

项目说明文档

数据结构课程设计

——N皇后问题

作 者 姓 名： 陆诚彬

学 号： 2254321

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

[1 项目背景 3](#_Toc7287)

[2 项目需求分析 3](#_Toc2091)

[2.1 功能需求 3](#_Toc32346)

[2.2 非功能需求 3](#_Toc6168)

[2.3 项目输入输出需求 3](#_Toc7532)

[2.3.1 输入格式 3](#_Toc32627)

[2.3.2 输出格式 4](#_Toc1603)

[2.3.3 项目示例 4](#_Toc8793)

[3 项目设计 4](#_Toc22267)

[3.1 数据结构设计 4](#_Toc3864)

[3.1.1 地图表示 4](#_Toc21160)

[3.2 类设计 4](#_Toc14961)

[3.2.1 类内主要函数 5](#_Toc18715)

[4 项目实现 5](#_Toc29372)

[4.1 初始化棋盘实现 5](#_Toc15856)

[4.1.1 初始化棋盘功能简介 5](#_Toc26941)

[4.1.2 初始化棋盘核心代码 5](#_Toc20746)

[4.2检查放置安全性实现 6](#_Toc27448)

[4.2.1 检查放置安全性功能简介 6](#_Toc15327)

[4.2.2 检查放置安全性核心代码 6](#_Toc28800)

[4.3 放置皇后实现 6](#_Toc4683)

[4.3.1 放置皇后功能简介 6](#_Toc5009)

[4.3.2 放置皇后核心代码 7](#_Toc10939)

[4.4 系统总体功能流程图 8](#_Toc26491)

[5 设计小结 9](#_Toc28912)

[5.1 问题背景和推广 9](#_Toc552)

[5.2 功能与非功能需求 9](#_Toc16302)

[5.3 数据结构与算法设计 9](#_Toc1470)

[5.4 代码实现 9](#_Toc277)

[6 软件测试 10](#_Toc2608)

[6.1 输入测试 10](#_Toc4716)

[6.1.1 正常输入 10](#_Toc4157)

[6.1.2 输入超界/非法 10](#_Toc4013)

[6.2 输出测试 10](#_Toc12645)

# 1 项目背景

N皇后问题源自经典的八皇后问题，它是回溯算法的经典应用案例。这个问题最早由十九世纪的数学家高斯在1850年提出。问题的核心是在一个8x8的国际象棋棋盘上放置8个皇后，要求这些皇后互不攻击，即任意两个皇后不能位于同一行、同一列或同一对角线上。高斯最初估计有76种可能的解决方案。1854年，一些作者在柏林的象棋杂志上发表了40种不同的解，后来，使用图论方法证实共有92种解决方案。

随着技术的发展，这个问题被推广到了更广泛的N皇后问题，其中“N”表示棋盘和皇后的数量，这个数量由用户输入确定。这个问题不仅仅是一个数学或计算机科学问题，它也是算法设计和计算能力的一个测试。

# 2 项目需求分析

## 2.1 功能需求

1. **动态输入：**用户可以输入任意的N值，程序应能处理不同大小的棋盘。
2. **有效性检测：**程序应能有效地检查皇后放置的位置，确保任意两个皇后不在同一行、同一列或同一对角线上。
3. **回溯算法应用：**使用回溯算法解决问题。该算法会尝试在每一行放置一个皇后，并对每一列进行检查，确保无冲突。如果某一行的所有列都无法放置皇后，则算法回溯到上一行，改变皇后的位置。
4. **结果输出：**输出所有可能的解决方案，包括每个方案的具体皇后位置。

## 2.2 非功能需求

1. 算法优化：考虑算法的效率，尤其是在较大的N值时。
2. 用户界面：简洁明了的用户界面，方便用户输入N值和查看结果。
3. 可扩展性：代码应具有良好的结构和注释，以便未来的维护和扩展。
4. 错误处理：合理的错误处理机制，如对非法输入的处理。

## 2.3 项目输入输出需求

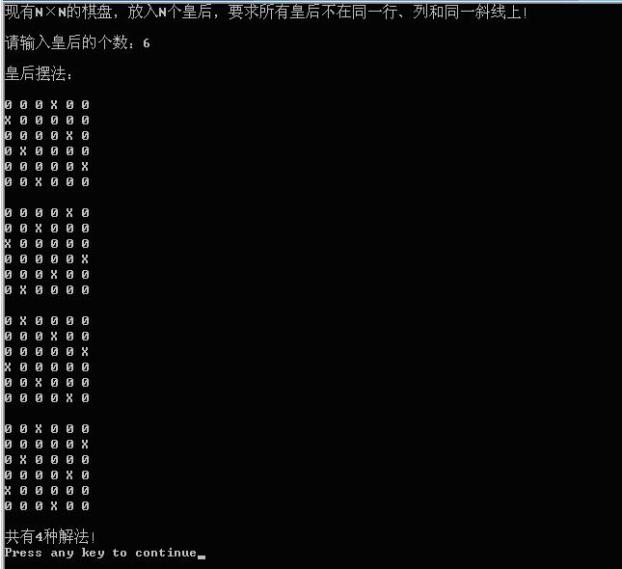
### 2.3.1 输入格式

输入皇后个数N（3 < N < 21）。

### 2.3.2 输出格式

输出所有皇后的布局。

### 2.3.3 项目示例



# 3 项目设计

## 3.1 数据结构设计

### 3.1.1 地图表示

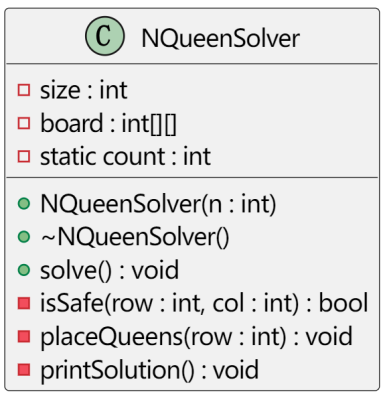
**目的：**有效地存储和访问棋盘的每个单元格。

**结构：**二维数组 int \*\*board。

**访问：**通过 board[x][y] 来访问特定位置。

## 3.2 类设计

NQueenSolver 类包含了解决N皇后问题所需的所有功能和数据，UML图如下：



## 3.2.1 类内主要函数

**私有方法：**

1. **bool** isSafe(**int** row, **int** col)；
2. 检查在row行col列放置皇后是否安全（不与其他皇后冲突）。
4. **void** placeQueens(**int** row)；
5. 递归函数，用于在棋盘上放置皇后。尝试在每一行的每一列放置皇后，并检查是否安全。如果发现冲突，则回溯并尝试不同的列。
7. **void** printSolution()；
8. 打印找到的一种解决方案，并增加解决方案的计数。

**公共方法：**

1. **void** solve()；
2. 启动皇后放置过程，是解决问题的主要入口。

# 4 项目实现

## 4.1 初始化棋盘实现

### 4.1.1 初始化棋盘功能简介

**功能描述：**

初始化一个 N x N 的棋盘，其中 N 是用户输入的皇后数量。棋盘上的每个单元格最初都设置为 0，表示没有皇后。

**实现方式：**

1. 在 NQueenSolver 类的构造函数中，根据用户输入的 N 值创建一个二维数组 board。
2. 使用嵌套循环，将 board 的每个元素初始化为 0。

### 4.1.2 初始化棋盘核心代码

1. NQueenSolver(**int** n) : size(n)
2. {
3. board = **new** **int** \*[size];
4. **for** (**int** i = 0; i < size; ++i)
5. {
6. board[i] = **new** **int**[size]{0}; // Initialize each row with 0s.
7. }
8. }

## 4.2检查放置安全性实现

### 4.2.1 检查放置安全性功能简介

**功能描述：**

检查在棋盘的特定位置放置皇后是否安全。安全意味着在同一行、同一列和两个对角线上没有其他皇后。

**实现方式：**

1. 实现一个私有方法 isSafe，它接受行和列作为参数。
2. 方法遍历棋盘上当前行之前的所有行，检查三个方向：当前列、左对角线和右对角线。
3. 如果任何方向上有皇后，返回 false；否则返回 true。

### 4.2.2 检查放置安全性核心代码

1. **bool** isSafe(**int** row, **int** col)
2. {
3. **for** (**int** i = 0; i < row; ++i)
4. {
5. // Check column, and two diagonals.
6. **if** (board[i][col] == 1)
7. **return** **false**;
8. **if** (col - (row - i) >= 0 && board[i][col - (row - i)] == 1)
9. **return** **false**;
10. **if** (col + (row - i) < size && board[i][col + (row - i)] == 1)
11. **return** **false**;
12. }
13. **return** **true**;
14. }

## 4.3 放置皇后实现

### 4.3.1 放置皇后功能简介

**功能描述：**

递归地在棋盘上放置皇后。从第一行开始，为每一行选择一个安全的列位置放置皇后。

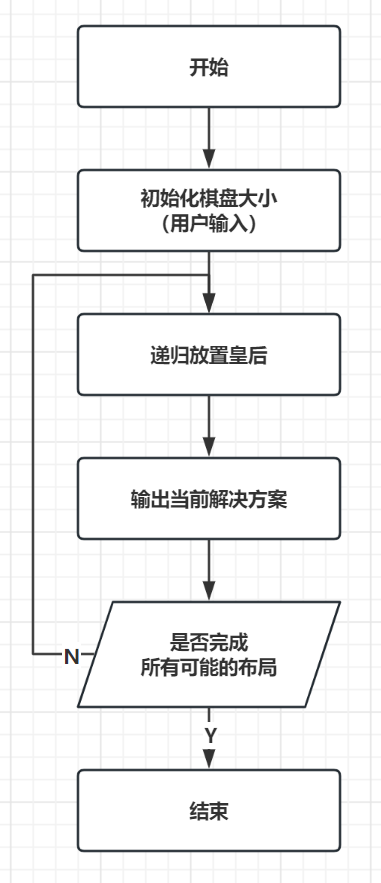
**实现方式：**

1. 使用私有方法 placeQueens，它接受当前行作为参数。
2. 对于棋盘的每一列，使用 isSafe 检查是否可以放置皇后。如果可以，放置皇后并递归地调用 placeQueens 放置下一行的皇后。
3. 如果在当前行找不到安全的位置，则回溯到上一行并尝试其他列位置。

### 4.3.2 放置皇后核心代码

1. **void** placeQueens(**int** row)
2. {
3. **if** (row == size)
4. {
5. printSolution();
6. **return**;
7. }
9. **for** (**int** col = 0; col < size; ++col)
10. {
11. **if** (isSafe(row, col))
12. {
13. board[row][col] = 1;  // Place queen.
14. placeQueens(row + 1); // Recur to place the rest of the queens.
15. board[row][col] = 0;  // Backtrack: remove queen and try next column.
16. }
17. }
18. }

## 4.4 系统总体功能流程图



# 5 设计小结

本项目深入探索了N皇后问题，一项具有挑战性的算法问题，旨在展示高效算法设计和编程实践。通过深入理解问题的背景、需求分析、数据结构设计、类设计、以及具体的实现细节，本项目展现了从理论到实践的完整开发过程。

## 5.1 问题背景和推广

项目从传统的八皇后问题扩展至N皇后问题，显示了从特定情况到一般情况的推广。这种推广不仅增加了问题的复杂度，还提高了解决问题所需的算法智能和计算资源。

## 5.2 功能与非功能需求

功能需求强调了程序必须能够接受动态输入、有效检测皇后的放置，运用回溯算法，并输出所有可能的解决方案。而非功能需求则关注了算法效率、用户界面的友好性、代码的可扩展性和错误处理机制，体现了软件工程的综合要求。

## 5.3 数据结构与算法设计

选择适合的数据结构（如二维数组）来高效存储棋盘信息是关键。算法设计中，特别是回溯算法的应用，展示了如何通过递归方式探索解决方案空间，并在必要时进行回溯。

## 5.4 代码实现

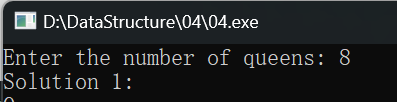
代码实现部分详细介绍了初始化棋盘、检查放置安全性、以及放置皇后的核心代码。这些实现细节揭示了理论和实践之间的紧密联系，展现了算法设计在实际编程中的应用。

整体上，本项目不仅是对经典N皇后问题的深入研究，也是对高效算法设计和编程实践的一次全面展示。通过精心设计的数据结构、算法和代码实现，项目成功地解决了一个经典的计算机科学问题，并为类似问题的解决提供了参考框架。此外，考虑到非功能需求，如用户界面和代码可维护性，也为软件工程领域提供了重要的学习案例。

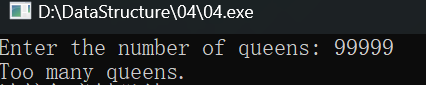
# 6 软件测试

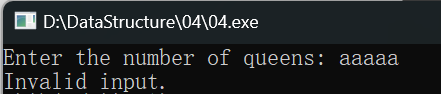
## 6.1 输入测试

### 6.1.1 正常输入



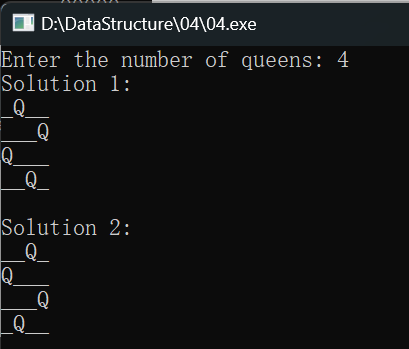
### 6.1.2 输入超界/非法





**结论：**符合输入逻辑判断

## 6.2 输出测试



**结论：**符合输出逻辑，且是所有解决方案。