N 皇后问题求解程序算法说明

一、问题描述

N 皇后问题是一个经典的回溯算法问题：在一个 N×N 的国际象棋棋盘上放置 N 个皇后，使得任意两个皇后不处于同一行、同一列或同一对角线上。程序的目标是求出所有可能的摆放方式，或仅求出其中一个解。

二、程序结构

该程序主要由以下几个部分组成：

1. is_safe(queens, row, col) 函数

功能：判断当前位置 (row, col) 是否安全，即是否可以放置皇后。

- 参数：

  - queens：当前棋盘状态，索引为行号，值为该行皇后所在列号。

  - row：当前尝试放置皇后的行号。

  - col：当前尝试放置皇后的列号。

- 返回值：True 表示当前位置安全，False 表示有冲突。

算法逻辑：

- 遍历已放置皇后的行；

- 判断是否同列或在同一对角线上；

- 如果发现冲突则返回 False。

2. solve_n_queens(N, find_one=False) 函数

功能：通过回溯算法求解 N 皇后问题。

- 参数：

  - N：棋盘大小。

  - find_one：布尔值，是否仅查找一个解。

- 返回值：一个二维列表，每个解是一个长度为 N 的列表，表示每行皇后的位置。

算法逻辑：

- 使用回溯方法尝试在每一行放置一个皇后；

- 对于每个列位置，调用 is_safe() 检查安全性；

- 若找到解则保存，若设置 find_one=True 则立即返回。

3. print_solution(sol) 函数

功能：以图形化方式输出一个皇后摆放解。

- 使用 'Q' 表示皇后，'.' 表示空位；

- 每行打印一个字符串表示当前棋盘状态。

4. run() 主程序

功能：程序交互入口，执行整体求解流程。

- 用户输入：

  - N 值（大于等于 4）；

  - 是否仅输出一个解；

- 调用 solve_n_queens() 求解；

- 输出所有解或一个解以及运行时间。

三、算法核心思想

回溯算法（Backtracking）

- 在每一行选择一个合法的位置放置皇后；

- 若当前行没有合法位置则回溯到上一行重新尝试；

- 若到达最后一行并成功放置，则找到一个解。

剪枝策略通过 is_safe() 函数实现，避免无效搜索路径，从而提高效率。

四、性能分析

- 时间复杂度：最坏情况下为 O(N!)，因为每一行有 N 种放置方式；

- 空间复杂度：O(N)，用于保存当前棋盘状态；

- 算法效率受 N 值影响显著，较大的 N 会有指数级增长的解空间。

五、输入输出示例


```
N = 4 的皇后问题 的一个解如下：

解 1:
. Q . .
. . . Q
Q . . .
. . Q .

共找到 1 个解。
运行时间：0.00000000 秒。
```

```
N = 12 的皇后问题 的一个解如下

解 1:
Q . . . . . . . . . . .
. . Q . . . . . . . . .
. . . . Q . . . . . . .
. . . . . . . Q . . . .
. . . . . . . . . Q . .
. . . . . . . . . . . Q
. . . . . Q . . . . . .
. . . . . . . . . . Q .
. Q . . . . . . . . . .
. . . . . . Q . . . . .
. . . . . . . . Q . . .
. . . Q . . . . . . . .

共找到 1 个解。
运行时间：0.01615238 秒。
```

六、优化思路

1. 位运算加速（Bitmask 优化）

当前算法通过列表和 is_safe() 函数判断列与对角线冲突，效率较低。

使用位掩码（bitmask）来记录哪些列、主对角线、副对角线已被占用，可显著提升性能，特别是在 N 较大时。

```
def solve_n_queens_bitmask(N):
    solutions = []
    def backtrack(row, cols, diag1, diag2, state):
        if row == N:
            solutions.append(state[:])
```

```
            return
        available = (~(cols | diag1 | diag2)) & ((1 << N) - 1)
        while available:
            p = available & -available  # 取最低位的 1
            col = (p - 1).bit_length()
            state.append(col)
            backtrack(row + 1, cols | p, (diag1 | p) << 1, (diag2 | p) >> 1, state)
            state.pop()
            available &= available - 1
    backtrack(0, 0, 0, 0, [])
    return solutions
```

2. 提前终止路径剪枝

原始代码的 is_safe 方法每次都循环遍历之前所有行的皇后位置进行检查，时间复杂度为 O(N)。我们可以使用 三个辅助集合：

cols: 标记某列是否已有皇后。

diag1: 标记主对角线（row - col）是否已有皇后。

diag2: 标记副对角线（row + col）是否已有皇后。

这样判断冲突只需 O(1) 时间，大大提升效率。

```
def solve_n_queens(n, find_one=False):
    def backtrack(row):
        if row == n:
            solutions.append(queens[:])
            return not find_one  # 找到一个解就终止（如果需要）
        for col in range(n):
            if col in cols or (row - col) in diag1 or (row + col) in diag2:
                continue
```

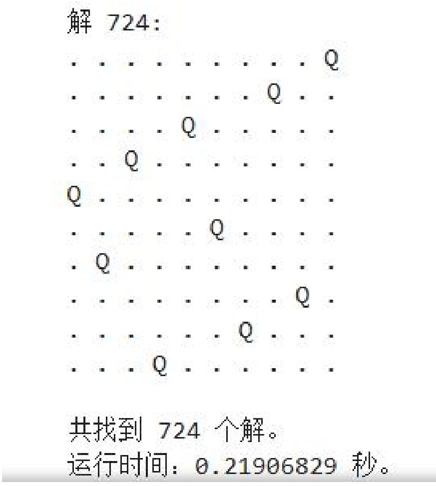
```
        # 放置皇后
        queens[row] = col
        cols.add(col)
        diag1.add(row - col)
        diag2.add(row + col)

        if not backtrack(row + 1) and find_one:
            return False  # 剪枝提前返回

        # 回溯撤销
        cols.remove(col)
        diag1.remove(row - col)
        diag2.remove(row + col)

    return True

queens = [-1] * n
solutions = []
cols = set()
diag1 = set()
diag2 = set()
backtrack(0)
return solutions
```



```
解 724:
. . . . . . . . . Q
. . . . . . . Q . .
. . . . Q . . . . .
. . Q . . . . . . .
Q . . . . . . . . .
. . . . . Q . . . .
. Q . . . . . . . .
. . . . . . . . Q .
. . . . . . Q . . .
. . . Q . . . . . .

共找到 724 个解。
运行时间：0.21906829 秒。
```

```
解 724:
. . . . . . . . . Q
. . . . . . . Q . .
. . . . Q . . . . .
. . Q . . . . . . .
Q . . . . . . . . .
. . . . . Q . . . .
. Q . . . . . . . .
. . . . . . . . Q .
. . . . . . Q . . .
. . . Q . . . . . .

共找到 724 个解。
运行时间：0.04764414 秒。
```

与原代码相比，运行效率确实显著提高。

3. 多线程/多进程加速

对于 N 较大的情况，可以对第一行的列分支进行并行处理（使用 multiprocessing 或 joblib）。

4. 增量解生成器

将解生成器 yield 化，支持流式生成和懒加载，避免一次性生成大量解而占用内存。

```
def solve_n_queens_generator(N):
    def backtrack(row):
        if row == N:
            yield queens[:]
        for col in range(N):
            if is_safe(queens, row, col):
                queens[row] = col
                yield from backtrack(row + 1)
    queens = [-1] * N
    yield from backtrack(0)
```