放置一个皇后。

选择位置：在当前行（比如 row），我们从第 0 列开始，依次尝试到第 N-1 列。

冲突检测 (剪枝)：在尝试将皇后放置在 (row, col) 位置时，必须检查该位置是否与之前所有行（0 到 row-1）已放置的皇后发生冲突。

列冲突：新的皇后所在的列 col 是否已经被占用？

对角线冲突：新的皇后是否与之前的皇后在同一对角线上？

主对角线：对于两个皇后 (row1, col1) 和 (row2, col2)，如果 row1 - col1 == row2 - col2，则它们在同一主对角线。

副对角线：如果 row1 + col1 == row2 + col2，则它们在同一副对角线。

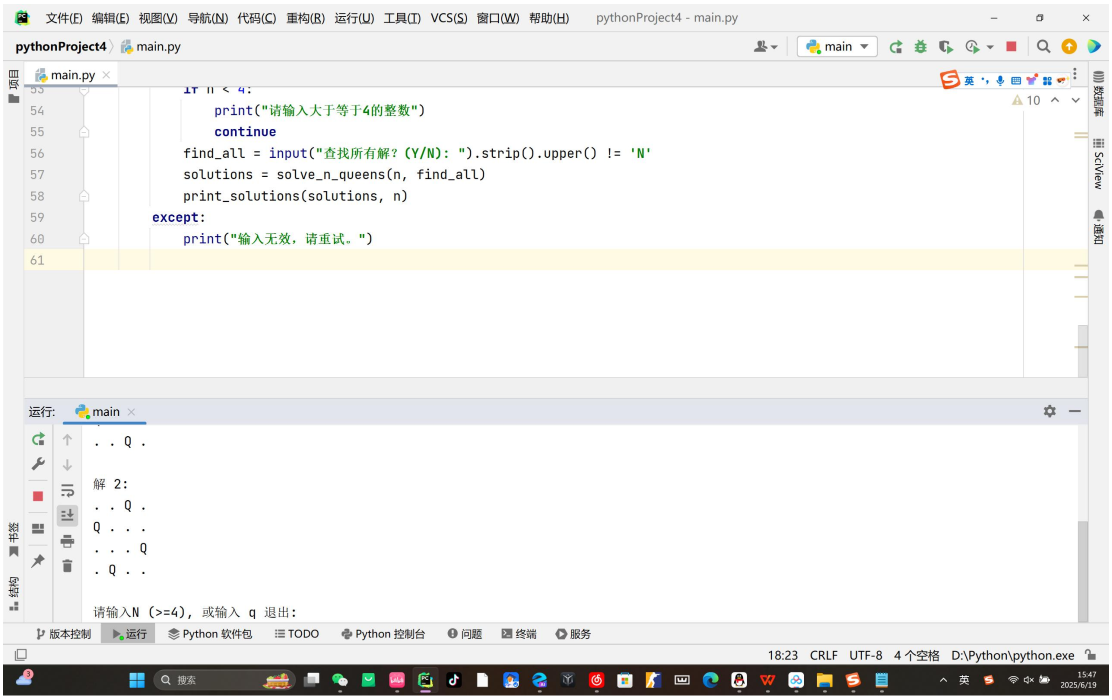
递归深入：如果 (row, col) 位置安全，我们就将皇后放在这里，然后递归地去解决下一行 (row + 1)。

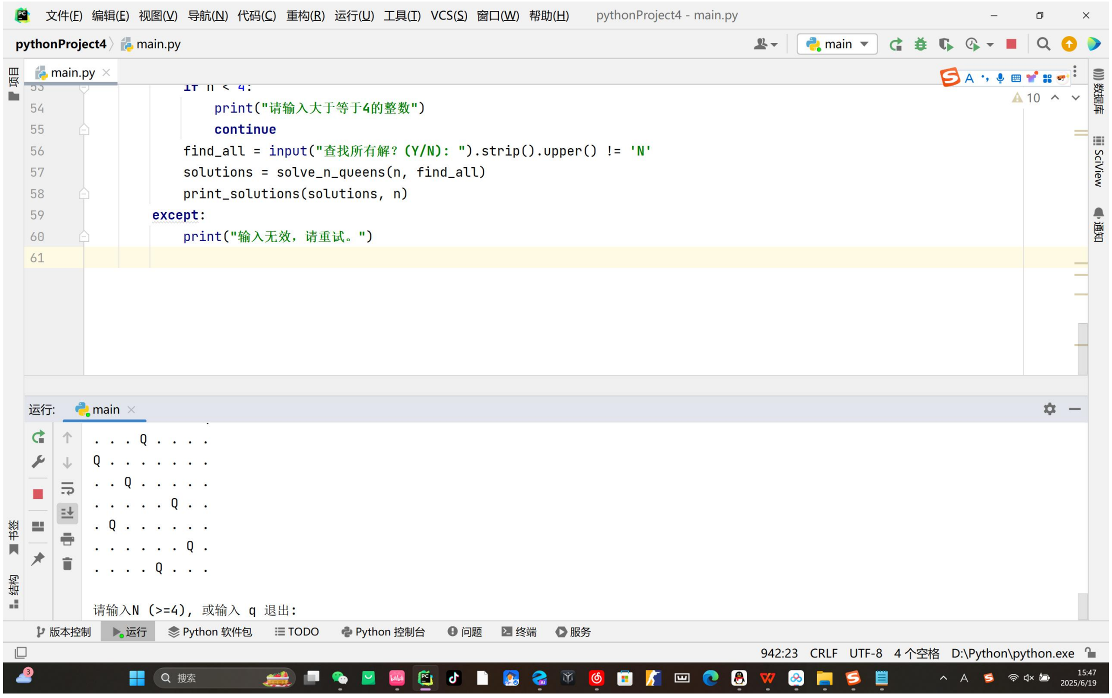
回溯：如果下一行的所有列都无法放置皇后，说明当前行的 (row, col) 选择是错误的。我们返回到当前行，撤销 (row, col) 的选择，并尝试当前行的下一个可用列。

找到解：当成功放置到第 N 行时（即 row == N），说明我们已经找到了一个完整的、有效的解。记录这个解。

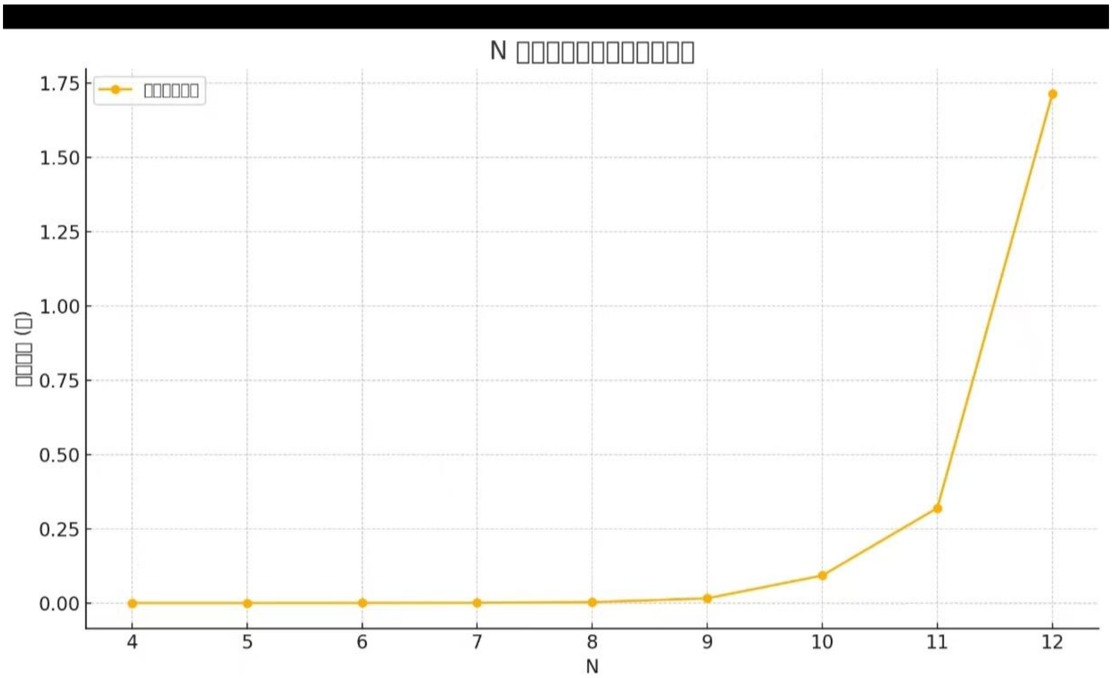
继续搜索：找到一个解后，不要停下来（除非用户只要求一个解）。我们应该继续回溯，以寻找所有可能的解

2 输出截图





运行时间



结论

实际运行时间曲线验证了回溯算法在 N 皇后问题中的指数级增长特征。

优化后的算法仍需遍历大量可能解，因此对大 N（如 N ≥ 15）效率下降明显。

对于超大规模的 N 皇后问题，需引入剪枝、并行计算或约束编程等方法进一步优化。