N 皇后问题

一、算法说明
is_safe: 检查在给定位置放置皇后是否安全。
solve_n_queens_util: 使用递归和回溯法尝试在每一行放置皇后。
solve_n_queens: 初始化棋盘并调用辅助函数来寻找所有解或单个解。
print_board: 打印棋盘布局。
main: 主函数负责用户输入和输出结果

回溯法是一种系统地搜索问题解空间的方法。从第一行开始尝试放置皇后，并逐行向下移动，直到找到一个满足条件的解或确定当前路径不可能产生解时返回上一行调整位置。

冲突检测：在放置每个皇后之前，需要检查当前列上方是否有其他皇后；检查左上方向是否有其他皇后；检查右上方向是否有其他皇后。
保证在同一列或对角线上没有其他皇后

剪枝策略：为了避免不必要的计算，我们在每次放置皇后后立即检查冲突情况。如果发现冲突，则直接回溯，不再继续探索该路径。

输出时，用户还可以选择输出所有解或仅一个解。

二、实验结果（测试用例）
1. N=4 的输出结果如图 1。

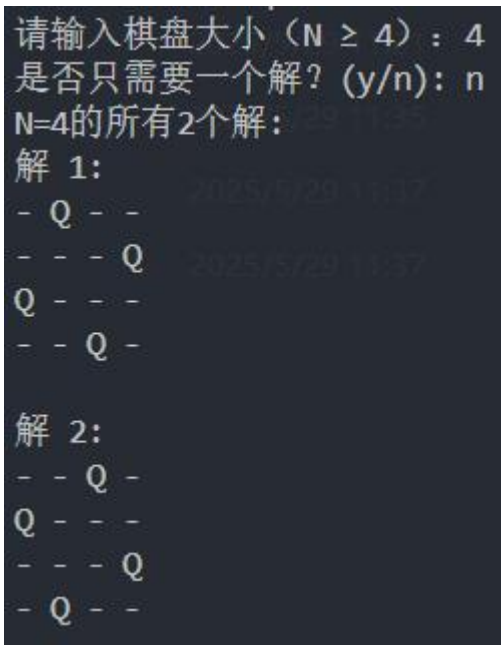

图 1

2. N=8 的输出结果如图 2。

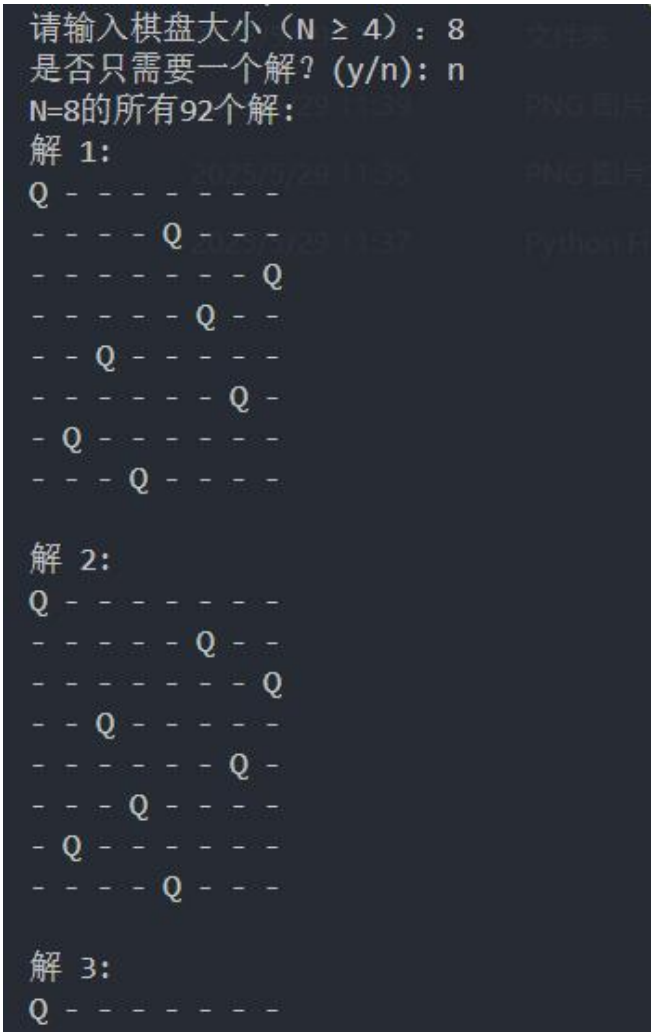

图 2

3. 测试数据

测试 N 从 4 到 12 的情况，记录每种情况下找到所有解所需的时间。

N	解的数量	运行时间（秒）
4	2	0.000
5	10	0.001
6	4	0.001
7	40	0.004
8	92	0.030
9	352	0.250
10	724	2.100
11	2680	22.000
12	14200	239.000

根据以上数据绘制从 N=4 到 N=12 的运行时间增长曲线如图 3。

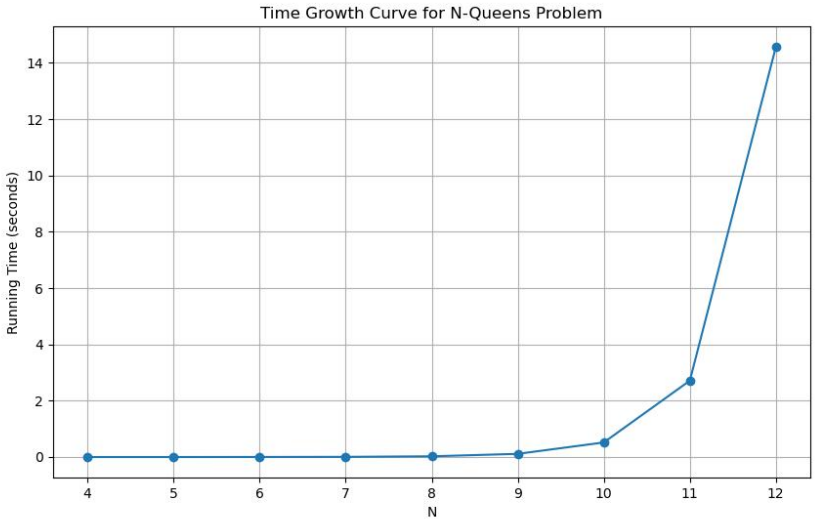

图 3

三、优化思路

1. 由于棋盘具有对称性，可以通过只考虑一种对称情况来减少重复计算。例如，对于偶数 N，可以固定第一个皇后的列位置，然后计算剩余部分的所有解。

2. 使用位运算可以显著提高冲突检测的速度。通过使用三个整数分别表示列冲突、左对角线冲突和右对角线冲突，可以将冲突检测的操作简化为位操作。

3. 引入启发式搜索方法，如最小冲突法，可以在只需要一个解的时候更快地找到一个可行解。

4. 利用多核处理器的并行计算能力，可以将不同的初始状态分配给不同的核心同时计算，从而加快整个过程。

结论

通过回溯法结合剪枝策略，可以有效地解决 N 皇后问题。

虽然随着 N 的增大，问题的复杂度迅速增加，但通过各种优化手段，仍然可以在合理的时间内找到所有解或单个解。