注意：本次实验全程是在python3.8版本下进行

在高版本kenren可能存在不兼容等问题

本代码一共有三个版本，效率由低到高（这个可能会在后续直接提到，所有提前说明）

1. **问题概述**
2. **问题的直观描述**

**8皇后问题是一个经典的组合优化问题，源自国际象棋。问题要求在一个8×8的棋盘上放置8个皇后，使得没有任何两个皇后能够互相攻击。根据国际象棋的规则，皇后可以攻击同一行、同一列或同一对角线上的任何棋子。**

**具体来说，8皇后问题的约束条件包括：**

* **每行只能放置一个皇后**
* **每列只能放置一个皇后**
* **每条对角线只能放置一个皇后**

**这里，我们不妨将8皇后问题推广到任意N值（也就是N皇后问题），这样有助于对问题进行由简到繁的分析。**

**对于N皇后问题，当N ≥ 4时才有解，且随着N的增大，解的数量呈指数级增长。**

1. **解决问题的思路和想法**

**解决N皇后问题需要采用有效的搜索和约束表示方法。在本实验中，根据作业要求，将使用约束逻辑编程的方法，具体采用Python的kanren库实现。**

**它的又是在于，只需描述问题的约束条件，而不必详细指定解决问题的具体步骤。系统会自动进行约束推理和传播，减少搜索空间。**

**针对N皇后问题，我的主要思路如下：**

1. **利用每列只能有一个皇后的特性，使用一个长度为N的列表表示解决方案，其中索引代表列号，值代表该列中皇后所在的行号**
2. **定义约束条件：行约束，列表中的值互不相同**

**对角线约束，任意两个皇后的行差不等于列差的绝对值**

1. **使用kanren库的逻辑变量和约束求解机制来搜索满足所有约束的解**
2. **利用问题的对称性来优化求解过程，减少计算量（此思路仅在最后一版代码中体现，这里约能节省一半的运行时间）**

**此外，在主要功能实现部分还是保持着以N皇后问题为主，以便于在出现问题时，进行调试。**

**和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你**

1. **算法设计**
2. **算法功能**

**本算法旨在通过约束逻辑编程的方法解决N皇后问题。在设计过程中，算法经历了多个版本的迭代优化，每个版本都针对特定的问题进行了改进。算法的主要功能包括：**

* **问题建模：将N皇后问题转化为约束满足问题**
* **约束求解：使用逻辑编程方法找出所有可行解**
* **解的优化：利用问题的对称性减少等方法计算量**
* **结果输出：输出所有解以及运算时间**

**版本1：基础约束求解**

**最初的设计采用直接的约束求解方法，主要功能包括：**

* **创建N个逻辑变量表示每列皇后的位置**
* **定义基本约束条件（行、列、对角线）**
* **使用kanren库的求解器找出所有解**

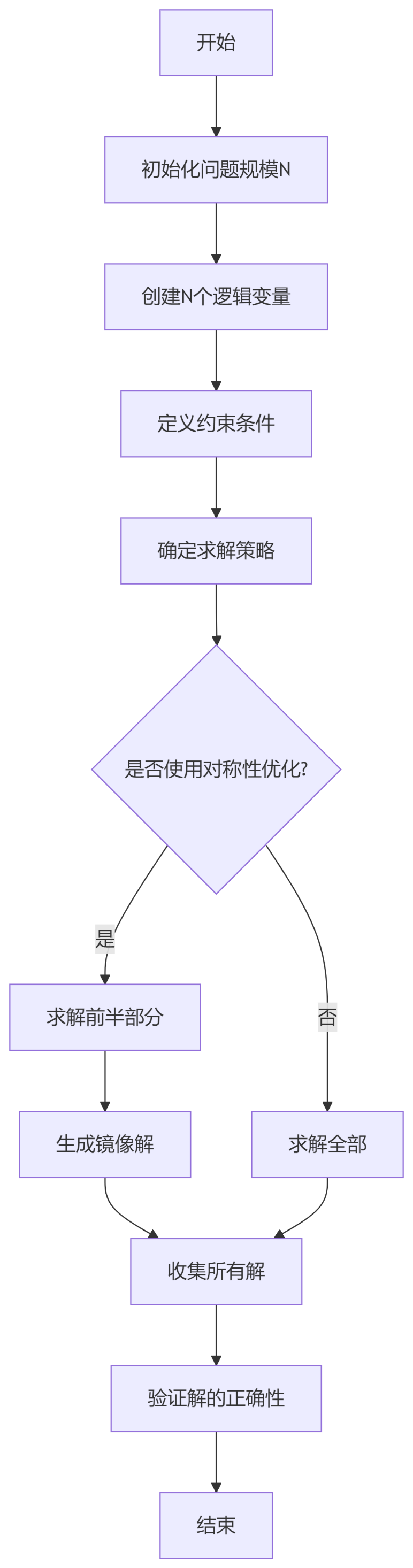
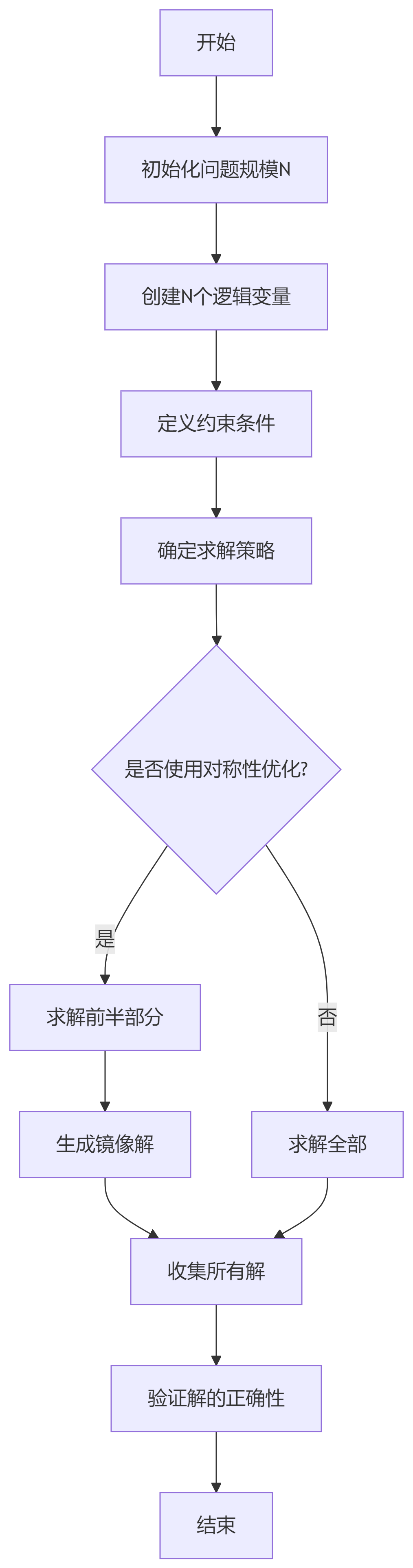
**在基础版本中发现直接对由于运算时间过慢，因此版本2中改进了约束表示方法，分离行相等检查和对角线检查**

**版本3为提高求解效率，引入了对称性优化，只求解一半的问题，利用水平镜像生成其他解**

**你和你和**

1. **算法流程图**

**如下（由两种，应该是采用了对称优化的，一个是没有的）**

**你和你和你和你和你和你和你和你和你和你和你和**

**和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你**

**和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你和你**

1. **设计思路**
2. **约束表示设计**

使用逻辑变量表示皇后位置

通过数据结构隐式保证某些约束

设计高效的约束检查机制

1. **约束求解优化**

实现增量约束检查

优化约束评估顺序

使用延迟评估机制

1. **对称性利用**

识别问题的对称特性

设计高效的解生成策略

实现重复解的检测和过滤

1. **性能优化考虑**

最小化约束检查次数

优化内存使用

提高求解效率

**你和你和**

1. **算法实现**

**1. 核心数据结构和变量定义**

在N皇后问题的实现中，我们使用了以下关键的数据结构和变量：

from kanren import run, eq, var, conde

from kanren.core import lall, success, fail

import time

# 每列的皇后位置用逻辑变量表示

queens = [var() for \_ in range(n)]  # n为棋盘大小

这种表示方法的优势在于：

* 列号通过列表索引隐式表示，避免额外存储
* 每个变量只需要存储行号，简化了状态表示
* 使用逻辑变量便于约束求解系统进行推理1

**2. no\_attack函数**

no\_attack函数是算法的核心，它实现了皇后之间的互不攻击约束。其涉及多个关键点：

def no\_attack(q1, q2, col1, col2):

    def goal(s):

**这个函数用于检查两个皇后之间是否存在攻击关系。参数含义：**

* **q1, q2: 两个皇后的行位置（逻辑变量）**
* **col1, col2: 两个皇后的列位置（固定值）**

**这个函数返回一个goal函数，它是一个闭包，它接收状态参数s，用于在约束求解过程中进行状态检查。它可以访问外部函数的参数，并可以处理动态变化的状态**

**这个函数符合kanren库的约束求解机制**

        # 获取变量的值，如果已经绑定的话

        q1\_val = q1 if not hasattr(q1, '\_id') else s.get(q1, q1)

        q2\_val = q2 if not hasattr(q2, '\_id') else s.get(q2, q2)

这段代码的关键点在于：

1. **获取q1\_val和q2\_val**：

* 逻辑变量在求解过程中可能处于未绑定状态
* 直接对逻辑变量进行算术运算会导致类型错误
* 需要先检查变量是否已绑定值

1. **hasattr(q1, '\_id')的作用**：

* '\_id'属性表示这是一个逻辑变量
* 如果存在此属性，说明变量尚未绑定具体值
* 需要从状态s中获取当前绑定值

1. **s.get(q1, q1)的设计**：
   * s是当前的状态对象，包含变量绑定信息
   * 如果变量已绑定，返回绑定的值
   * 如果未绑定，返回变量本身

**以上内容为遇到相应问题后查阅资料所得，现在仅仅只会简单使用，可能解释的不到位**

# 如果任一变量未绑定，则延迟评估

if hasattr(q1\_val, '\_id') or hasattr(q2\_val, '\_id'):

    return success(s)

**这个检查主要作用是避免对未绑定变量进行运算**

**它允许约束系统在变量绑定后再进行检查，能够提高求解效率，减少不必要的计算**

# 行相等检查

if q1\_val == q2\_val:

    return fail(s)

# 对角线检查

if abs(q1\_val - q2\_val) == abs(col1 - col2):

    return fail(s)

**这个约束检查分为两部分：**

**行相等检查：确保两个皇后不在同一行，直接比较行号是否相等，相等则返回fail表示约束不满足**

**对角线检查：检查是否在同一对角线上，使用行差和列差的绝对值比较，相等则表示在同一对角线上**

return success(s)

return goal

**当内部goal函数在约束满足时返回success(s)**

**外部no\_attack函数返回goal函数本身**

1. **Solve\_n\_queens函数**

**solve\_n\_queens函数是N皇后问题的核心实现，该函数经过多次优化迭代，下面详细分析其实现细节。**

def solve\_n\_queens(n):

    start\_time = time.time()

    queens = [var() for \_ in range(n)]

**使用逻辑变量数组queens表示每列皇后的位置，每个变量代表一列中皇后所在的行号**

**使用var()创建逻辑变量，便于约束求解系统处理**

constraints = []

for i in range(len(queens)):

    q = queens[i]

    # 添加域约束

    constraints.append(conde(\*[[eq(q, row)] for row in range(n)]))

**这段代码构建了域约束，确保每个皇后都在合法的行位置上（0到n-1行之间），并使用conde构建析取（OR）约束。**

**其中eq(q, row)确保变量q的值等于某个有效的行号。**

**接着，我们需要处理皇后之间的攻击关系：**

    # 攻击约束：确保皇后之间互不攻击

    for j in range(i):

        constraints.append(no\_attack(queens[j], q, j, i))

**这段函数为每对皇后添加互不攻击的约束，使用no\_attack函数检查行和对角线冲突**

**由简单数学关系可知只检查当前皇后与之前放置的皇后即可**

solutions = run(0, queens, \*constraints)

**使用run函数执行约束求解**

**其中，参数0表示获取所有可能的解，\*constraints展开所有约束条件**

**Solutions接受所有返回的满足约束的解**

end\_time = time.time()

return solutions, end\_time - start\_time

**返回解集和运行时间，便于性能分析和评估**

**这个函数是重点优化的函数，共有三版，这里讲解的是第二版，这个版本相对来说效率还可以（第三版仅仅只是额外采用了对称生成解的方法，逻辑约束方面优化不大，所以没有特别讲解**

1. **Main函数**

**Main函数有两个版本，一个是程序本身决定n为几，另一个是由外界输入并且可以一直循环，这里讲解后者**

def main():

    while True:

        try:

            n = int(input("请输入皇后数量 (输入0退出): "))

            if n == 0:

                break

            elif n < 4:

                print("请输入至少4的值以获得有意义的解")

                continue

            solutions, runtime = solve\_n\_queens(n)

            print(f"找到 {len(solutions)} 个解决方案，用时 {runtime:.4f} 秒")

            for solution in (solutions):

                        print(solution)

        except ValueError:

            print("请输入有效的整数")

**这个函数功能如下**

**输入**

**使用无限循环持续接收用户输入,当输入0时退出循环**

**对输入进行有效性验证：确保输入为整数（通过try-except捕获ValueError）,确保N≥4（N皇后问题在N<4时无解）**

**问题求解与展示**

**调用solve\_n\_queens函数处理用户输入，并获取解决方案和运行时间**

**然后展示解的数量以及求解所用时间（精确到小数点后4位）**

**遍历打印所有解**

**且这个函数包含了两层错误处理机制：**

* **数值验证：确保输入的N大于等于4**
* **类型验证：通过try-except捕获非法输入（不是数字）**

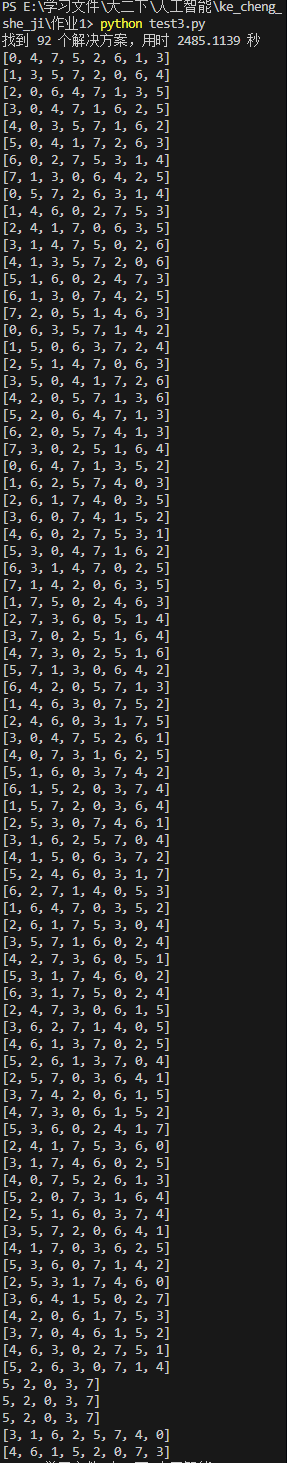
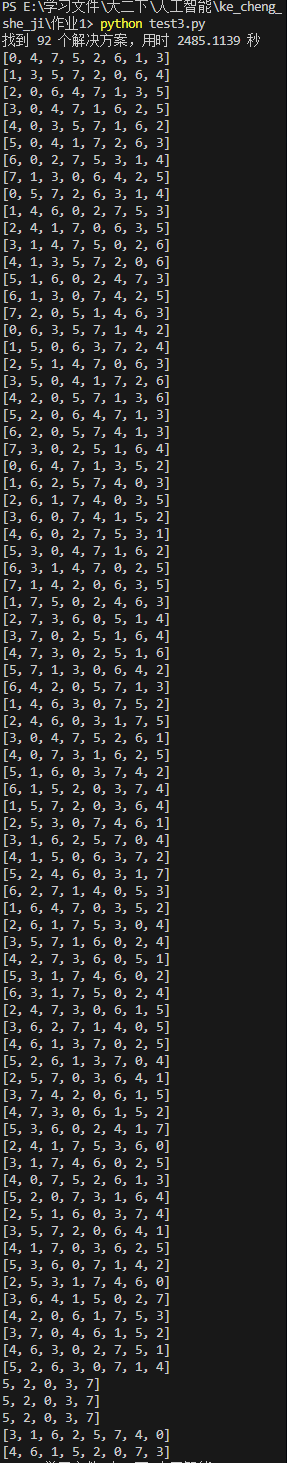
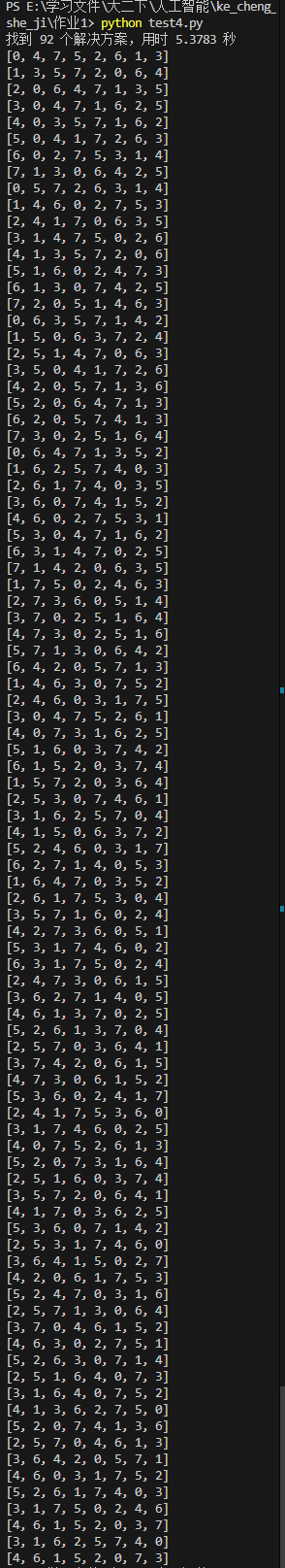
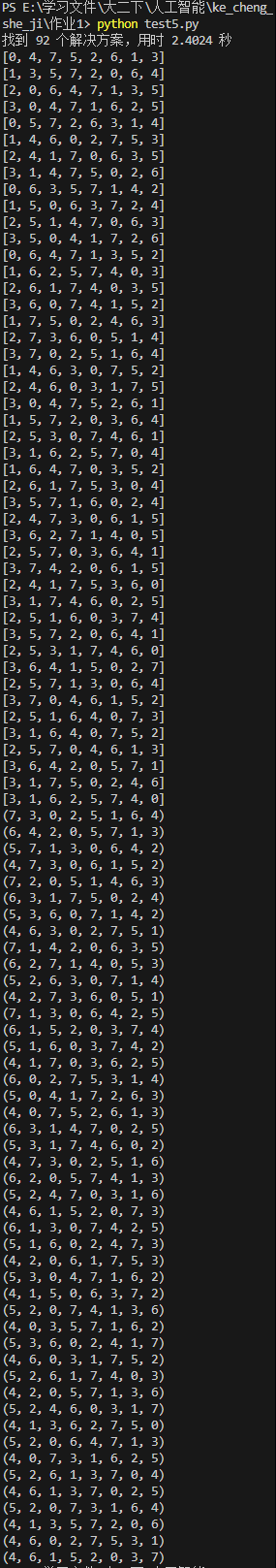
1. **实验结果**

**以下为三个版本代码的运行时间，结果如下**

[**具体结果见最后（点击跳转）**](#运行结果)

**你和你和你和你和**

**你和你和**

**从图中，我们能明显看到运行时间的缩短，2485秒到5.3秒再到2.4秒**

**效率提到了上千倍**

**你和你和**

**初始版本运行时间达到四十多分钟，明显需要优化，这里可以进行优化的点应该主要有两个，一个是no\_attack函数，另一个是solve\_n\_queens函数。**

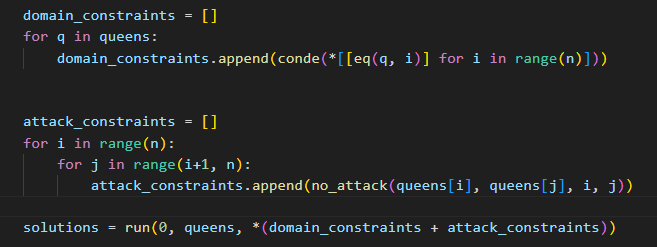
**在进行大量尝试后，no\_attack并未发现良好的优化方案。**

**你和你和**

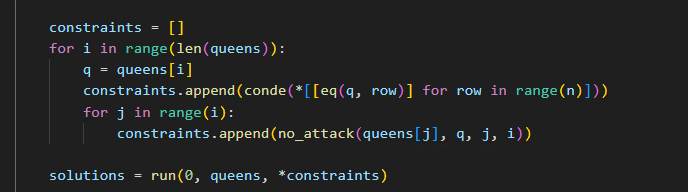
**而solve\_n\_queens函数可能存在的问题有**约束检查效率低，搜索空间过大，重复计算多等

所以这里对与约束条件进行了优化

**约束条件从**



**变为了**



**这里多次独立的约束检查会增加计算开销，这样优化后可以减少约束检查的次数，提高约束传播效率，更好地利用变量间的关联性**

**你和你和**

**在这一步优化后，运行效率提高了四百多倍**

**你和你和**

**再然后，在观察了输出的结果以及进行分析后发现最终输出结果理应是堆成的，那么我们可以尝试去利用这个性质去优化运行时间，只求解一半，然后另一半由对称生成**

**你和你和**

    # 通过对称性生成完整解集

    if n % 2 == 1 and half\_n > 1:  # 奇数 n 的特殊处理

        middle = half\_n - 1

        middle\_solutions = [sol for sol in partial\_solutions if sol[0] == middle]

        remaining = [sol for sol in partial\_solutions if sol[0] != middle]

    else:

        middle\_solutions = []

        remaining = partial\_solutions

    # 生成对称解

    full\_solutions = list(partial\_solutions)

    for sol in remaining:

        # 水平镜像

        mirror\_sol = tuple(n - 1 - pos for pos in sol)

        if mirror\_sol not in full\_solutions:

            full\_solutions.append(mirror\_sol)

**这样，我们成功较少了一半的运行时间**

**你和你和**

**你和你和你和**

1. **总结分析**

**在进行实验的过程中，我也遇到了不少问题**

**以下，我将结合调试过程进行一些讲解说明**

**变量运算问题**

**在开发初期，我们遇到了逻辑变量不能直接进行算术运算的问题。**

# 错误的实现

if abs(q1 - q2) == abs(col1 - col2):  # 直接对逻辑变量进行运算

    return fail(s)

**这里两个var逻辑变量进行加减，程序直接报错，在查询了相关资料后，发现了如下解决办法**

# 正确的实现

q1\_val = q1 if not hasattr(q1, '\_id') else s.get(q1, q1)

q2\_val = q2 if not hasattr(q2, '\_id') else s.get(q2, q2)

if abs(q1\_val - q2\_val) == abs(col1 - col2):

    return fail(s)

**这个可以说是整个过程中最麻烦的点了，然后根据资料显示，可能存在变量为绑定情况，这个时候就要进行延时分析了（虽然本题未出现）**

        # 如果任一变量未绑定，则延迟评估

        if hasattr(q1\_val, '\_id') or hasattr(q2\_val, '\_id'):

            return success(s)

**最后通过这次N皇后问题的实验，我深入理解了约束逻辑编程的特点和优化技巧。从最初的简单实现到最终的优化版本，整个过程让我收获颇丰。**

**在处理逻辑变量运算问题时1，我学会了如何正确处理约束编程中的变量特性。通过实现延迟评估机制2，不仅解决了变量运算的问题，还提高了程序的执行效率。最令人兴奋的是发现并利用问题的对称性特征，这让程序性能得到了质的飞跃。**

**优化过程中，我们采用了多项策略：合并约束检查3、使用更高效的约束求解方式，以及利用问题的对称性。这些优化措施使得程序运行时间从最初的41分钟降到了2秒多，效果显著。**

**这个项目不仅让我掌握了约束逻辑编程的技术要点，更重要的是培养了我分析问题、优化算法的能力。我认识到，编程不仅是实现功能，更重要的是要理解问题本质，找到最优解决方案。这些经验对我今后解决类似问题将会很有帮助。**

**附**

**[0, 4, 7, 5, 2, 6, 1, 3]**

**[1, 3, 5, 7, 2, 0, 6, 4]**

**[2, 0, 6, 4, 7, 1, 3, 5]**

**[3, 0, 4, 7, 1, 6, 2, 5]**

**[4, 0, 3, 5, 7, 1, 6, 2]**

**[5, 0, 4, 1, 7, 2, 6, 3]**

**[6, 0, 2, 7, 5, 3, 1, 4]**

**[7, 1, 3, 0, 6, 4, 2, 5]**

**[0, 5, 7, 2, 6, 3, 1, 4]**

**[1, 4, 6, 0, 2, 7, 5, 3]**

**[2, 4, 1, 7, 0, 6, 3, 5]**

**[3, 1, 4, 7, 5, 0, 2, 6]**

**[4, 1, 3, 5, 7, 2, 0, 6]**

**[5, 1, 6, 0, 2, 4, 7, 3]**

**[6, 1, 3, 0, 7, 4, 2, 5]**

**[7, 2, 0, 5, 1, 4, 6, 3]**

**[0, 6, 3, 5, 7, 1, 4, 2]**

**[1, 5, 0, 6, 3, 7, 2, 4]**

**[2, 5, 1, 4, 7, 0, 6, 3]**

**[3, 5, 0, 4, 1, 7, 2, 6]**

**[4, 2, 0, 5, 7, 1, 3, 6]**

**[5, 2, 0, 6, 4, 7, 1, 3]**

**[6, 2, 0, 5, 7, 4, 1, 3]**

**[7, 3, 0, 2, 5, 1, 6, 4]**

**[0, 6, 4, 7, 1, 3, 5, 2]**

**[1, 6, 2, 5, 7, 4, 0, 3]**

**[2, 6, 1, 7, 4, 0, 3, 5]**

**[3, 6, 0, 7, 4, 1, 5, 2]**

**[4, 6, 0, 2, 7, 5, 3, 1]**

**[5, 3, 0, 4, 7, 1, 6, 2]**

**[6, 3, 1, 4, 7, 0, 2, 5]**

**[7, 1, 4, 2, 0, 6, 3, 5]**

**[1, 7, 5, 0, 2, 4, 6, 3]**

**[2, 7, 3, 6, 0, 5, 1, 4]**

**[3, 7, 0, 2, 5, 1, 6, 4]**

**[4, 7, 3, 0, 2, 5, 1, 6]**

**[5, 7, 1, 3, 0, 6, 4, 2]**

**[6, 4, 2, 0, 5, 7, 1, 3]**

**[1, 4, 6, 3, 0, 7, 5, 2]**

**[2, 4, 6, 0, 3, 1, 7, 5]**

**[3, 0, 4, 7, 5, 2, 6, 1]**

**[4, 0, 7, 3, 1, 6, 2, 5]**

**[5, 1, 6, 0, 3, 7, 4, 2]**

**[6, 1, 5, 2, 0, 3, 7, 4]**

**[6, 1, 3, 0, 7, 4, 2, 5]**

**[7, 2, 0, 5, 1, 4, 6, 3]**

**[0, 6, 3, 5, 7, 1, 4, 2]**

**[1, 5, 0, 6, 3, 7, 2, 4]**

**[2, 5, 1, 4, 7, 0, 6, 3]**

**[3, 5, 0, 4, 1, 7, 2, 6]**

**[4, 2, 0, 5, 7, 1, 3, 6]**

**[5, 2, 0, 6, 4, 7, 1, 3]**

**[6, 2, 0, 5, 7, 4, 1, 3]**

**[7, 3, 0, 2, 5, 1, 6, 4]**

**[0, 6, 4, 7, 1, 3, 5, 2]**

**[1, 6, 2, 5, 7, 4, 0, 3]**

**[2, 6, 1, 7, 4, 0, 3, 5]**

**[3, 6, 0, 7, 4, 1, 5, 2]**

**[4, 6, 0, 2, 7, 5, 3, 1]**

**[5, 3, 0, 4, 7, 1, 6, 2]**

**[6, 3, 1, 4, 7, 0, 2, 5]**

**[7, 1, 4, 2, 0, 6, 3, 5]**

**[1, 7, 5, 0, 2, 4, 6, 3]**

**[2, 7, 3, 6, 0, 5, 1, 4]**

**[3, 7, 0, 2, 5, 1, 6, 4]**

**[4, 7, 3, 0, 2, 5, 1, 6]**

**[5, 7, 1, 3, 0, 6, 4, 2]**

**[6, 4, 2, 0, 5, 7, 1, 3]**

**[1, 4, 6, 3, 0, 7, 5, 2]**

**[2, 4, 6, 0, 3, 1, 7, 5]**

**[3, 0, 4, 7, 5, 2, 6, 1]**

**[4, 0, 7, 3, 1, 6, 2, 5]**

**[5, 1, 6, 0, 3, 7, 4, 2]**

**[6, 1, 5, 2, 0, 3, 7, 4]**

**[3, 6, 0, 7, 4, 1, 5, 2]**

**[4, 6, 0, 2, 7, 5, 3, 1]**

**[5, 3, 0, 4, 7, 1, 6, 2]**

**[6, 3, 1, 4, 7, 0, 2, 5]**

**[7, 1, 4, 2, 0, 6, 3, 5]**

**[1, 7, 5, 0, 2, 4, 6, 3]**

**[2, 7, 3, 6, 0, 5, 1, 4]**

**[3, 7, 0, 2, 5, 1, 6, 4]**

**[4, 7, 3, 0, 2, 5, 1, 6]**

**[5, 7, 1, 3, 0, 6, 4, 2]**

**[6, 4, 2, 0, 5, 7, 1, 3]**

**[1, 4, 6, 3, 0, 7, 5, 2]**

**[2, 4, 6, 0, 3, 1, 7, 5]**

**[3, 0, 4, 7, 5, 2, 6, 1]**

**[4, 0, 7, 3, 1, 6, 2, 5]**

**[5, 1, 6, 0, 3, 7, 4, 2]**

**[6, 1, 5, 2, 0, 3, 7, 4]**

**[1, 4, 6, 3, 0, 7, 5, 2]**

**[2, 4, 6, 0, 3, 1, 7, 5]**

**[3, 0, 4, 7, 5, 2, 6, 1]**

**[4, 0, 7, 3, 1, 6, 2, 5]**

**[5, 1, 6, 0, 3, 7, 4, 2]**

**[6, 1, 5, 2, 0, 3, 7, 4]**

**[1, 5, 7, 2, 0, 3, 6, 4]**

**[2, 5, 3, 0, 7, 4, 6, 1]**

**[3, 1, 6, 2, 5, 7, 0, 4]**

**[4, 1, 5, 0, 6, 3, 7, 2]**

**[5, 2, 4, 6, 0, 3, 1, 7]**

**[6, 2, 7, 1, 4, 0, 5, 3]**

**[1, 6, 4, 7, 0, 3, 5, 2]**

**[2, 6, 1, 7, 5, 3, 0, 4]**

**[3, 5, 7, 1, 6, 0, 2, 4]**

**[4, 2, 7, 3, 6, 0, 5, 1]**

**[5, 3, 1, 7, 4, 6, 0, 2]**

**[6, 3, 1, 7, 5, 0, 2, 4]**

**[2, 4, 7, 3, 0, 6, 1, 5]**

**[3, 6, 2, 7, 1, 4, 0, 5]**

**[4, 6, 1, 3, 7, 0, 2, 5]**

**[5, 2, 6, 1, 3, 7, 0, 4]**

**[1, 5, 7, 2, 0, 3, 6, 4]**

**[2, 5, 3, 0, 7, 4, 6, 1]**

**[3, 1, 6, 2, 5, 7, 0, 4]**

**[4, 1, 5, 0, 6, 3, 7, 2]**

**[5, 2, 4, 6, 0, 3, 1, 7]**

**[6, 2, 7, 1, 4, 0, 5, 3]**

**[1, 6, 4, 7, 0, 3, 5, 2]**

**[2, 6, 1, 7, 5, 3, 0, 4]**

**[3, 5, 7, 1, 6, 0, 2, 4]**

**[4, 2, 7, 3, 6, 0, 5, 1]**

**[5, 3, 1, 7, 4, 6, 0, 2]**

**[6, 3, 1, 7, 5, 0, 2, 4]**

**[2, 4, 7, 3, 0, 6, 1, 5]**

**[3, 6, 2, 7, 1, 4, 0, 5]**

**[4, 6, 1, 3, 7, 0, 2, 5]**

**[5, 2, 6, 1, 3, 7, 0, 4]**

**[6, 3, 1, 7, 5, 0, 2, 4]**

**[2, 4, 7, 3, 0, 6, 1, 5]**

**[3, 6, 2, 7, 1, 4, 0, 5]**

**[4, 6, 1, 3, 7, 0, 2, 5]**

**[5, 2, 6, 1, 3, 7, 0, 4]**

**[2, 5, 7, 0, 3, 6, 4, 1]**

**[3, 7, 4, 2, 0, 6, 1, 5]**

**[4, 7, 3, 0, 6, 1, 5, 2]**

**[5, 3, 6, 0, 2, 4, 1, 7]**

**[2, 4, 1, 7, 5, 3, 6, 0]**

**[3, 1, 7, 4, 6, 0, 2, 5]**

**[4, 0, 7, 5, 2, 6, 1, 3]**

**[5, 2, 0, 7, 3, 1, 6, 4]**

**[2, 5, 1, 6, 0, 3, 7, 4]**

**[3, 5, 7, 2, 0, 6, 4, 1]**

**[4, 1, 7, 0, 3, 6, 2, 5]**

**[5, 3, 6, 0, 7, 1, 4, 2]**

**[2, 5, 3, 1, 7, 4, 6, 0]**

**[3, 6, 4, 1, 5, 0, 2, 7]**

**[4, 2, 0, 6, 1, 7, 5, 3]**

**[5, 2, 4, 7, 0, 3, 1, 6]**

**[2, 5, 7, 1, 3, 0, 6, 4]**

**[3, 7, 0, 4, 6, 1, 5, 2]**

**[4, 6, 3, 0, 2, 7, 5, 1]**

**[5, 2, 6, 3, 0, 7, 1, 4]**

**[2, 5, 1, 6, 4, 0, 7, 3]**

**[3, 1, 6, 4, 0, 7, 5, 2]**

**[4, 1, 3, 6, 2, 7, 5, 0]**

**[5, 2, 0, 7, 4, 1, 3, 6]**

**[2, 5, 7, 0, 4, 6, 1, 3]**

**[3, 6, 4, 2, 0, 5, 7, 1]**

**[4, 6, 0, 3, 1, 7, 5, 2]**

**[5, 2, 6, 1, 7, 4, 0, 3]**

**[3, 1, 7, 5, 0, 2, 4, 6]**

**[4, 6, 1, 5, 2, 0, 3, 7]**

**[3, 1, 6, 2, 5, 7, 4, 0]**

**[4, 6, 1, 5, 2, 0, 7, 3]**

**你和你和**