项目说明文档

数据结构课程设计

——N 皇后问题

作 者 姓 名　林继申

学　　号　2250758

指 导 教 师　张　颖

学 院 专 业　软件学院　软件工程

同济大学
TONGJI UNIVERSITY

二〇二三 年 十二 月 十三 日

目录

1 项目分析

1.1 项目背景分析

八皇后问题是一个古老而著名的问题，是回溯算法的经典问题。该问题是 19 世纪著名数学家高斯在 1850 年提出的：在 8×8 的国际象棋棋盘上，放置 8 个皇后，要求没有一个皇后能够攻击任何其它一个皇后，即任意两个皇后不能处于同一行，同一列或者同一条对角线上，求解有多少种摆法。

本实验拓展了八皇后问题，即皇后个数可以由用户输入（即 N 皇后问题）。

N 皇后问题是典型的八皇后问题的一般化版本，其中 N 可以是任何正整数。解决 N 皇后问题的方法通常是使用回溯算法。回溯算法的基本思想是逐行放置皇后，确保每个皇后都不会攻击其他皇后，如果发现无法继续放置皇后而不违反规则，就回溯到前一步，重新尝试其他位置。这个过程一直持续，直到找到解法或者确定没有解法为止。

N 皇后问题的应用不仅限于国际象棋，它还在计算机科学和人工智能领域具有重要意义，因为它是一种经典的组合问题，可以用来测试和优化搜索算法的性能。此外，N 皇后问题还具有数学背景，它涉及到组合数学、图论和计算复杂性理论等领域。N 皇后问题的解法数量随 N 的增加呈指数增长，因此对于较大的 N，寻找所有解可能会变得非常困难。

1.2 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)用户输入参数：允许用户输入 N 皇后问题中的 N 的值，即皇后个数和棋盘大小；

(2)求解 N 皇后问题：使用回溯算法，寻找 N 皇后问题的所有解法。确保每个解法中的皇后都不会互相攻击，即不在同一行、同一列或同一对角线上；

(3)输出解法：将找到的每个解法以可视化的方式输出；

(4)统计解法数量：在找到所有解法后，需要输出 N 皇后问题的总解法数量；

(5)用户友好性：用户应该能够轻松理解和使用该程序。程序应提供合适的输入提示和错误处理，以确保用户输入的 N 值在合理范围内；

(6)效率和性能：尽量优化算法以提高程序的效率和性能，以便在 N 值较大的情况下也能够有效地求解问题；

(7)异常处理机制：实现异常处理机制，确保系统稳定性和安全性，避免因

用户输入错误导致系统崩溃或信息丢失。

1.3 项目功能分析

本项目旨在通过输入参数、求解 N 皇后问题、输出解法（界面设计与可视化）和异常处理等功能，以实现求解 N 皇后问题的核心逻辑。下面对项目的功能进行详细分析。

1.3.1 输入参数功能

允许用户通过标准输入输入一个正整数 N，表示 N 皇后问题中皇后个数与棋盘的大小。程序应确保 N 的正确性，即 N 必须是正整数且在合理范围内。

1.3.2 求解 N 皇后问题功能

使用回溯算法来查找 N 皇后问题的解法。这包括逐行放置皇后，确保每个皇后都不会攻击其他皇后，如果发现无法继续放置皇后而不违反规则，就回溯到前一步，重新尝试其他位置。这个过程一直持续，直到找到解法或确定没有解法为止。

1.3.3 界面设计与可视化功能

将找到的每个解法以可视化的方式（一个棋盘）输出，其中皇后的位置用特殊字符（“●”）表示，而空格表示该位置没有皇后。每个解法都应该有一个编号，以便用户知道是第几种解法。

1.3.4 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

2 项目设计

2.1 数据结构设计

基于项目分析，本项目使用动态分配的 `int` 类型一维数组存储数据，而不使用 `int` 类型二维数组存储数据，原因如下：

(1)节省内存空间：本来应该用 `int` 类型二维数组来表示棋盘，但是由于八皇后问题中皇后们处在不同的行，所以可以用一维数组来存储。`array[n]=i` 表示棋盘中第 `n` 行第 `i` 列存在皇后（从 0 开始索引）。一维数组存储数据占用的内

存空间更小。在 N 皇后问题中，使用二维数组表示整个棋盘，需要 N×N 的内存空间，而使用一维数组，只需要 N 个元素的内存空间。这在处理较大 N 值时尤其有利，因为它减少了内存的使用；

⑵简化索引计算：一维数组的索引表示棋盘的行号，而值表示皇后所在的列号。这使得在放置和检查皇后位置时的索引计算更加简单和直观。对于二维数组，需要同时考虑行和列的索引，导致代码稍显复杂；

⑶方便回溯操作：在回溯算法中，使用动态分配的 int 类型一维数组存储数据，更加方便进行回溯操作，而不使用 int 类型二维数组存储数据，因为代码会稍显繁琐；

⑷提高程序性能：使用动态分配的 int 类型一维数组存储数据，可以降低算法的时间复杂度和空间复杂度，提高程序性能，这在处理较大 N 值时尤其有利。

2.2 N_Queens 类的设计

2.2.1 概述

N_Queens 类是一个用于解决 N 皇后问题的类。N 皇后问题要在 N×N 的棋盘上放置 N 个皇后，以确保它们互不攻击，即任意两个皇后不在同一行、同一列或同一对角线上。这个类实现了回溯算法来找到 N 皇后问题的所有解法，并提供了输出解法的功能。

2.2.2 类定义

```
class N_Queens {
private:
    int size;
    int count;
    int* array;
public:
    N_Queens(int n);
    ~N_Queens();
    bool isSafeToPlace(int n);
    void findRecursively(int n);
    void printChessboard(void);
    void solve(void);
};
```

2.2.3 私有数据成员

`int size`：皇后个数与棋盘大小

`int count`：解法数量

`int* array`：一个指向整数数组的指针，用于存储当前解法的皇后位置

2.2.4 构造函数

`N_Queens(int n);`

构造函数，用于初始化 `N_Queens` 对象。它接受一个整数参数 `n`，表示 N 皇后问题中皇后个数与棋盘的大小。在构造函数中，它分配内存来存储皇后位置，并将皇后位置初始化为-1，表示初始状态下没有皇后。

2.2.5 析构函数

`~N_Queens();`

析构函数，用于释放在构造函数中分配的内存，以避免内存泄漏。

2.2.6 公有成员函数

`bool isSafeToPlace(int n);`

检查在第 `n` 行放置皇后是否安全，即不会攻击其他皇后。它遍历数组中的已放置皇后位置，检查是否有冲突，返回 `true` 表示安全（可以放置皇后），`false` 表示不安全（不可以放置皇后）。

`void findRecursively(int n);`

递归地查找 N 皇后问题的解法。当 `n` 等于棋盘大小时，表示找到了一个解法，将其输出。否则，尝试在第 `n` 行放置皇后，并递归到下一行，直到找到所有解法。

`void printChessboard(void);`

以可视化方式输出一个解法，将棋盘状态打印到标准输出，显示皇后的位置和棋盘结构。

`void solve(void);`

解决 N 皇后问题，同时统计并输出所有解法的数量。

2.3 项目主体架构设计

项目主体架构设计为：

(1)程序开始：在程序开始时，输出项目信息和 N 皇后问题描述；

(2)输入 N 值并初始化 `n_queens` 对象：用户输入 N 值，用户提供 N 值后，程序使用这个值来初始化 `N_Queens` 类的对象 `n_queens`，用于解决问题；

(3)解决 N 皇后问题：使用回溯法解决 N 皇后问题；

(4)依次输出所有解法和输出解法总数；

(5)等待用户按下回车键以退出程序：在解决问题之后，程序将等待用户按下回车键，以便用户有足够的时间查看解法。

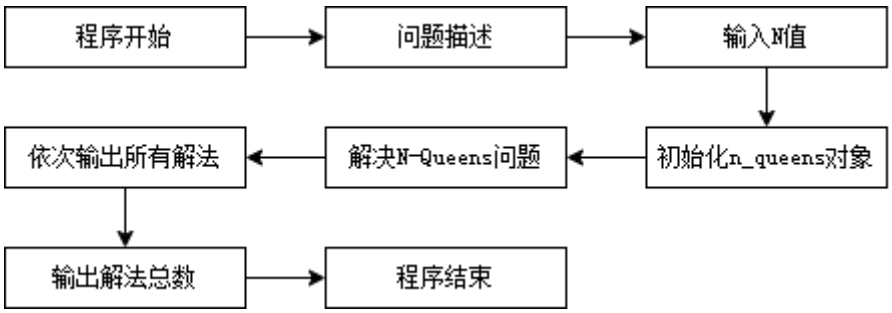

图 2.3.1 项目主体架构设计流程图

3 项目功能实现

3.1 项目主体架构的实现

3.1.1 项目主体架构实现思路

项目主体架构实现思路为：

(1)main 函数入口：主体架构从 main 函数开始执行。main 函数主要负责初始化、用户交互和程序结束。在 main 函数中，首先输出项目信息和 N 皇后问题的描述；

(2)获取用户输入 N 值：通过调用 inputInteger 函数来获取用户输入的 N 值。inputInteger 函数会要求用户输入一个整数，检查输入的有效性和范围，然后返回用户提供的 N 值。这个值将用于初始化 n_queens 对象；

(3)初始化 n_queens 对象：使用获取的 N 值，创建 N_Queens 对象 n_queens 并传递 N 值作为构造函数参数。在 N_Queens 的构造函数中，初始化棋盘大小、解法计数和皇后位置数组，为后续解决 N 皇后问题做准备；

(4)解决 N 皇后问题：调用 n_queens 对象的 solve 函数来解决 N 皇后问题。solve 函数内部会调用 findRecursively 函数，通过回溯算法递归地查找所有解法。每当找到一个解法时，会调用 printChessboard 函数将其可视化输出；

(5)等待用户按下回车键：在解决问题之后，程序会输出解法数量，然后进入等待状态。程序会等待用户按下回车键以继续，这增加了用户友好性，让用户有足够的时间查看解法；

(6)程序结束：最后 main 函数返回 0，表示成功结束，程序执行完毕。

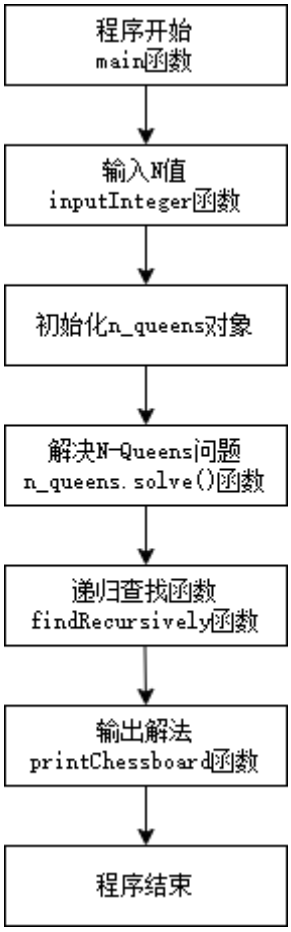

图 3.1.1.1 项目主体架构实现流程图

3.1.2 项目主体架构核心代码

```
int main()
{
    /* System entry prompt */
    std::cout << "+--------------------+" << std::endl;
    std::cout << "|     N 皇后问题      |" << std::endl;
    std::cout << "|  N Queens Problem  |" << std::endl;
    std::cout << "+--------------------+" << std::endl << std::endl;

    /* Problem description */
    std::cout << ">>> 问题描述" << std::endl << std::endl;
    std::cout << "     在一个 N × N 的棋盘上，放置 N 个皇后，使其互不攻"
<< std::endl;
    std::cout << "击，即没有任意两个皇后在同一行、同一列或同一对角线上。"
<< std::endl << std::endl;

    /* Input N and initialize n_queens object */
    N_Queens n_queens(inputInteger(1, 99, "皇后个数 N"));
```

```
    /* Solve N-Queens Problem */
    n_queens.solve();

    /* Wait for enter to quit */
    std::cout << "Press Enter to Quit" << std::endl;
    while (_getch() != '\r')
        continue;

    /* Program ends */
    return 0;
}
```

3.1.3 项目主体架构示例

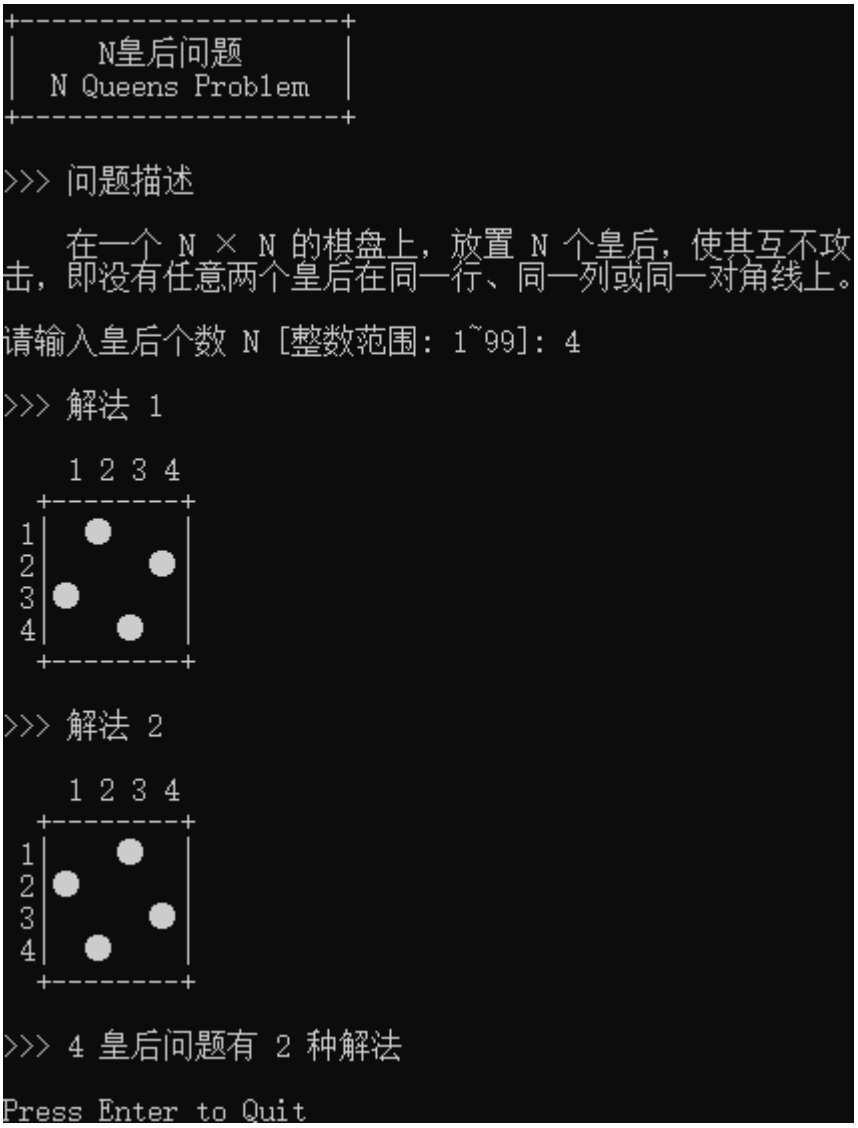

图 3.1.3.1 项目主体架构示例

3.2 输入参数功能的实现

3.2.1 输入参数功能实现思路

程序通过调用 inputInteger 函数输入皇后个数和棋盘的大小。inputInteger 函数用于获取用户输入的整数，同时限制输入必须在指定的范围内。inputOddInteger 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

(1)进入一个无限循环，它会一直运行直到用户提供有效的输入；

(2)用户的输入被读取到 tempInput 变量中，这里采用 double 类型来接收输入以便后续检查；

(3)进行输入验证：std::cin.good()检查输入流的状态是否正常，确保没有发生数据类型输入错误，tempInput==static_cast<int>(tempInput)检查用户输入是否为整数，通过将其转换为整数再比较，tempInput>=lowerLimit 和 tempInput<=upperLimit 确保输入在指定的范围内；

(4)合法输入处理：如果用户提供了合法的输入，函数会清除输入流的错误状态，丢弃输入缓冲区中的任何剩余内容，然后返回转换后的整数值；

(5)非法输入处理：如果用户提供的输入不合法，函数会输出错误消息，清除输入流的错误状态，丢弃输入缓冲区中的内容，并继续循环以等待用户提供合法的输入。

表 3.2.1.1 N 皇后问题不同 N 值的解法总数

皇后个数 N	解法总数
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596

本程序限制皇后个数范围在 1 至 99 之间。因为解决 N 皇后问题使用回溯算法。这种算法尝试不同的皇后布局，如果发现某个布局违反了规则，就会回溯到前一步，然后尝试其他可能的布局。回溯算法的复杂性是指数级的，因为它需要探索大量的可能性。

3.2.2 输入参数功能核心代码

```
    int inputInteger(int lowerLimit, int upperLimit, const char* prompt)
    {
        while (true) {
            std::cout << "请输入" << prompt << " [整数范围: " << lowerLimit
<< "~" << upperLimit << "]: ";
            double tempInput;
            std::cin >> tempInput;
            if (std::cin.good() && tempInput == static_cast<int>(tempInput)
&& tempInput >= lowerLimit && tempInput <= upperLimit) {
                std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                return static_cast<int>(tempInput);
            }
            else {
                std::cerr << std::endl << ">>> " << prompt << " 输入不合法，
请重新输入" << prompt << " ！" << std::endl << std::endl;
                std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            }
        }
    }
```

3.2.3 输入参数功能示例

输入参数功能示例见图 3.1.3.1。

3.3 求解 N 皇后问题功能的实现

3.3.1 求解 N 皇后问题功能实现思路

本程序主要通过 N_Queens 类中的 isSafeToPlace、findRecursively、solve 三个函数实现求解 N 皇后问题功能，下面对这三个函数的逻辑进行详细分析：

⑴isSafeToPlace(int n)：这个函数用于检查在第 n 列放置皇后是否是

安全的。具体来说，它会遍历之前已经放置的皇后（0 到 n-1 列），并检查是否有任何冲突。它检查两种类型的冲突：

①同一列冲突：它检查是否有其他皇后已经放置在同一列（array[n]==array[i]）；

②对角线冲突：它检查是否有其他皇后位于与当前皇后相对角度相同的位置，即在同一对角线上（std::abs(n-i)==std::abs(array[n]-array[i])）。

如果在任何情况下存在冲突，函数返回 false，表示不能在第 n 列放置皇后，否则返回 true，表示可以安全地放置皇后；

(2)findRecursively(int n)：这是递归的核心函数，用于递归地寻找 N 皇后问题的解。它的参数 n 表示当前要尝试放置皇后的列。如果 n 等于棋盘的大小（size），意味着所有的皇后都已经成功放置，这时会打印出一个解，然后增加计数器 count。如果 n 小于 size，表示还有皇后要放置，就会在第 n 列的每个可能的位置尝试放置皇后，然后检查 isSafeToPlace 函数来确保放置是安全的。如果安全，就继续递归调用 findRecursively(n+1)来放置下一个皇后；

(3)solve(void)：这个函数是 N 皇后问题的解决入口点。它开始调用 findRecursively(0)，从第 0 列开始递归地尝试放置皇后，然后找到所有可能的解。一旦所有可能的解都被找到，它会输出解的数量，即 count。

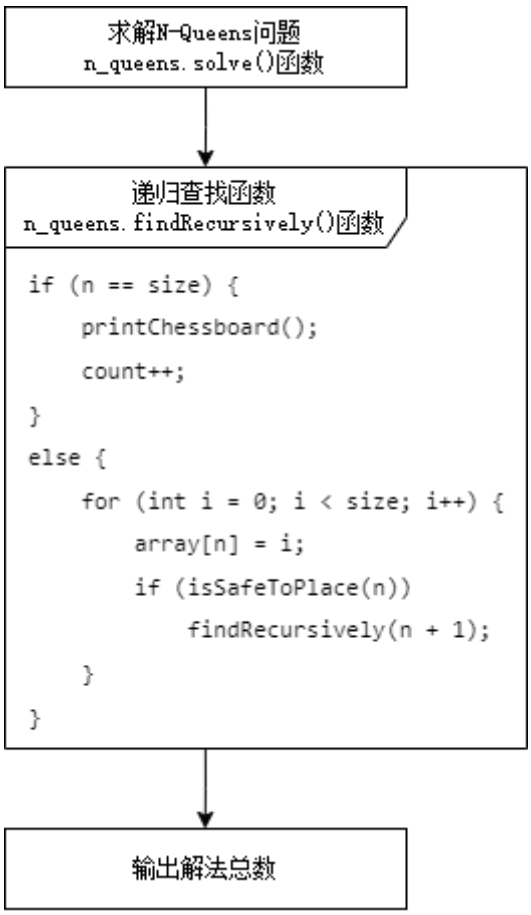

图 3.3.1.1 求解 N 皇后问题功能实现流程图

这些函数结合在一起，通过递归回溯的方式，遍历所有可能的解空间，找到N皇后问题的所有解。每当找到一个解时，它将打印出棋盘状态，并继续搜索下一个解。整个过程会重复，直到找到所有可能的解。这种递归回溯方法是解决组合问题的一种常见方法，它在满足特定条件的情况下不断尝试各种可能性，直到找到所有解或者穷尽了所有可能性。

3.3.2 求解N皇后问题功能核心代码

```
bool N_Queens::isSafeToPlace(int n)
{
    for (int i = 0; i < n; i++)
        if (array[n] == array[i] || std::abs(n - i) == std::abs(array[n]
- array[i]))
            return false;
    return true;
}

void N_Queens::findRecursively(int n)
{
    if (n == size) {
        printChessboard();
        count++;
    }
    else {
        for (int i = 0; i < size; i++) {
            array[n] = i;
            if (isSafeToPlace(n))
                findRecursively(n + 1);
        }
    }
}

void N_Queens::solve(void)
{
    findRecursively(0);
    std::cout << std::endl << ">>> " << size << " 皇后问题有 " << count
<< " 种解法" << std::endl << std::endl;
}
```

3.3.3 求解N皇后问题功能示例

N皇后问题是一个具有指数级复杂性的组合问题，随着N值的增加，寻找解的难度急剧上升。解决这个问题需要使用高效的算法，以降低搜索空间的复杂性和提高求解效率。由于篇幅有限，求解N皇后问题功能示例只列出1皇后问题、

4 皇后问题、5 皇后问题和 6 皇后问题的示例。

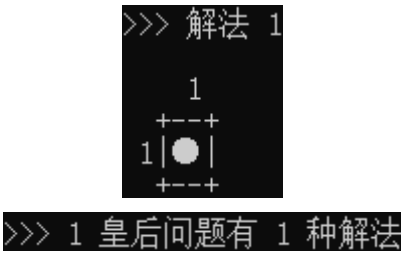

图 3.3.3.1 求解 1 皇后问题功能示例

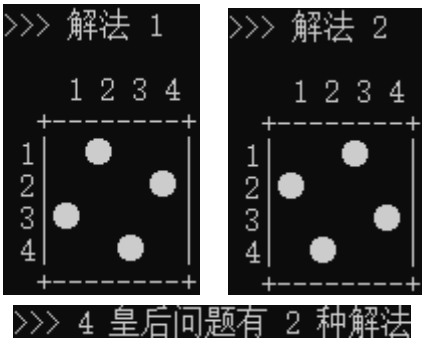

图 3.3.3.2 求解 4 皇后问题功能示例

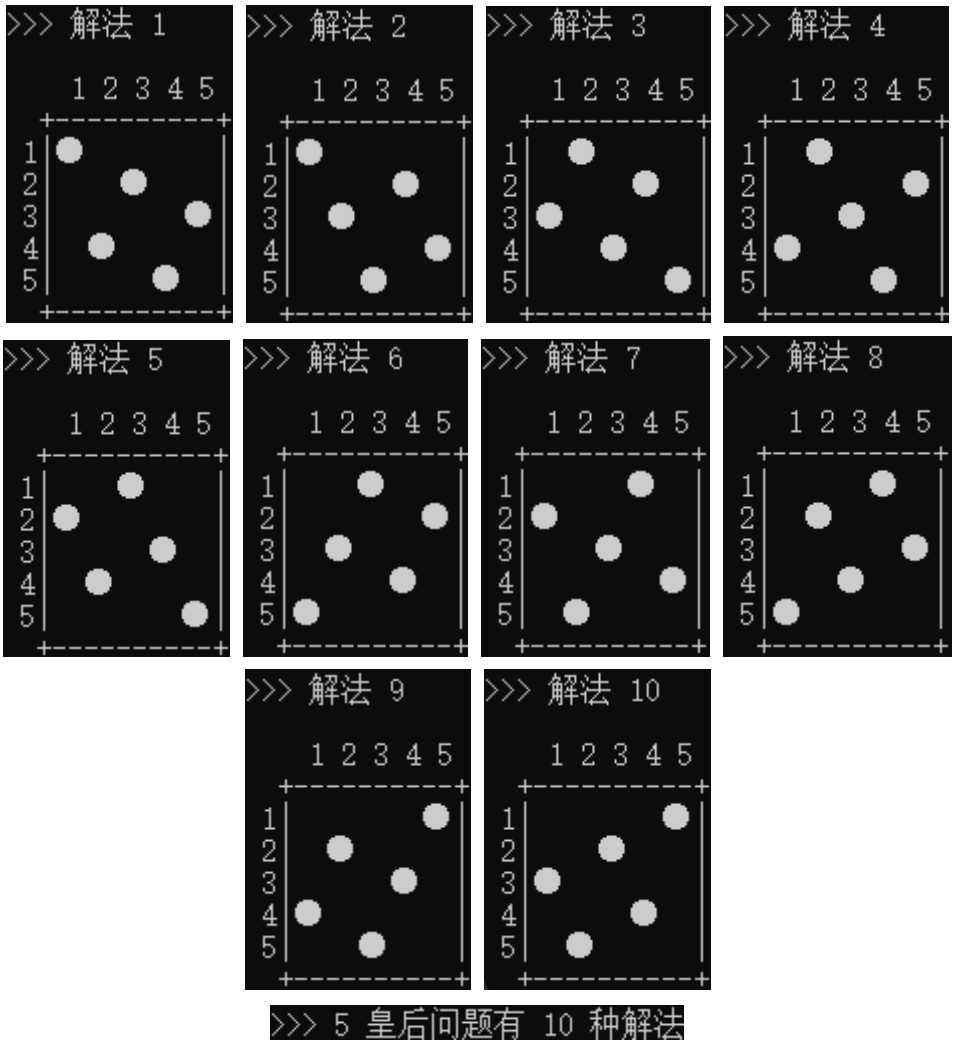

图 3.3.3.3 求解 5 皇后问题功能示例

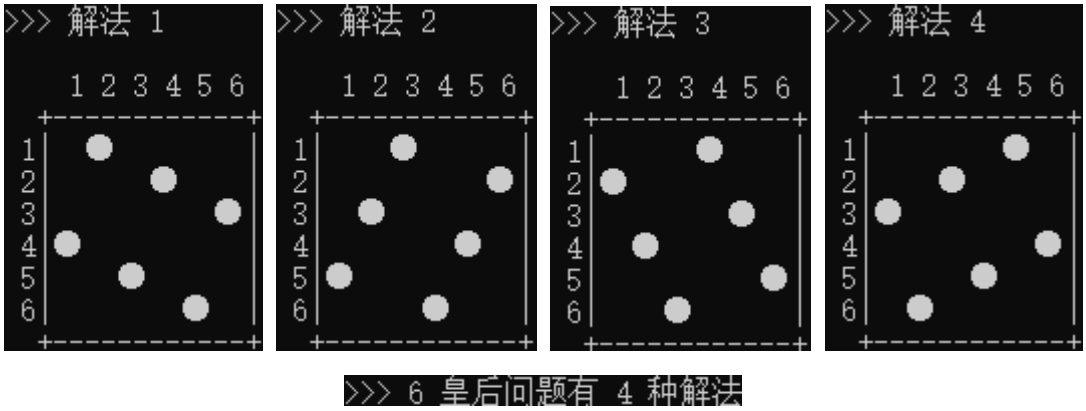

图 3.3.3.4 求解 6 皇后问题功能示例

3.4 界面设计与可视化功能的实现

3.4.1 界面设计与可视化功能实现思路

执行界面设计与可视化功能的函数为 N_Queens 类的成员函数 printChessboard，其实现思路为：

(1)输出解法编号：首先，在控制台上输出当前解的编号，这是通过 count 计算得到的，因为计数器 count 表示已经找到的解的数量；

(2)绘制列标：在下一行输出列标，从 1 到 N，表示棋盘的列号；

(3)绘制顶部边框：在下一行输出一个顶部边框，这个边框会包围整个棋盘；

(4)逐行绘制棋盘：接下来，使用两层循环，外层循环遍历每一行（0 到 N-1），内层循环遍历每一列（0 到 N-1）。在每一行内，外层循环会输出行号，然后进入内层循环；

(5)内层循环：内层循环用于遍历当前行的每一列，检查是否应该在该位置绘制皇后（“●”），或者应该绘制空白位置（“　”）。这是通过比较当前列（j）是否等于数组 array 中对应行的值（array[i]）来实现的。如果相等，表示在该位置应该绘制皇后，否则绘制空白；

(6)绘制行尾边框：在每一行绘制完成后，在同一行的行号后输出一个竖线（“|”），表示行尾边框；

(7)重复：重复上述步骤，绘制完所有行的棋盘状态；

(8)绘制底部边框：在最后一行绘制底部边框，与顶部边框类似。

3.4.2 界面设计与可视化功能核心代码

```
void N_Queens::printChessboard(void)
{
    std::cout << std::endl << ">>> 解法 " << count + 1 << std::endl << std::endl << "   ";
```

```
        for (int i = 0; i < size; i++)
            std::cout << std::setw(2) << i + 1;
        std::cout  <<  std::endl  <<  "    +"  <<  std::setfill('-')  <<
std::setw(2 * size) << "-" << std::setfill(' ') << "+" << std::endl;
        for (int i = 0; i < size; i++) {
            std::cout << std::setw(2) << i + 1 << "|";
            for (int j = 0; j < size; j++)
                std::cout << (j == array[i] ? "●" : "  ");
            std::cout << "|" << std::endl;
        }
        std::cout << "  +" << std::setfill('-') << std::setw(2 * size) <<
"-" << std::setfill(' ') << "+" << std::endl;
    }
```

3.4.3 界面设计与可视化功能示例

界面设计与可视化功能示例见图 3.1.3.1、图 3.3.3.1、图 3.3.3.2、图 3.3.3.3、图 3.3.3.4。

3.5 异常处理功能的实现

在进行 N_Queens 类中的 int 类型一维数组等的动态内存申请时，程序使用 new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL 或 nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1) 向标准错误流 std::cerr 输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2) 调用 exit 函数，返回错误码 MEMORY_ALLOCATION_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

下面是动态内存申请的异常处理的一个代码示例：

```
N_Queens::N_Queens(int n)
{
    size = n;
    count = 0;
    array = new(std::nothrow) int[size];
    if (array == NULL) {
        std::cerr << "Error: Memory allocation failed." << std::endl;
        exit(MEMORY_ALLOCATION_ERROR);
    }
    for (int i = 0; i < size; i++)
        array[i] = -1;
}
```

4 项目测试

4.1 输入参数功能测试

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。

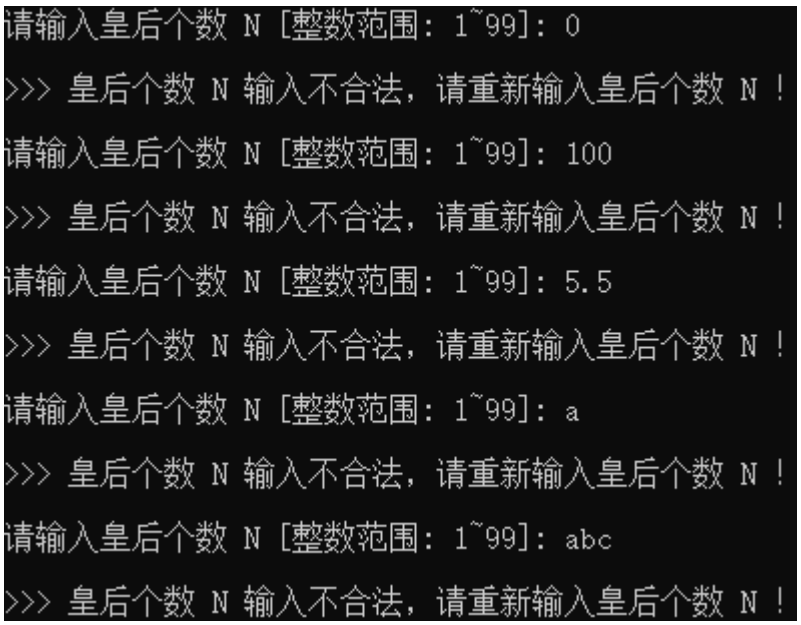

图 4.1.1 输入参数功能测试

当输入合法时，程序继续运行。

4.2 求解 N 皇后问题功能测试

由于篇幅有限和运算复杂度因素，求解 N 皇后问题功能测试只列出 14 种情况。

图 4.2.1 求解 N 皇后问题功能测试

测试结果与表 3.2.1.1 一致。

4.3 界面设计与可视化功能测试

界面设计与可视化功能测试见图 3.1.3.1、图 3.3.3.1、图 3.3.3.2、图 3.3.3.3、图 3.3.3.4。

4.4 退出程序功能测试

为避免直接运行可执行文件在输入完成后会发生闪退的情况，本程序使用如下代码，创建了一个无限循环，等待用户按下回车键，以便退出程序。避免结果输出结束后程序迅速退出的情况。

```
/* Wait for enter to quit */
std::cout << "Press Enter to Quit" << std::endl;
while (_getch() != '\r')
    continue;
```

5 集成开发环境与编译运行环境

Windows 系统：Windows 11 x64

Windows 集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

Windows 编译运行环境：本项目适用于 x86 架构和 x64 架构

Linux 系统：CentOS 7 x64

Linux 编译命令：

g++ '/root/桌面/Share_Folder/n_queens_problem.cpp' -o '/root/桌面/Share_Folder/n_queens_problem' -lncurses

Linux 运行命令：

'/root/桌面/Share_Folder/n_queens_problem'

图 5.1 Linux 环境程序运行示例

本项目使用条件编译解决 Windows 系统和 Linux 系统编译环境的差异，示例代码如下。

```
#ifdef _WIN32
#include <conio.h>
#elif __linux__
#include <ncurses.h>
#endif
```