数据结构课程设计

项目说明文档

N皇后问题

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

1. **题目简介**

**1.实验背景**

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。该问题是十九世纪著名的数学家高斯在1850年提出的：在8\*8的国际象棋棋盘上，安放8个皇后，要求没有一个皇后能够“吃掉”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。

高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法得到结论，有92中摆法。

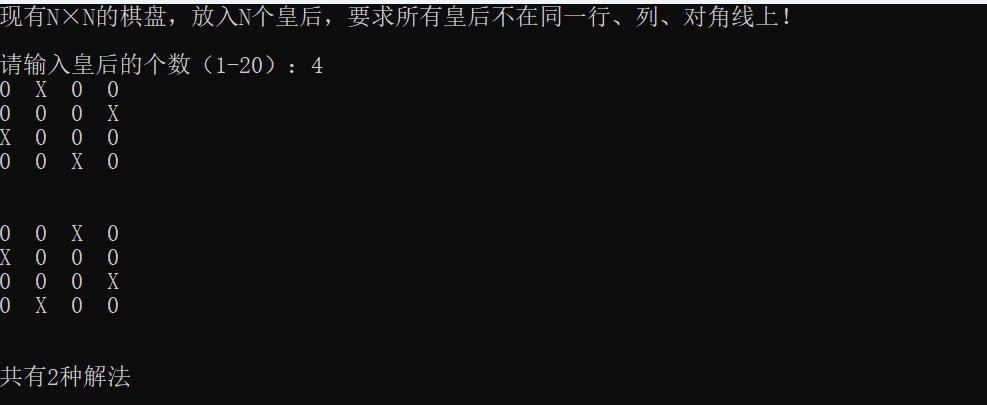
本实验拓展了N皇后问题，即皇后个数由用户输入。

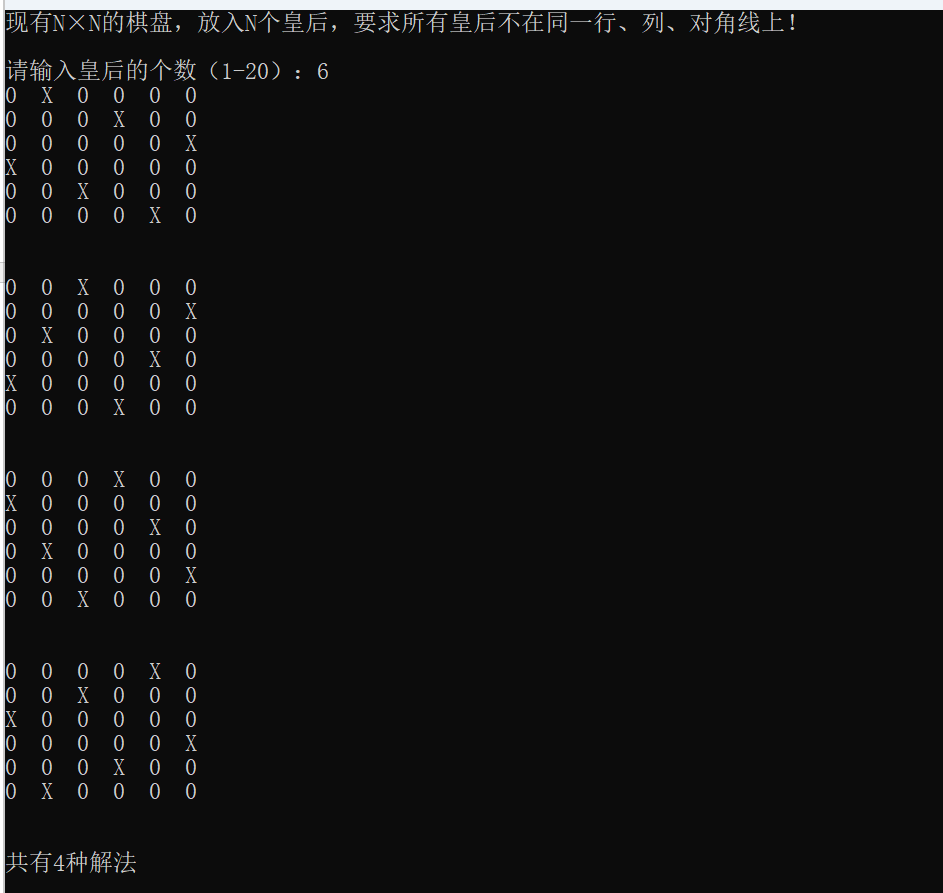
**2.实验功能**

八皇后在棋盘上分布的各种可能的格局数目非常大，约等于2的32次方种，但是，可以将一些明显不满足问题要求的格局排除掉。由于任意两个皇后不能同行，即每行只能放置一个皇后，因此将第i个皇后放在第i航上，这样在放置第i个皇后时，只要考虑它与前i-1个皇后处于不同列和不同对角线位置上即可。

解决这个问题采用回溯法，首先将第一个皇后放置在第一行第一列，然后，依次在下一行上放置一个皇后，直到八个皇后全部放置安全。在放置每个皇后时，都依次兑每一列进行检测，首先检测放在第一列是否与已放置的皇后冲突，如不冲突，则将皇后放置在该列，否则，选择改行的下一列进行检测。如整行的八列都冲突，则回到上一行，重新选择位置，依次类推。

**3.项目示例**





1. **项目设计**

## 1.数据结构设计

**棋盘表示**：使用二维字符数组 char board[MAX][MAX] 来表示棋盘。每个元素代表棋盘上的一个单元格，可以是空（'0'）或皇后（'X'）。

**对角线冲突检测**：利用两个布尔数组 left\_diagonal[MAX \* 2] 和 right\_diagonal[MAX \* 2] 来分别记录棋盘上的左对角线和右对角线上是否已有皇后。对角线上的每个位置的状态由这两个数组的相应位置表示。

**列冲突检测**：使用布尔数组 bool is\_queen[MAX + 1] 来记录每一列是否已放置皇后。

## 2.类结构设计

**queen 类：用于实现N皇后问题的求解。**

私有成员变量：

char board[MAX][MAX]：存储棋盘的当前状态。

bool left\_diagonal[MAX \* 2] 和 bool right\_diagonal[MAX \* 2]：记录对角线上皇后的放置情况。

bool is\_queen[MAX + 1]：记录每一列皇后的放置情况。

公有成员变量：

int count：记录找到的解决方案的数量。

公有成员函数：

queen()：构造函数，初始化棋盘和相关数组。

void print\_board()：打印当前棋盘布局。

void find\_queen(int r)：递归搜索放置皇后的所有可能方案。

## 3.成员与操作设计

queen() 构造函数：

初始化解决方案计数器 count 为 0。

初始化 left\_diagonal 和 right\_diagonal 为 false，表示对角线上没有皇后。

初始化 is\_queen 为 false，表示各列上没有皇后。

初始化棋盘 board 所有单元格为 '0'。

print\_board() ：

遍历并打印棋盘布局，将皇后的位置显示为 'X'，空位显示为 '0'。

find\_queen(int r) ：

递归方法，用于在第 r 行放置皇后。

检查每一列以及相关对角线是否可以放置皇后，如果可以，则放置皇后并递归调用下一行的 find\_queen 。

如果到达棋盘的底部，则增加解决方案计数并返回。

在递归返回后，将皇后从当前位置移除，并重置相关的对角线和列的状态，以便尝试下一个可能的位置。

## 4.算法设计

本程序采用了回溯算法来解决N皇后问题，一种在递归过程中系统地搜索所有可能配置的方法。算法从棋盘的第一行开始，逐行尝试在每一列放置皇后。对于每个位置，算法首先检查当前列以及两条对角线上是否已经放置了皇后（冲突检测），如果没有冲突，则在该位置放置皇后并递归地继续处理下一行。如果找到一个完整的解决方案，算法会记录下这种布局。每当算法返回到上一行（回溯）时，它会移除上一次放置的皇后，并尝试在当前行的下一个位置放置皇后。这个过程会一直重复，直到搜索了棋盘上所有可能的排列组合。通过这种方式，算法能够有效地找出所有满足N皇后问题约束条件的解决方案。

1. **项目实现**

**3.1. N皇后问题初始化**

1. queen::queen() {
2. count = 0;
3. **for** (**int** i = 0; i < MAX \* 2; i++) {
4. left\_diagonal[i] = right\_diagonal[i] = **false**;
5. }
6. **for** (**int** i = 0; i < MAX; i++) {
7. is\_queen[i] = **false**;
8. **for** (**int** j = 0; j < MAX; j++) {
9. board[i][j] = '0';
10. }
11. }
12. }

queen::queen() 构造函数的目的是初始化N皇后问题的棋盘和相关的辅助数组。这个初始化过程为找到所有可能的皇后摆放方案设定了起始条件。

#### 实现步骤

**初始化解法计数：**

将解法计数 count 设置为0。这个计数器用于追踪找到的所有有效皇后摆放方案的数量。

**初始化对角线标记数组：**

通过遍历 left\_diagonal 和 right\_diagonal 数组，并将它们的所有元素设置为 false，标记所有对角线上初始时都没有皇后。

**初始化列标记数组：**

通过遍历 is\_queen 数组，并将其所有元素设置为 false，标记所有列上初始时都没有皇后。

**初始化棋盘：**

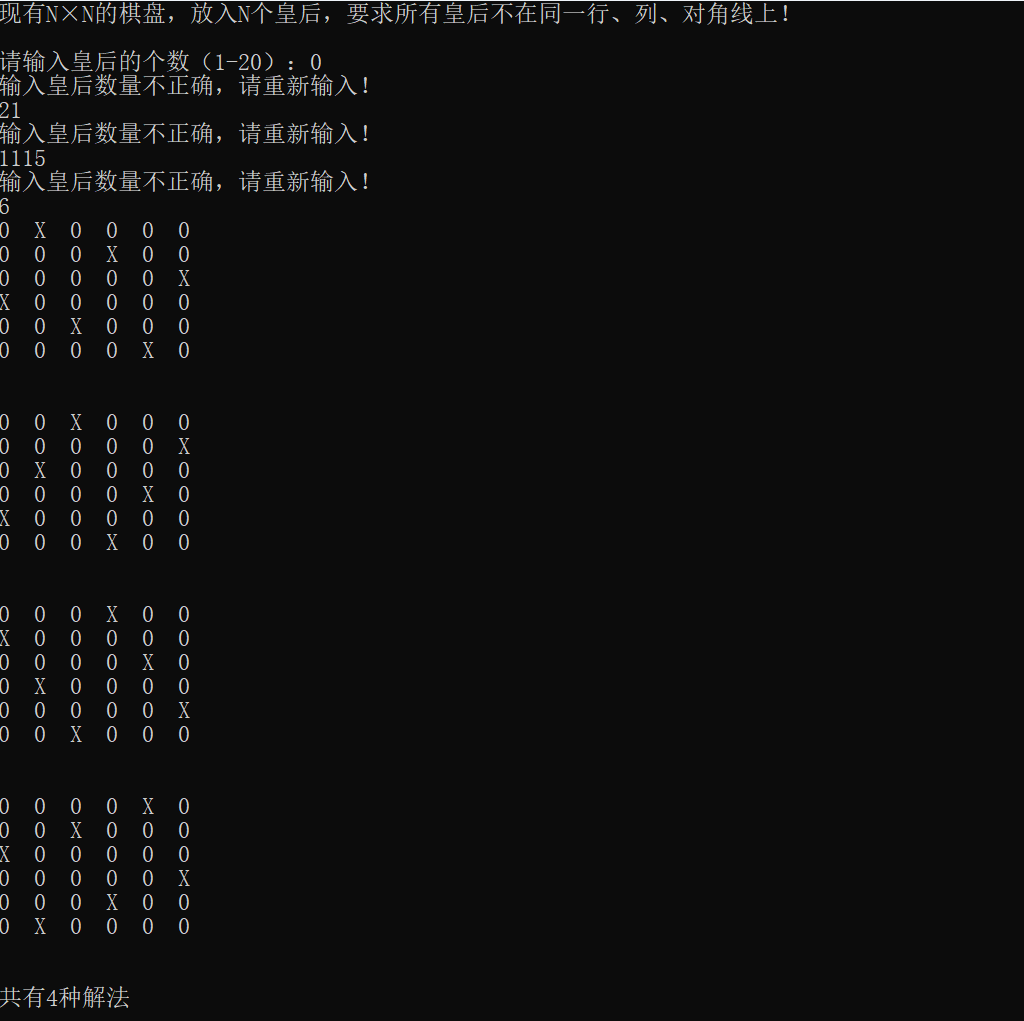
通过双重循环遍历 board 数组，并将每个单元格设置为 '0'。这表示棋盘的每个位置初始时都是空的，没有放置皇后。

**3.2. 找到所有皇后算法实现**

1. **void** queen::find\_queen(**int** r) {
2. **if** (r == n) {
3. print\_board();
4. count++;
5. **return**;
6. }
7. **for** (**int** i = 0; i < n; i++) {
8. **if** (!is\_queen[i] && !left\_diagonal[i + r] && !right\_diagonal[r - i + n]) {
9. board[r][i] = 'X';
10. is\_queen[i] = left\_diagonal[i + r] = right\_diagonal[r - i + n] = **true**;
11. find\_queen(r + 1);
12. is\_queen[i] = left\_diagonal[i + r] = right\_diagonal[r - i + n] = **false**;
13. board[r][i] = '0';
14. }
15. }
16. }

该方法通过递归和回溯技术系统地搜索棋盘上所有可能的皇后排列。对于每一行，它尝试在不违反N皇后规则的前提下放置一个皇后，并递归地在下一行重复这一过程。当找到一个有效的解决方案时，它会记录并打印出来。通过回溯，该方法能够探索棋盘上所有可能的组合，从而找到所有满足条件的解决方案。

**健全性测试：**



**皇后数量输入：**

在主函数中，程序要求用户输入皇后的数量。它使用一个循环来确保用户输入的数量在合法范围内（1到20）。

如果输入的数字不在这个范围内，或者输入的不是数字（!cin.good()），程序会清除输入流的错误状态（cin.clear()），忽略错误输入（cin.ignore(1024, '\n')），并提示用户重新输入。

**内存分配错误处理**

棋盘和辅助数组初始化：

在 queen 类的构造函数中，棋盘和辅助数组的初始化不涉及动态内存分配，因此不会出现内存分配错误。

递归深度安全性

递归函数 find\_queen：

递归深度由皇后的数量控制，因为用户输入的皇后数量限制在1到20，递归深度是安全的，不会导致栈溢出。

**逻辑错误处理**

皇后放置规则检查：

在放置皇后时，find\_queen 方法仔细检查每一列和对角线上是否已经放置了皇后。这确保了所有找到的解决方案都符合N皇后问题的规则。

**解决方案计数**

解决方案的记录：

每当找到一个有效的解决方案时，解决方案的计数器 count 会增加。这个计数器正确地反映了程序找到的所有可能解决方案的数量。

1. **项目创新点**

本实验在解决经典的N皇后问题中展现了几个创新点。首先，它采用了一种高效的递归回溯算法，不仅能够快速找到所有可能的解决方案，而且在算法执行过程中，对每个可能的解进行了动态的可视化展示，增强了程序的交互性和用户体验。其次，实验中对于冲突检测采用的方法，如利用数组标记对角线和列上的皇后位置，提升了算法的效率和简洁性。此外，程序对用户输入进行了严格的验证，确保了程序的健壮性和稳定性，同时提高了用户的使用便利性。这些创新不仅使得程序在技术层面上更为先进，也为用户提供了更加友好和直观的操作体验。**五、实验小结**

通过这次实验，深入了解了N皇后问题的复杂性及其解决方案的多样性。实验过程中，不仅锻炼了编程技巧，特别是在递归算法和数组操作方面的应用，而且加深了对算法设计和问题解决策略的理解。通过实现递归回溯算法，更好地掌握了算法的思想和编码实践，尤其是在递归深度控制和冲突检测机制方面。同时，这次实验也加强了对程序健壮性和用户交互设计的认识，明白了在实际编程中如何优雅地处理用户输入和程序输出。总的来说，这次实验不仅提升了编程技能，而且增强了解决实际问题的能力，对未来的编程实践和学习都有着重要的意义。