项目文档

谈瑞

项目四：N皇后问题

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 专业班级：软件工程4班  电话：18936361545 | 学号：1452775  电子邮件：tanrui106@163.com | 课目：数据结构课程设计  个人网站：http://guitoubing.top |
|  |  |  |

目录

[项目简介 3](#_Toc502844995)

[项目概要 3](#_Toc502844996)

[项目功能及要求 3](#_Toc502844997)

[项目结构 4](#_Toc502844998)

[项目类的实现 5](#_Toc502844999)

[Chessboard类 5](#_Toc502845000)

[Queen类 6](#_Toc502845001)

[主要代码分析 7](#_Toc502845002)

[寻找皇后 7](#_Toc502845003)

[运行测试 8](#_Toc502845004)

[正常运行 8](#_Toc502845005)

[错误检测 9](#_Toc502845006)

# 项目简介

## 项目概要

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种摆法。

本实验拓展了N皇后问题，即皇后个数由用户输入。

## 项目功能及要求

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行上，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，都依次对每一列进行检测，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。

# 项目结构

N皇后采用回溯法的思想，即试错，对于每一行的一个可能的位置进行错误检测，若符合条件则递归进行下一行的查找，直到找到一种可能结果或者中断某一个不可能结果。

本项目中采用三个数组valid\_left、valid\_right、valid\_line来标记不能被使用的位置集合，如下图所示：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **0** |  |  |  |  |  |  |  |  |
| **1** |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |
| **5** |  |  |  |  |  |  |  |  |
| **6** |  |  |  |  |  |  |  |  |
| **7** |  |  |  |  |  |  |  |  |

每放置一个皇后，其所在的左斜线、右斜线、列三者均被标记为不可用状态，即只能从剩下的未标记位置找皇后位置。易知，其整个棋盘左斜线与右斜线条数均为2N-1，且对于任意一个点（a,b）来说，其列号为：b、左斜线号为：a+b、右斜线号为：N+a-b-1（行号列号均是从0开始），这样在每次确定所查看皇后位置是否为可用时，只需要查看这三个数组对应位置是否被标记即可。而在回溯前又需要将此次位置的三个标记消除。这样避免了确定第i个皇后位置时需要对前i-1个已经放置好的皇后再进行一次遍历检测，大大节省了运行时间。下面看下16皇后问题以下的排列数量及运行时间：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| **摆法/种** | 1 | 0 | 0 | 2 | 10 | 4 | 40 | 92 |
| **时间/s** | <0.001 | <0.001 | <0.001 | <0.001 | <0.001 | <0.001 | <0.001 | 0.001 |
| **N** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| **摆法/种** | 352 | 724 | 2680 | 14200 | 73712 | 365596 | 2279184 | 14772512 |
| **时间/s** | 0.001 | 0.006 | 0.027 | 0.14 | 0.79 | 4.813 | 31.445 | 214.363 |

# 项目类的实现

## Chessboard类

|  |  |
| --- | --- |
| 类成员 | 作用 |
| Chessboard(); | 构造函数 |
| ~Chessboard(); | 析构函数 |
| Chessboard(int \_amount); | 带参数的构造函数，构造指定大小的（amount\*amount）棋盘 |
| void findQueen(int row); | 在第row行寻找皇后的递归函数 |
| void printQueen(); | 输出皇后 |
| inline int getCount(); | 获取所有摆法的数量 |
| inline double getTime(); | 获取某一棋盘获取摆法结束花费的时间 |
| vector<Queen\*> queen\_vec; | 存放符合条件的皇后的数组 |
| ~~int \*present\_queen;~~ | 原用于存放符合条件皇后的列号，与queen\_vec有重复，但在项目中未删除 |
| bool \*valid\_line,  \*valid\_left, \*valid\_right; | 标记列、左斜线、右斜线可用位置的数组 |
| int queens\_amount; | 皇后的数量 |
| int count; | 所有摆法的数量 |
| bool if\_print; | 是否输出（输出所有棋盘会大大降低运行时间） |
| clock\_t time\_begin, time\_end; | 计时器，开始时间和结束时间 |
|  |  |

## Queen类

|  |  |
| --- | --- |
| 类成员 | 作用 |
| Queen(); | 构造函数 |
| Queen(int \_x, int \_y); | 带参数的构造函数，创建一个横坐标为\_x，纵坐标为\_y的Queen |
| ~Queen(); | 析构函数 |
| int index\_x, index\_y; | 存放皇后的横纵坐标 |
|  |  |

# 主要代码分析

### 寻找皇后

void Chessboard::findQueen(int row){

    //寻找皇后的函数是一个利用编译器自身回溯机制的递归函数，编译器在某次递归结束后会自动返回上一次递归位置，并将结果带回，这种特性与N皇后问题恰好相符合。

    //回溯过程中，如何确定某一位置的皇后是否符合条件是N皇后问题所要解决的最主要的方面，这里采用三个标记数组，对每次添加的皇后更新相应位置的标记，使得下一次选择皇后的位置更具体，降低了暴力搜索的花费。

    for (int i = 0; i < queens\_amount; i++){

        if (valid\_left[row + i] || valid\_right[queens\_amount+row-i-1] || valid\_line[i]){

            //如果该位置已经被前i-1个皇后标记过，则直接跳到下一个位置查找

            continue;

        }else{

            //否则将其加入可用皇后数组中

            Queen\* temp = new Queen(row+1, i+1);

            queen\_vec.push\_back(temp);

            valid\_left[row+i] = valid\_right[queens\_amount+row-i-1] = valid\_line[i] = true;

            //将此时的列、左斜线、右斜线三个位置添加标记

            if (row == queens\_amount-1){

                //如果已经找到最后一行，则说明找到了一种摆法，摆法数量加1，并输出之（需要的话）

                count++;

                if (if\_print)   printQueen();

            }

            findQueen(row+1);

            //递归对下一行开始查找

            valid\_left[row+i] = valid\_right[queens\_amount+row-i-1] = valid\_line[i] = false;

            //回溯时将三个标记取消

            queen\_vec.pop\_back();

            //将该皇后弹出，以查找下一个皇后

            free(temp);

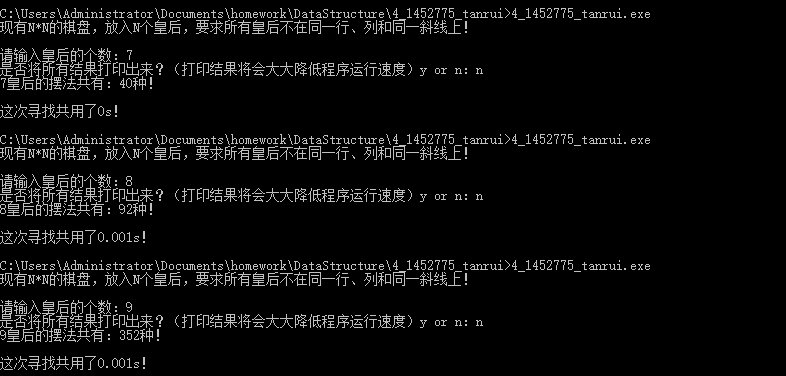
        }

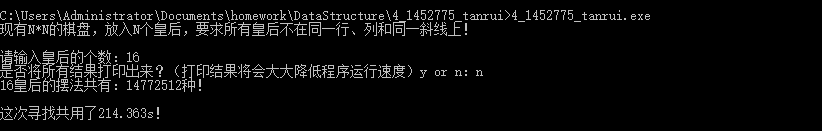
    }

}

# 运行测试

## 正常运行





## 错误检测