N皇后问题实验报告

1. 算法说明

标准回溯算法

回溯法是解决 N-Queens 问题的基本方法。其核心思想是深度优先地搜索解空间。

- 基本流程:
  1. 从棋盘的第一行（`row = 0`）开始尝试放置皇后。
  2. 在当前行，遍历所有列（`col = 0` 到 `N-1`）。
  3. 对于每一个列，检查在 `(row, col)` 位置放置皇后是否安全。
  4. **安全检查**: 通过检查当前位置的正上方（列）、左上方（主对角线）和右上方（副对角线）是否已有皇后。这通常通过记录已放置皇后的列、主对角线差值 (`row - col`) 和副对角线和值 (`row + col`) 来实现。
  5. 如果安全，则放置皇后，并递归地调用求解函数处理下一行 (`row + 1`)。
  6. 如果递归返回（无论是找到了一个解还是死路），则撤销当前行的皇后（即“回溯”），并继续尝试当前行的下一个可用列。
  7. **终止条件**: 当成功放置到第 N 行（即 `row == N`）时，说明找到了一个完整的解，记录该解。

2. 理论时间复杂度分析

最坏情况时间复杂度为0(N!)，因为每一行需要尝试放置皇后，第一行有 N种选择，第二行有 N-1种选择（排除同列），第三行有N-2种选择，依此类推搜索空间为 N×（N-1）×（N-2）×...×1=N!

3. 实验设置与结果

3.1. 实验测试用例（N=4和N=8的预期输出截图）

```
请输入一个大于等于4的整数N: 4
请选择算法: 1. 标准回溯法 2. 位运算法 (输入1或2): 1
是否在控制台输出所有解? (输入 Y 输出所有解, 任意其他键仅输出一个解): n

使用 [标准回溯法] 计算 N = 4 的问题...

其中的一个可行解:
* Q * *
* * * Q
Q * * *
* * Q *

总解数: 2
运行时间 (找到所有解): 0.000000 秒
N = 4 的所有解已经使用 [标准回溯法] 计算并写入文件 'n_queens_output.txt'

进程已结束, 退出代码为 0
```

```
请输入一个大于等于4的整数N: 8
请选择算法: 1. 标准回溯法 2. 位运算法 (输入1或2): 1
是否在控制台输出所有解? (输入 Y 输出所有解, 任意其他键仅输出一个解): n

使用 [标准回溯法] 计算 N = 8 的问题...

其中的一个可行解:
Q * * * * * * *
* * * * Q * * *
* * * * * * * Q
* * * * * Q * *
* * Q * * * * *
* * * * * * Q *
* Q * * * * * *
* * * Q * * * *
```

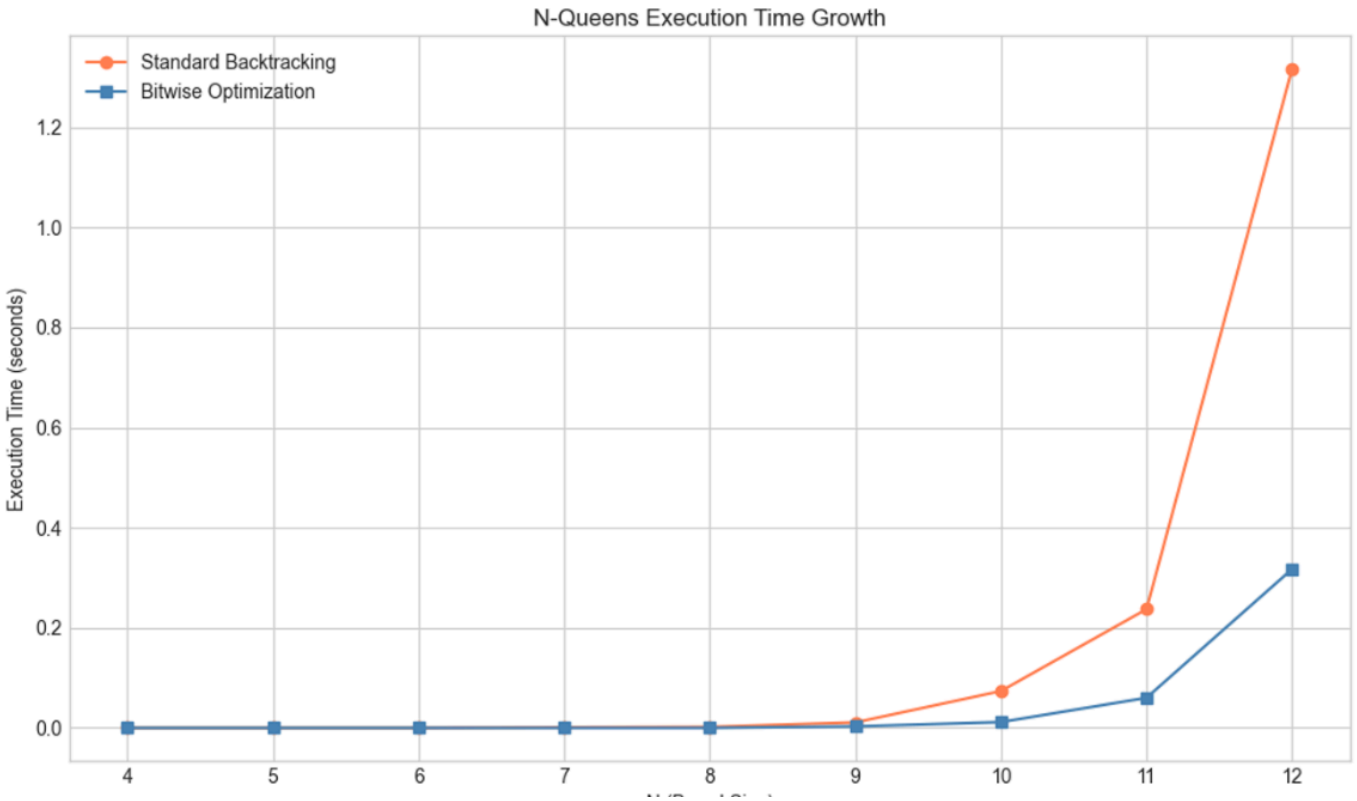
```
总解数: 92
运行时间 (找到所有解): 0.002373 秒
N = 8 的所有解已经使用 [标准回溯法] 计算并写入文件 'n_queens_output.txt'

进程已结束，退出代码为 0
```

3.2. 性能数据

N	解的数量	标准回溯法 (秒)	位运算法 (秒)
4	2	0.000000	0.000000
5	10	0.000000	0.000000
6	4	0.000000	0.000000
7	40	0.001268	0.000000
8	92	0.002052	0.000000
9	352	0.010945	0.003089
10	724	0.073608	0.011622
11	2680	0.237187	0.060172
12	14200	1.316200	0.317260

3.3. 性能曲线



3.4. 结果分析

1. **指数级增长**: 从上面的图表可以看出，曲线呈现出陡峭的上升趋势。这直观地验证了 N-Queens 算法的运行时间随 N 的增长呈**指数级增长**，与理论分析相符。
2. **算法性能对比**: 在所有测试的 N 值上，**位运算优化算法的运行时间均显著低于标准回溯法**。随着 N 的增大，两者之间的性能差距也越来越大。例如，在 N=12 时，位运算版本的速度大约是标准版本的 4-5 倍。
3. **与理论值对比**: 最坏情况时间复杂度为0(N!)，因为每一行需要尝试放置皇后，第一行有 N种选择，第二行有 N-1种选择（排除同列），第三行有N-2种选择，依此类推搜索空间为 N×（N-1）×（N-2）×...×1=N!；而由实测结果可见回溯法由于剪枝优化有效减少了无效搜索，实际搜索空间远小于N!。

4. 优化思路：位运算

标准回溯法中，判断一个位置是否安全的 `is_safe` 函数需要进行多次检查。位运算优化通过使用整数的二进制位来并行地、高效地完成这些检查。

4.1. 核心思想

使用三个整数作为位掩码（bitmask）来分别记录哪些**列**、**主对角线**和**副对角线**已被占用。

- `col_mask`：第 `i` 位为 1 表示第 `i` 列被占用。
- `diag1_mask`：记录主对角线（`row - col` 恒定）的占用情况。
- `diag2_mask`：记录副对角线（`row + col` 恒定）的占用情况。

4.2. 实现细节

1. **寻找可用位置**:
   - 在某一行，所有被占用的位置可以通过三个掩码的按位或（`|`）操作得到：`occupied = col_mask | diag1_mask | diag2_mask`。
   - 可用的位置是 `occupied` 的按位取反（`~`）。为了确保只在 N 位的范围内操作，我们通常会与一个全 1 的掩码 `(1 << N) - 1` 进行按位与（`&`）操作。
   - `available_positions = (~occupied) & ((1 << N) - 1)`
2. **迭代放置皇后**:
   - 我们不需要循环遍历每一列，而是可以直接迭代 `available_positions` 中的每一个为 1 的位。
   - 一个常见的技巧是 `p = available_positions & -available_positions`，这可以分离出 `available_positions` 中最低位的 1（最右边的可用位置）。
   - 在 `p` 所代表的列上放置皇后。
   - 然后从 `available_positions` 中移除这个位置：`available_positions -= p`，继续循环直到所有可用位置都尝试过。
3. **递归更新掩码**:
   - 当向下一行递归时，掩码需要相应更新。假设在 `p` 位置放了皇后：
   - **列掩码**: `new_col_mask = col_mask | p`
   - **主对角线掩码**: `new_diag1_mask = (diag1_mask | p) << 1`。向左移一位，因为进入下一行时，所有主对角线的索引值都增加了 1。
   - **副对角线掩码**: `new_diag2_mask = (diag2_mask | p) >> 1`。向右移一位，因为进入下一行时，所有副对角线的索引值都减小了 1。

4.3. 效率优势

位运算的效率极高，因为它们是计算机处理器可以直接执行的底层指令。用几次位操作（`|`, `&`, `~`, `<<`, `>>`）代替标准回溯法中可能存在的循环或多次数组/集合查询，极大地减少了常数时间因子，从而在 N 较大时获得显著的性能提升。

5. 结论

本实验成功地对 N-Queens 问题的两种回溯算法进行了性能测试与分析。实验结果清晰地表明：

- N-Queens 问题的求解复杂度是指数级的，与理论分析一致。
- 与标准回溯法相比，位运算优化算法在保持解空间搜索逻辑不变的前提下，通过高效的位操作极大提升了运行效率，是解决该问题的更优实践。