数据结构课程设计

项目说明文档

**N皇后问题**



同济大学

Tongji University

姓名： 林觉凯

学号： 2253744

指导老师： 张颖

学院专业： 软件学院 软件工程

**目录**

**1.项目分析-------------------------------------------------------------------------3**

**1.1 项目背景分析-------------------------------------------------------------------------3**

**1.2项目功能分析--------------------------------------------------------------------------3**

**1.2.1项目功能要求-------------------------------------------------------------------------------3**

**1.2.2项目输入要求-------------------------------------------------------------------------------3**

**1.2.3项目输出要求-------------------------------------------------------------------------------3**

**1.2.4项目实例-------------------------------------------------------------------------------------4**

**2.项目设计-------------------------------------------------------------------------4**

**2.1 数据结构设计-------------------------------------------------------------------------4**

**2.2算法设计--------------------------------------------------------------------------------5**

**2.3系统流程设计-------------------------------------------------------------------------5**

**3.项目实现------------------------------------------------------------------------5**

**3.1 Print\_Board函数的实现-----------------------------------------------------------6**

**3.2 Judgement函数的实现-------------------------------------------------------------6**

**3.3 main函数的实现--------------------------------------------------------------------8**

**4.项目代码功能测试------------------------------------------------------------9**

**4.1数据类型输入错误测试----------------------------------------9**

**4.2项目数据测试-----------------------------------------------------------------------10**

**5.项目心得与体会--------------------------------------------------------------11**

**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。

**1.2项目功能分析**

**1.2.1项目功能要求**

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。

**1.2.2项目输入要求**

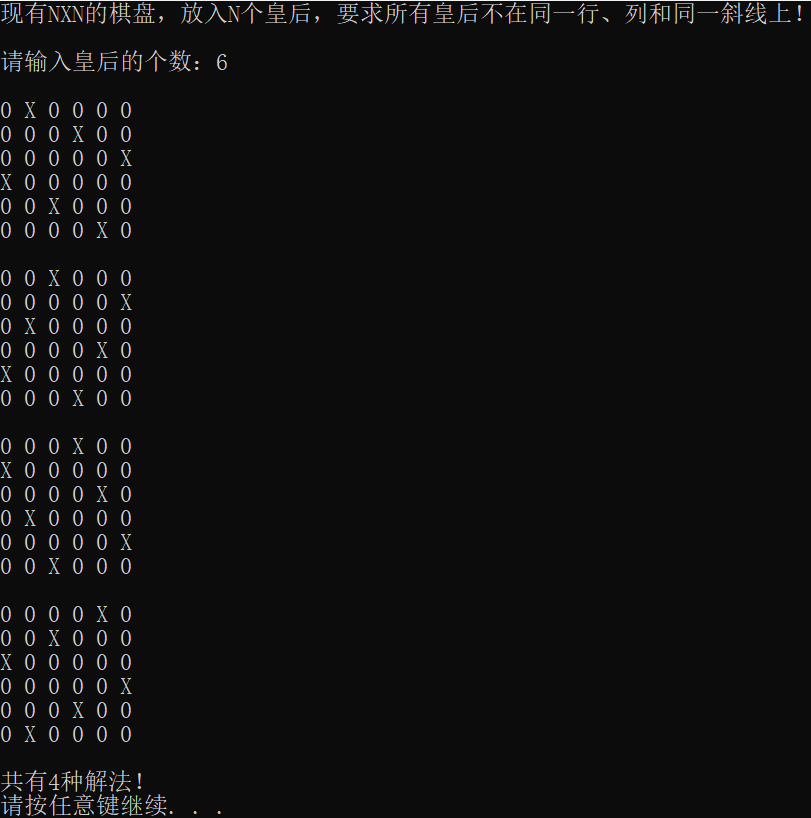
输入皇后的数量N。

(注意，当N逐渐变大时，N皇后的解法呈现指数级增长，本题所用的是动态数组内存分配，但是不建议N输入过大。经过测试，当N=12时，电脑运行解出所有解法的时间为50秒左右，所以建议N的输入不必过大)

**1.2.3项目输出要求**

依次输出在N皇后的情况下每一种棋盘的摆盘方式和最后的解法统计。

**1.2.4项目示例**



**2.项目设计**

**2.1 数据结构设计**

经过分析可以知道，N皇后问题需要判断摆放皇后的同一行、同一列、主对角线和次对角线这四个方向是否放置别的皇后。我们进行递归是一行一行进行递归的，所以在行这一方向我们不需要设置数组来判断。(因为我们每一行都是用for循环来遍历这一行的每一列)。同时在遍历某一行的时候如果有符合要求的皇后放置方法，我们需要将它的列号记录下来。

这样以来我们就需要四个数组：

1.int quesschess[N]，它用来记录遍历所在行的皇后应该摆放在第几列

2.bool column[N]，一个元素为bool类型的数组，column[i]用来判断在第i列是否安放了皇后，true为已经放置了皇后，false为没有放置皇后。

3.bool md[2\*N]，一个元素为bool类型的数组，md[k]用来判断在第k条主对角线是否安放了皇后，true为已经放置了皇后，false为没有放置皇后。

4.bool sd[2\*N]，一个元素为bool类型的数组，sd[i]用来判断在第k条次对角线是否安放了皇后，true为已经放置了皇后，false为没有放置皇后。

同时，这四个数组用动态内存分配来分配空间，避免内存的浪费。

**2.2算法设计**

N皇后问题使用递归回溯搜索法即深度搜索实现。

深度优先搜索算法（Depth First Search , DFS）的思想即回溯法（Backtrack），以下为回溯法思想的如下：在一定的约束条件下沿着某一个方向搜索前进，如果在这一个搜索前进的方向失败，便回头寻找其他搜索的方法继续搜索前进。

在N皇后问题的求解过程中，我们首先将第一个皇后放在第一行第一列，然后在下一行上放置一个皇后，直到每一行都能放置皇后，打印出一种结果。在向下一行放置皇后的时候，先从这一行的第一列开始放置皇后，再对每一列、主对角线和次对角线进行检验，如果这一列无法放置皇后，则向这一行的下一列放置皇后，继续进行上述检验。如果这一整行所有列的位置都不能放置皇后，则返回到上一行，选择上一行的下一列位置继续重新递归判断。每一次放置完皇后都打印出当前棋盘的布局解法的总数加1。

**2.3系统流程设计**

**程序开始**

**输入N**

**递归回溯寻找摆法**

**统计总数**

**打印棋盘**

**3.项目实现**

**3.1 Print\_Board函数的实现**

//打印棋盘函数，通过输入N(棋盘长宽)和数组queenchess(当前某行的皇后放置在某列)，输出所有棋盘摆法和解法总数

void Print\_Board(int N, int\* queenchess)

{

for (int row = 0; row < N; row++)

{

for (int column = 0; column < N; column++)

{

if (queenchess[row] != column)

{

cout << "0 "; //如果第row行的皇后不摆放在第column列，输出"0 "

}

else

{

cout << "X "; //如果第row行的皇后摆放在第column列，输出"X "

}

}

cout << endl;

}

cout << endl;

return;

}

Print\_Board函数是打印出每一个符合条件的棋盘的布局，通过输入棋盘的边长N和皇后摆放的位置queenchess数组，queenchess数组表示着在第几行的皇后应该摆放在第几列，比如，queenchess[1] = 3就说明第一行的皇后应该摆放在第三列，依次类推。通过两重循环遍历，外层循环为row的循环，即每一行的循环，内层循环为column的循环，为每一列的循环。如果第row行的皇后不摆放在第column列，则输出"0 "；如果第row行的皇后摆放在第column列，则输出"X "；内层循环每遍历完成，代表这一行已经完成遍历，输出换行；内层循环每遍历完成，代表每一种方式已经打印完成，也要输出换行。

**3.2 Judgement函数的实现**

//判断函数，用于递归计算，判断遍历行每个位置的所有皇后适合的摆放方法

void Judgement(int start\_row,int\* queenchess, bool\* column, bool\* md, bool\* sd)

{

if (start\_row == N) //如果递归完成了N-1行，说明已经完成了一种方法的计算，打印棋盘

{

Print\_Board(N, queenchess);

Totalmethods++; //每打印一种解法，解法总数加1

return;

}

for (int i = 0; i < N; i++) //遍历这一行所对应的每一列

{

if (!column[i] && !md[start\_row + i] && !sd[start\_row - i + N])

//如果在第start\_row行第i列放置皇后和其他位置没有冲突

{

column[i] = md[start\_row + i] = sd[start\_row - i + N] = true;

//放置皇后，并且在相应位置标志true

queenchess[start\_row] = i; //记录第start\_row行的皇后位置

Judgement(start\_row + 1,queenchess, column, md, sd); //继续遍历下一行

}

else

{

continue;

//如果在第start\_row行第i列放置皇后有冲突，直接换到此行的下一列

}

column[i] = md[start\_row + i] = sd[start\_row - i + N] = false;

//恢复棋盘，以免影响下一次遍历

}

}

Judgement函数用来递归计算，判断遍历行每一个位置的所有皇后的合适摆放方法。通过传入五个参数，第一是递归的起始条件start\_row，我们在主函数传入的是第0行，即从头开始遍历。第二是queenchess函数，它用来储存在每一种合适的棋盘布局的皇后的所在位置。第三是column函数，它用来判断遍历所在同一列的位置是否有皇后存在，第四是md函数，它用来判断遍历所在位置的主对角线的位置是否有皇后存在。第五是sd函数，它用来判断遍历所在位置的次对角线的位置是否有皇后存在。在递归开始时，先判断遍历行数是否已经到达了N，如果是的话，说明遍历完成，调用Print\_Board函数打印一种布局；如果还未便利完成，则判断当前位置是否合适摆放皇后，如果合适的话，将其列、主对角线和次对角线位置设为true，说明接下来的遍历皇后需要考虑到这些位置。然后继续递归下一行进行判断。最后恢复棋盘，将每一个位置设为false(清空棋盘)，换下一列再继续遍历。

**3.3 main函数的实现**

//主函数，初始化皇后数量，创建并初始化三个数组

int main()

{

cout << "现有NXN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！";

cout << endl << endl;

cout << "请输入皇后的个数：";

while (1)

{

cin >> N;

if (cin.good() == 0 || N < 1)

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "输入不合法，请重新输入：";

continue;

}

break;

} //代码的健壮性，确保输入的N合法

cout << endl;

int\* queenchess = new int[N];

//动态数组queenchess，用来存放某一行的皇后放置在第几列

bool\* column = new bool[N];

//动态数组column，column[i]用于标识第i列是否放置了皇后

bool\* md = new bool[2 \* N];

//动态数组md，md[k]用于标识第k条主对角线是否放置了皇后

bool\* sd = new bool[2 \* N];

//动态数组sd，sd[k]用于标识第k条主对角线是否放置了皇后

if (queenchess == NULL || column == NULL || md == NULL || sd == NULL)

{

cout << "内存分配失败！" << endl;

system("pause");

return 0;

} //对内存分配进行错误处理，以免出现空间分配失败情况

for (int i = 0; i < N; i++)

{

column[i] = false; //初始化动态数组queenchess，开始时棋盘为空，未放置皇后

}

for (int i = 0; i < 2\*N; i++)

{

md[i] = sd[i] = false; //初始化动态数组md和sd，开始时棋盘为空，未放置皇后

}

Judgement(0,queenchess, column, md, sd); //从第0行开始递归计算所有解法

cout << "共有" << Totalmethods << "种解法！";

delete[] queenchess; //释放动态数组queenchess

delete[] column; //释放动态数组column

delete[] md; //释放动态数组md

delete[] sd; //释放动态数组sd

system("pause");

return 0;

}

主函数的作用是输入N皇后N的值，并且通过动态内存分配创建并初始化我们在后续所需要的三个数组。在输入N皇后N的值的时候，我们对输入的数据进行的检验，如果输入的不是大于0的正整数便会有相应的提示。queenchess数组表示着在某一行的皇后所摆放的位置，一共有N种摆放方法，所以我们初始化queenchess数组的长度为N；column数组用于标识某一列是否放置了皇后，在NXN的棋盘中，最多有N列，所以我们初始化column数组的长度为N；md数组标识主对角线是否放置了皇后，在NXN的棋盘中，一共有2\*N条主对角线，所以我们初始化md数组的长度为2\*N；sd数组标识主对角线是否放置了皇后，在NXN的棋盘中，一共有2\*N条次对角线，所以我们初始化sd数组的长度为2\*N。同时，我们也要初始化这几个动态数组的内容。一开始的棋盘是还没有放置皇后的，所以对于column、md和sd这几个存储内容为bool类型的数组来说，我们都要将其初始化为false。在main函数的最后释放这几个数组。

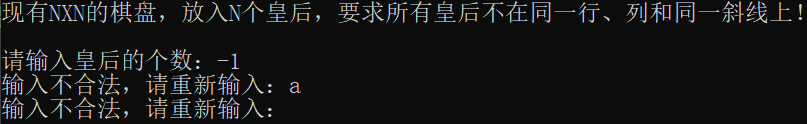
**4.项目代码功能测试**

**4.1数据输入错误测试**

**测试用例：**输入N皇后的N为非正数或不为int型

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

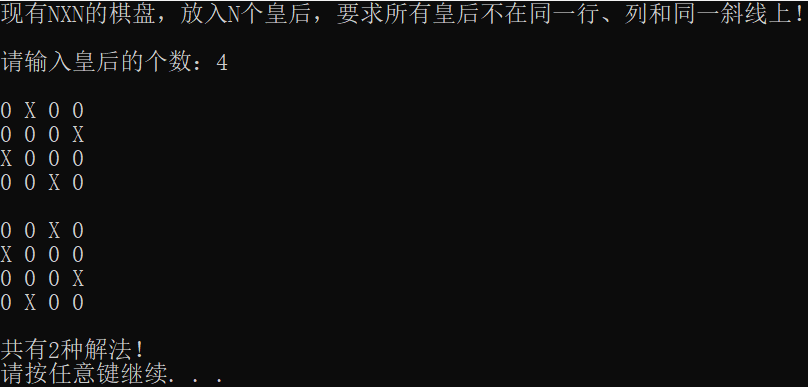
****

**4.2项目数据测试**

**测试用例1：**输入N = 4

**预期结果：**最后有2种解法，并且打印出所有解法的棋盘布局

**实验结果：**

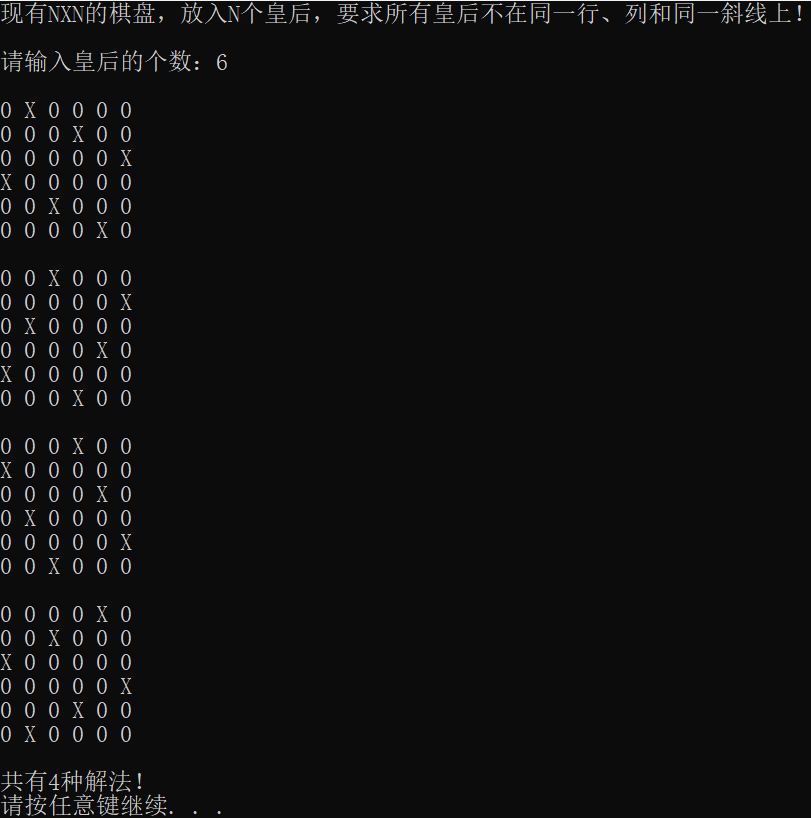
****

**测试用例2：**输入N = 6

**预期结果：**

最后有4种解法，并且打印出所有解法的棋盘布局

**实验结果：**如右图



**5.项目心得与体会**

N皇后问题是一道经典的递归回溯法例题，也是我初步遇见的递归题目之一。在编写程序一开始的时候，我遇到了两个问题：第一是递归的终止条件不知道设置在哪里，最后我在递归函数的开始加了一个判断：如果搜索的行数到达了N的时候就输出一种布局；第二是我一开始忘记了将每次递归后的棋盘重置，即没有将column[i] = md[start\_row + i] = sd[start\_row - i + N] = false；经过细读老师给的PPT内容和debug之后才发现了问题的所在。

在这次项目完成之后，我对动态数组的创建这一方法运用得更加熟练，最重要的是，我对递归回溯法的运用获得了一定的认识，递归算法的思想比较复杂，我还需要更多的练习加以巩固！