项目说明文档

——N皇后问题

姓名：吴桐欣

学号：1652677

同济大学 软件学院 软件工程专业

目录

**1.项目概述 4**

**1.1项目简介 4**

1.1.1功能分析 4

**1.2文件目录 4**

**1.3操作指南 4**

**1.4注意事项 5**

**2.思路与设计 5**

**2.1基本思路 5**

**2.2设计 5**

2.2.1数据结构 5

2.2.2成员与成员函数 5

**3.具体实现 6**

**3.1函数的实现 6**

3.1.1 onList()函数 6

函数代码 6

说明 6

3.1.2 initialize()函数 7

函数代码 7

说明 7

3.1.3 addPoint()函数 7

函数代码 7

说明 8

3.1.4 deletePoint()函数 8

函数代码 8

说明 8

**3.2回溯法的实现 9**

回溯法流程图 9

**4.测试 9**

**4.1边界测试 9**

**4.2出错测试 9**

1.项目概述

1.1项目简介

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。

本实验拓展了N皇后问题，即皇后个数由用户输入。

1.1.1功能分析

本项目要求对N皇后问题求解，最直接的功能就是

（1）输出所有的解题方案；

（2）计算出最多有多少种方案。

1.2文件目录

（1）P04\_1652677\_吴桐欣\_说明文档.docx（本文档）

（2）P04\_1652677\_吴桐欣.exe（可执行文件）

（3）P04\_1652677\_吴桐欣.cpp（源文件）

（4）P04\_1652677\_吴桐欣.h（头文件）

1.3操作指南

运行程序后，将获得程序提示“输入皇后个数n(0<n<11):”

用户输入一个数字，即可获得结果

[输入样例]

4

[输出样例]

皇后摆法：

[方案 #1]

0 x 0 0

0 0 0 x

x 0 0 0

0 0 x 0

[方案 #2]

0 0 x 0

x 0 0 0

0 0 0 x

0 x 0 0

共有 2 种解法

1.4注意事项

（1）用户不得输入除数字以外的字符

（2）用户所输入的数字n需符合程序要求，大于0且小于11

2.思路与设计

2.1基本思路

此题用到回溯法，用栈对已放置的皇后进行存储。回溯时，依次弹出栈内的皇后。

2.2设计

2.2.1数据结构

为了存储及读取数据的方便，用vector来存储所放置的皇后的位置，vector的back()和pop\_back()函数可以模拟stack的top()和pop()函数。

2.2.2成员与成员函数

为了存储数据直观，用Point结构将棋盘上每一个位置的行(x)、列(y)数都放在一个Point中，使一个棋盘位置只对应一个Point。

struct Point{

**int** x;*//*行

**int** y;*//*列

};

每一个皇后放置之后，都会对后面的皇后的放置造成影响，为了方便检查棋盘上的位置是否可放，用List结构包装4个vector数组，为节省空间，存储值为bool值。值为true表示这一行或这一列或这一对角线上没有皇后，值为false则表示已经放有皇后。

struct List{

**int** n;//棋盘大小

vector<**bool**> row;*//*行

vector<**bool**> col;*//*列

vector<**bool**> mainDiag;//主对角线（左上-右下）

vector<**bool**> minorDiag;//副对角线（右上-左下）

**bool** **onList**(Point p)**const**;//检查某一位置是否可放皇后

**void** **initialize**();*//*初始化*4*个数组

**void** **addPoint**(Point p);//新增一个皇后，更新数组

**void** **deletePoint**(Point p);//回溯时移走一个皇后，更新数组

};

initialize(),addPoint(),deletePoint()函数是对内部数据进行修改，也代表着棋盘上的变化。

onList()函数是外部对内部数据的读取、比较、做出判断，不对任何值做修改，因此设置为const。

3.具体实现

3.1函数的实现

3.1.1 onList()函数

函数代码

**bool** **List**::onList(Point p)**const**

{

**if** (row[p.x] == **true**

&& col[p.y] == **true**

&& mainDiag[p.x-p.y+n-1] == **true**

&& minorDiag[p.x+p.y] == **true**) {

//当且仅当p位置在4个数组中对应元素值均为true时，p位置才可放置皇后

**return** **true**;

} **else** {

**return** **false**;

}

}

说明

返回值为bool值，返回true表示位置p可以放置皇后，返回false表示不可以。  
主对角线上，所有位置的x与y的差是相等的，x-y最小值是-(n-1)，要使每一条对角线在数组中有对应的下标，需要将x-y值全部加上(n-1)；  
副对角线上，所有位置的x与y的和是相等的，x+y的取值范围是0~(2n-2)，可直接作为数组对应的下标。

3.1.2 initialize()函数

函数代码

**void** List::initialize(){

**for** (**int** i=0; i<n; i++) {

row.push\_back(true);

col.push\_back(true);

}

**for** (**int** i=0; i<(2\*n-1); i++) {

mainDiag.push\_back(true);

minorDiag.push\_back(true);

}

}

说明

根据数组大小的不同，用两个for循环对数组进行初始化

3.1.3 addPoint()函数

函数代码

void List::addPoint(Point p){

row[p.x] = false;

col[p.y] = false;

mainDiag[p.x-p.y+n-1] = false;

minorDiag[p.x+p.y] = false;

}

说明

放置一个皇后，其所在的行、列、对角线设置为false，表示相应的行、列、对角线上不能再放皇后。

3.1.4 deletePoint()函数

函数代码

void List::deletePoint(Point p){

row[p.x] = true;

col[p.y] = true;

mainDiag[p.x-p.y+n-1] = true;

minorDiag[p.x+p.y] = true;

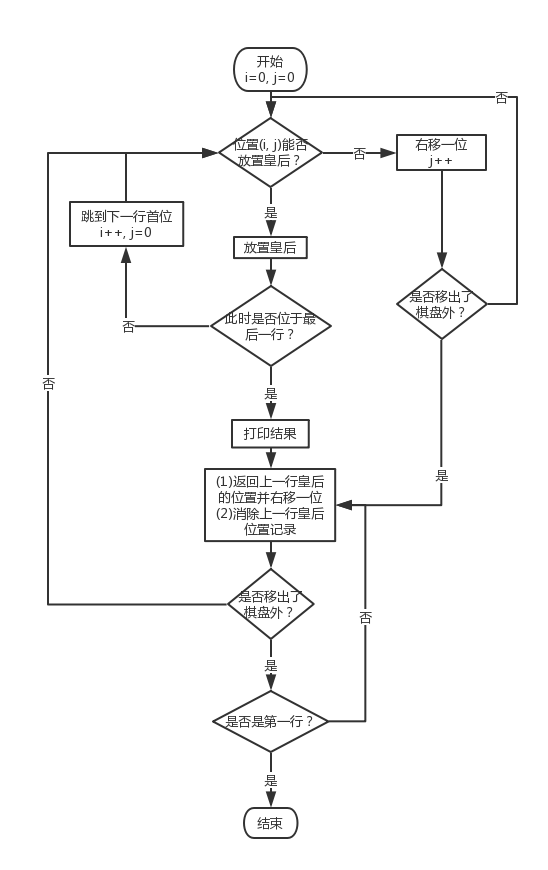
}

说明

移走一个皇后，其所在的行、列、对角线设置为true，表示相应的行、列、对角线上可以放皇后。

3.2回溯法的实现

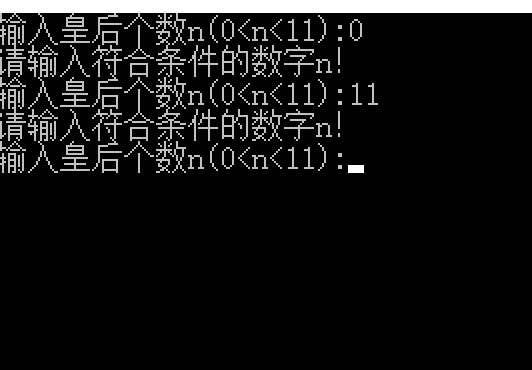
回溯法流程图



4.测试

4.1边界测试

输入数字太大或太小



4.2出错测试

输入非法字符

