项目说明文档

数据结构课程设计

——N 皇后问题

作 者 姓 名：杨鑫

学　　号：1950787

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 分析

1.1 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。

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。

现在有人提出：要是是 6*6,9*9.....呢？于是希望有这样一个程序，可以求出 N*N 棋盘的 N 皇后问题。

1.2 功能分析

程序可以从用户获取一个输入，作为棋盘的大小（即确定一个 N * N 大小的棋盘），然后通过调用内部函数，进行 N 皇后问题的求解，求解完成后将所有解法打印出来。

2 设计

2.1 数据结构设计

由于解决 N 皇后问题的核心是在寻找解的过程中不断向下搜索和回溯，需要大量的插入删除操作，所以可以设计一个栈来储存临时的这些信息，从而可以非常便捷的进行一系列操作。

2.2 类结构设计

本程序设计了节点结构体（Node）和栈类（Stack），可以用于实现最核心的回溯求解功能。此外，还设计了一个 N 皇后类（NQueen）,它整合了棋盘大小，栈等基本信息，还实现了问题求解和输出解的功能。

2.3 成员与操作设计

节点结构体（Node）

```
1.  int x; // x 坐标
2.  int y; // y 坐标
3.  Node* link; // 后驱指针
```

```
4.   Node(int newX = 0, int newY = 0, Node* newLink = NULL) : x(newX), y(newY), link(newLink) {} // 参数有默认值的构造函数
5.   ~Node() {}
6.   Node(const Node& n) :x(n.x), y(n.y), link(n.link) {} // 拷贝构造函数
```

栈类（Stack）

私有成员：

```
1.   Node* top; // 栈顶
```

公有操作：

```
1.   Stack() :top(NULL) {} // 默认构造函数
2.   ~Stack(); // 析构函数
3.   void Push(int x, int y); // 入栈
4.   bool Pop(int &x, int &y); // 出栈
```

八皇后类（NQueen）

私有成员：

```
1.   int size; // 棋盘大小（皇后个数）
2.   int numOfSolutions; // 解的个数
3.   char** board; // 棋盘
4.   Stack* stack; // 临时存储可能的解法节点的栈
```

公有操作：

```
1.   NQueen(int sz = 0); // 构造函数
2.   ~NQueen() { delete board, delete stack; } // 析构函数
3.   void Solution(); // 求解
4.   bool Illigal(int rows, int cols); // 判断位置是否合法
5.   void Print(); // 打印解法
```

2.4 系统设计

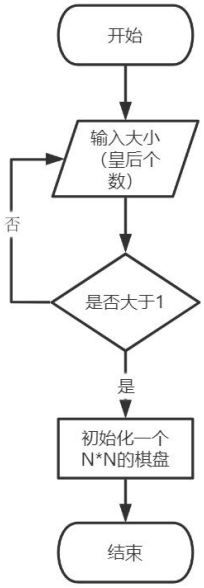
开始游戏后，由用户输入一个数字作为棋盘的大小，然后程序初始化一个该大小的棋盘。然后开始进行问题的求解。程序中主要使用了非递归的回溯解法，同时使用栈来存储每次临时坐标的信息，在回溯过程中不断弹出和压入坐标，最终求出一个解后打印棋盘信息，然后回溯寻求下一个解，直至所有解都被求出来。从而完成该问题的求解。

程序兼容了 windows 和 LINUX 平台，在双平台下均可以正常运行。

3 实现

3.1 初始化功能的实现

3.1.1 初始化功能的流程图



3.1.2 初始化功能的核心代码

```
int temSize;

// 检测输入的皇后个数是否是大于等于 2 的，若不是则重新输入
cin >> temSize;
while (temSize < 2) {
    cout << "皇后个数应该大于 1 ！请重新输入 ：  " << endl;
    cin >> temSize;
}

// 建立棋盘
MakeBoard(temSize);
```

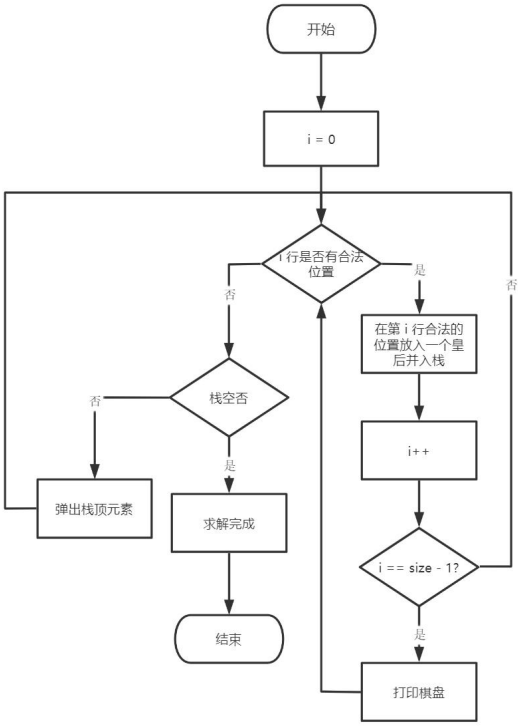
3.1.3 初始化功能的截屏示例

```
现有 N * N 的棋盘， 放入 N 个皇后， 要求所有皇后不在同一行、 列和同一斜线上!

请输入皇后的个数 ： 8
皇后摆法:
```

3.2 求解功能的实现

3.2.1 求解功能的流程图



3.2.2 求解功能的核心代码

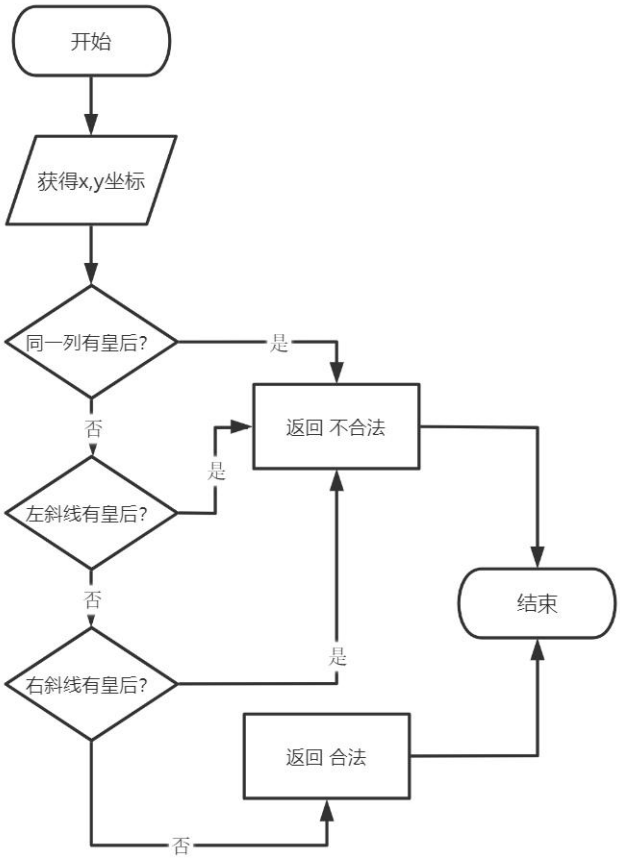
```
1.  int lastX = -1, lastY = -1; // 标记上次出栈的坐标位置，若无，则用 (-1, -1) 来表示
2.
3.  // 不断寻找解
4.  int rows = 0;
5.  while (true) {
6.      bool tag = false; // 标记，当前行是否有合法的位置并放置皇后于该位置上
7.      for (int cols = lastY + 1; cols < size; cols++) { // 寻找当前行是否有合法的位置
8.          if (Illigal(rows, cols)) {
9.              stack->Push(rows, cols);
10.             board[rows][cols] = 'X';
11.             tag = true; // 更新标记
12.             break;
13.         }
14.     }
15.     if (!tag) { // 未找到位置
```

```
            rows--;
            if (!(stack->Pop(lastX, lastY))) { // 栈空，说明已经完成寻找
                    cout << "共有" << numOfSolutions << "种解法！" << endl;
                return;
            }
            board[lastX][lastY] = '0'; // 棋盘该位置还原为无皇后状态
        }
        else { // 找到位置并放置
            if (rows == size - 1) { // 已经是最后一行，说明成功找到一组解
                Print(); // 打印
                numOfSolutions++; // 解数量++
                stack->Pop(lastX, lastY); // 弹出，从而进行下一轮解的寻找
                board[lastX][lastY] = '0';
            }
            else rows++, lastX = -1, lastY = -1; // 不是最后一行，继续往下一行寻找
        }
    }
```

3.2.3 求解功能的截屏示例

3.3 检验位置合法功能的实现

3.3.1 检验位置合法功能的流程图



3.3.2 检验位置合法功能的核心代码

```
// 条件一：同一列不能有两个及以上的皇后
for (int i = 0; i < rows; i++) {
    if (board[i][cols] == 'X') return false;
}
// 条件二：左斜线方向不能有两个及以上的皇后
for (int i = rows, j = cols; i >= 0 && j >= 0; i--, j--) {
    if (board[i][j] == 'X') return false;
}
// 条件三：右斜线方向不能有两个及以上的皇后
for (int i = rows, j = cols; i >= 0 && j < size; i--, j++) {
    if (board[i][j] == 'X') return false;
}
return true;
```

4 测试

4.1 功能测试

测试用例：启动程序后按照提示输入
预期结果：正常求解并给出结果
实验结果：

4.2 边界测试

4.2.1 数据过小测试

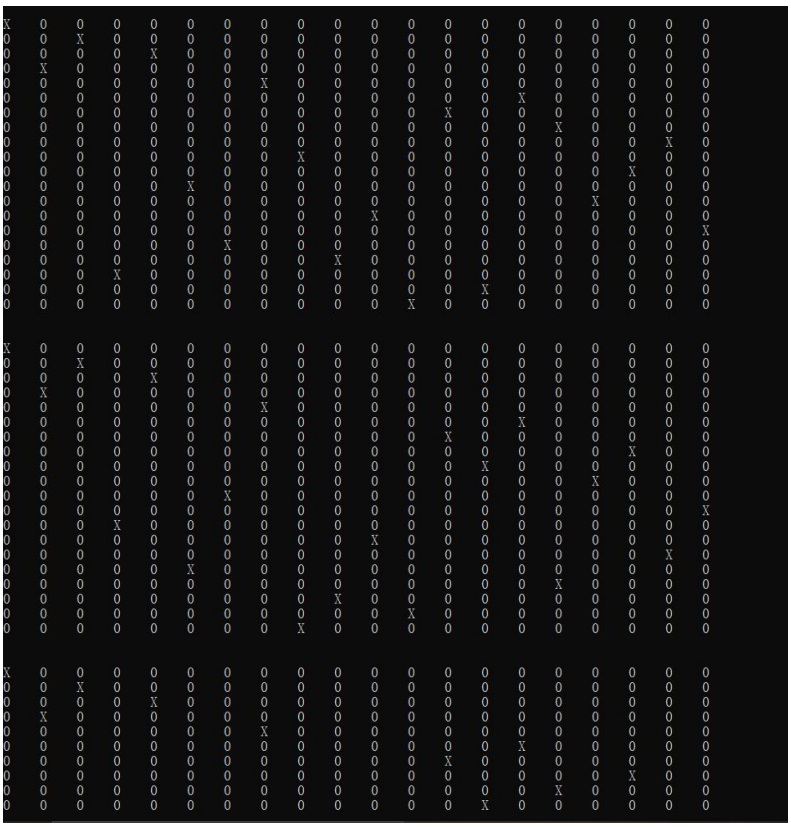
测试用例：输入 0,1 等
预期结果：程序会提示用户输入值应当大于 1，应当重新输入，程序不会异常终止或者崩溃。
实验结果：

4.2.1 数据过大测试

测试用例：输入很大的数据（如 20 等，通常数据过大后解的数量会非常多，因此会导致程序一直运行）

预期结果：虽然数据量庞大，程序会运行很久，但是由于程序代码比较规范，没有内存泄漏的地方，因此程序并不会崩溃。

实验结果：



因为解太多，所以程序一直在运行并输入解，但不会异常退出。