项目说明文档

数据结构课程设计

——N皇后问题

作 者 姓 名： 刘畅

学 号： 2054164

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](#_Toc84779144)

[1.1背景分析 1](#_Toc84779145)

[1.2功能分析 1](#_Toc84779146)

[2 设计 1](#_Toc84779147)

[2.1 算法设计 1](#_Toc84779148)

[2.2 结构设计 2](#_Toc84779149)

[3 实现 2](#_Toc84779150)

[3.1 递归搜索函数的实现 2](#_Toc84779151)

[3.2 输出函数的实现 2](#_Toc84779152)

[3.3 main函数的实现 3](#_Toc84779153)

[4 测试 3](#_Toc84779154)

[4.1 测试1 3](#_Toc84779155)

[4.2 测试2 4](#_Toc84779156)

[4.3 测试3 4](#_Toc84779157)

# 1 分析

## 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中摆法。

本实验拓展了N皇后问题，即皇后个数由用户输入。

## 功能分析

输入棋盘大小N\*N，输出所有的放置方法，其中空格子用0表示，放置皇后的格子用X表示，最后输出总的方法数。其中N处于

# 2 设计

## 2.1 算法设计

N皇后问题的经典解决方法是深度优先搜索（即回溯法）：首先在第一行放置皇后，然后再放置下一行的皇后；对于某一行皇后的放置过程，将列从1-N循环，先判断是否与前面已放置的皇后冲突（如处在同一列、同一对角线），若不冲突则放置，然后继续寻找下一行皇后的位置；由于是按照行的顺序来放置皇后的，因此不用检测行中是否会出现多个皇后，只用检测列。

为了检测皇后是否冲突的时候更加快速，采用辅助数组的形式，其中当column[j]为true则表示第j列已经被放置了皇后，当main\_diagonal[i - j + N]（sub\_diagonal[i + j]）true时则表示(i,j)所在的主（副）对角线已经被放置了皇后。由此，我们可以在搜索下一个皇后(i,j)的时候直接判断是否合理，若不合理直接丢弃。

## 2.2 结构设计

用三个辅助数组记录皇后的放置情况（列、主副对角线上是否已经被放置了皇后），用location[i]表示第i行的皇后被放置的列数，搜索回溯过程采用递归实现。

## 2.3 复杂度分析

N皇后递归回溯法的复杂度极高，接近，因此当输入的时需要运行极长时间才能得到全部答案。本题开设的最大棋盘为15\*15，因为输出

# 3 实现

## 3.1 递归搜索函数的实现

void search(int i) {

//若已经找到了第N + 1行，则输出

if (i == N + 1) {

print();

}

for (int j = 1; j <= N; j++) {

//对于当前的第i行，去遍历列

//若该列没有被放置过皇后、且主副对角线都没有被放置过皇后

//则在该位置放置皇后，并将相关辅助数组都进行标记

//标记过后去寻找第i + 1行皇后放置的位置

//回溯的时候将标记的值更改回来

if (column[j] || sub\_diagonal[i + j] || main\_diagonal[i - j + N]) {

continue;

}

location[i] = j;

column[j] = true;

sub\_diagonal[i + j] = true;

main\_diagonal[i - j + N] = true;

search(i + 1);

column[j] = false;

sub\_diagonal[i + j] = false;

main\_diagonal[i - j + N] = false;

}

}

## 3.2 输出函数的实现

void print() {

sum++;//先将总方案数加一

for (int i = 1; i <= N; i++) {

for (int j = 1; j <= N; j++) {

//准备输出一个N \* N矩阵

if (j == location[i]) {

//所有皇后的位置：(i,location[i])

//若(i,j)==(i,location[i])，则输出皇后，不然则输出0

cout << "X ";

}

else {

cout << "0 ";

}

}

cout << endl;

}

cout << endl;

}

## 3.3 main函数的实现

int main() {

//初始化为零和false

memset(location, 0, sizeof(location));

memset(column, false, sizeof(column));

memset(sub\_diagonal, false, sizeof(sub\_diagonal));

memset(main\_diagonal, false, sizeof(main\_diagonal));

cout << "请输入棋盘大小（一个4~15的正整数）：\n";

cin >> N;//输入棋盘大小

while (N < 1 || N > 15) {

cout << "输入的N不合法，请重新输入！\n";

cin >> N;

}

cout << "请选择是否要打印棋盘，1为打印，0为不打印：\n";

cin >> p;

search(1);//从第一行开始找

cout << "sum=" << sum << endl;

return 0;}

# 4 测试

## 4.1 测试1

**测试用例：**

**8**

实验结果：

X 0 0 0 0 0 0 0

0 0 0 0 X 0 0 0

0 0 0 0 0 0 0 X

0 0 0 0 0 X 0 0

0 0 X 0 0 0 0 0

0 0 0 0 0 0 X 0

0 X 0 0 0 0 0 0

0 0 0 X 0 0 0 0

……（由于篇幅问题只放置一种）

sum=92

## 4.2 测试2

**测试用例：**

**15**

实验结果：

X 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 X 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 X 0 0 0 0 0 0 0 0 0 0

0 X 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 X 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 X 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 X 0

0 0 0 X 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 X 0 0

0 0 0 0 0 0 0 0 X 0 0 0 0 0 0

0 0 0 0 0 X 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 X

0 0 0 0 0 0 X 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 X 0 0 0 0

0 0 0 0 0 0 0 X 0 0 0 0 0 0 0

……（由于篇幅问题只放置一种）

sum= 2279184

## 4.3 测试3

**测试用例：**

**5**

实验结果：

X 0 0 0 0

0 0 X 0 0

0 0 0 0 X

0 X 0 0 0

0 0 0 X 0

X 0 0 0 0

0 0 0 X 0

0 X 0 0 0

0 0 0 0 X

0 0 X 0 0

0 X 0 0 0

0 0 0 X 0

X 0 0 0 0

0 0 X 0 0

0 0 0 0 X

0 X 0 0 0

0 0 0 0 X

0 0 X 0 0

X 0 0 0 0

0 0 0 X 0

0 0 X 0 0

X 0 0 0 0

0 0 0 X 0

0 X 0 0 0

0 0 0 0 X

0 0 X 0 0

0 0 0 0 X

0 X 0 0 0

0 0 0 X 0

X 0 0 0 0

0 0 0 X 0

X 0 0 0 0

0 0 X 0 0

0 0 0 0 X

0 X 0 0 0

0 0 0 X 0

0 X 0 0 0

0 0 0 0 X

0 0 X 0 0

X 0 0 0 0

0 0 0 0 X

0 X 0 0 0

0 0 0 X 0

X 0 0 0 0

0 0 X 0 0

0 0 0 0 X

0 0 X 0 0

X 0 0 0 0

0 0 0 X 0

0 X 0 0 0

sum=10