数据结构课程设计

项目说明文档

N皇后

|  |  |
| --- | --- |
| 作者姓名： | 高逸轩 |
| 学 号： | 2053385 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

# 1项目分析

## 1.1 项目需求分析

本程序完成了以下需求：

* 执行效率高

能够存储的数据条数应该足够多，使用的数据结构和算法应该具有较低的时间和空间复杂度。

* 健壮性

对于错误的输入（输入数据不合法或超出算力范围），应当给出提示。

# 1.2 项目要求

### 1.2.1 功能要求

本项目完成了经典“N皇后问题”的求解，即在N\*N的棋盘上，安放N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。

### 1.2.2 输入格式

棋盘大小N

### 1.2.3 输出格式

全部解的棋盘摆放方式、解的总数量

### 1.2.4 项目示例

# 2 项目设计

## 2.1 类设计

为方便进行输入、搜索、展示等功能，并封存游戏所有数据，实现了ChessBoard棋盘类，代码部分如下：

// 棋盘类

class ChessBoard

{

friend void SetQueue(ChessBoard& queueProblem, const int currentLine);

public:

// 构造函数，设置数据成员信息

ChessBoard(const int n = 0)

// 析构函数，释放申请的空间

~ChessBoard()

// 得到答案计数器的值

int GetCounter()const { return counter; }

private:

int size; // 记录棋盘大小

int\* line; // 行

bool\* col; // 列

bool\* leftDiagonal; // 左对角线

bool\* rightDiagonal;// 右对角线

int counter; // 记录答案总数

};

## 2.2 算法设计

本题要求使用回溯搜索法即深度优先搜索实现，以下为深度优先搜索介绍：

深度优先搜索算法（Depth First Search , DFS）的思想即回溯法（Backtrack），即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路，则沿原路返回前一点，换下一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路，或无路可走又返回入口点。

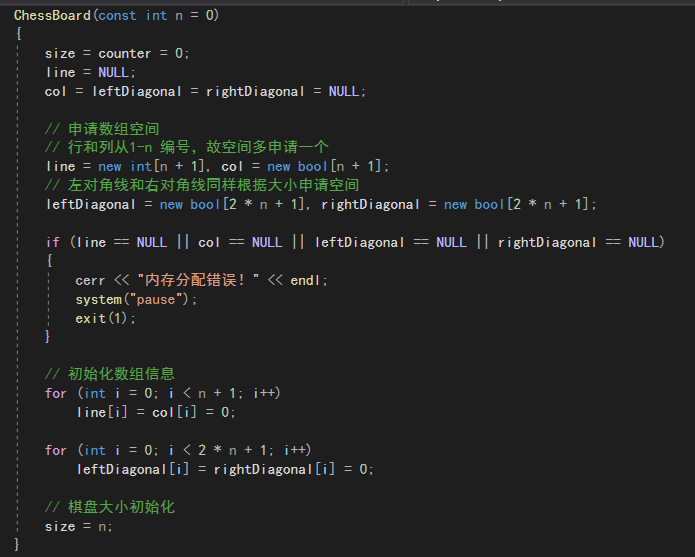
在本题的求解过程中，首先将第一个皇后放置在第一行第一列，然后，依次在下一行上放置一个皇后，直到N个皇后全部放置成功。在放置每个皇后时，都依次兑每一列进行检测，首先检测放在第一列是否与已放置的皇后冲突，如不冲突，则将皇后放置在该列，否则，选择改行的下一列进行检测。如整行的N列都冲突，则回到上一行，重新选择位置，依次类推。每次放置完全部的皇后棋之后，输出本次棋盘的布局。

**另外，在文章末尾，使用dancing links同样解决了此题，附在最后一部分内容中。**

## 2.3 项目功能流程图

# 3 核心代码详解

## 3.1 初始化棋盘信息

 在构造函数中，根据输入的棋盘大小，对代表行信息的line、列信息col、左对角线leftDiagonal、右对角线rightDiagonal 数组进行初始化。注意要将两条对角线数组元素的个数与棋盘大小找到合适的对应关系。另外对内存分配进行了错误处理，避免出现空间异常。

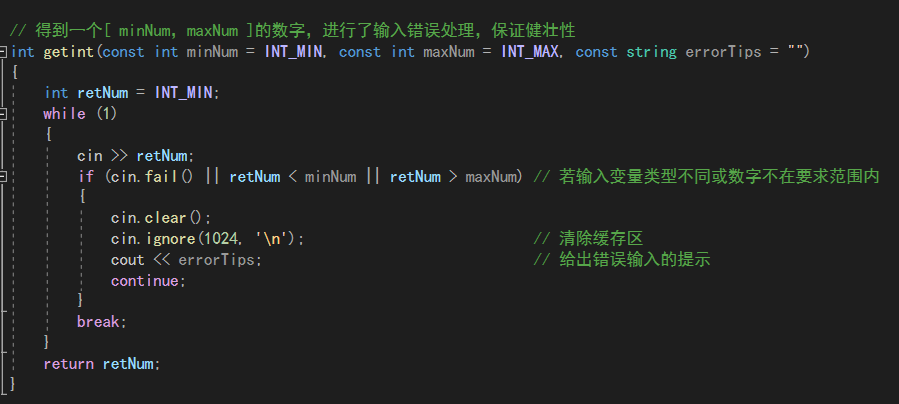
## 3.2 搜索结果

对于大小为N的棋盘，若当前已经遍历到了第N+1行，说明前N行全部已经设置结束，已经得到了一组合法的解。此时应该将这组解输出，同时令解的个数+1，之后回溯，不必再搜索下一行。

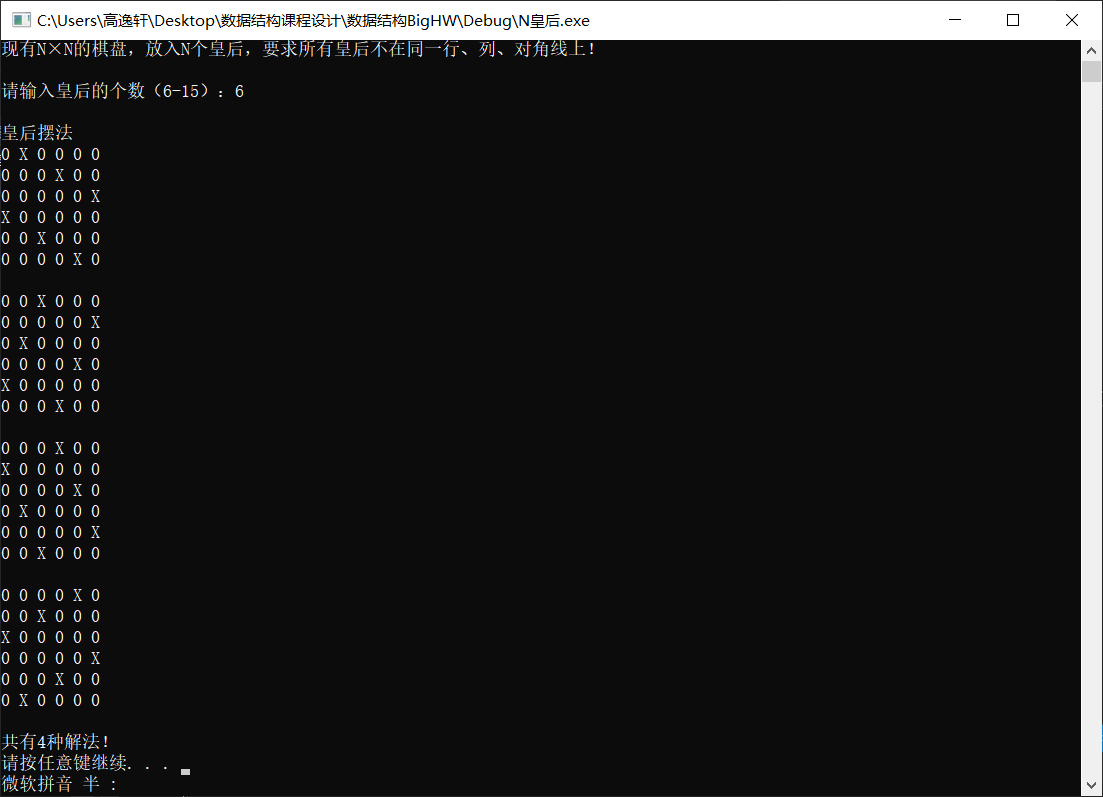
若本组解未设置结束，则遍历当前行的所有列：如果当前行当前列上均没有棋子、当前位置两条对角线上均没有棋子，则此处位置合法，可以作为一个安放位置。将其所在行、列、两条对角线标记为已经被占领，向下一行进行搜索。在第i列搜索结束后，要将列、对角线信息数组重置为未被占领，以便于在当前行的下一列进行搜索。

# 4 项目测试

## 4.1 健壮性测试

以上为两种错误信息的处理，一种为输入的数据类型与所需类型不同，一种为数据范围不合法，这种类型的输入错误通过自行编写的getint()函数实现。

## 4.2 功能测试

为方便老师测试，提供了文件4\_test.txt，内含一组数据，测试了本程序的全部功能。以下为一组测试数据：

# 5 心得与体会与扩展解法

八皇后是一道经典的DFS深搜例题，但我们可以看到，在数据规模偏大的情况下，由于其时间复杂度的限制，DFS算法不能较高效率的执行问题。在这里，有一种课上简单介绍过的数据结构可以解决这个问题：**十字链表实现Dancing Links。**

我们知道，**dancing links的用途是解决精确覆盖问题和重复覆盖问题**。由于篇幅限制，对精确覆盖问题不再进行介绍，如老师有需要请麻烦移步博客：

<https://www.cnblogs.com/grenet/p/3145800.html>

**而对于八皇后问题，我们可以把它转化为精确覆盖的问题：**

1. 每行只能放一个皇后
2. 每列只能放一个皇后
3. 每一个“/”斜行只能放一个皇后
4. 每一个“\”斜行只能放一个皇后

我们把每个状态当成列，每个点当成一行 然后精确覆盖这个矩阵。

注意：与模版精确覆盖问题不同的是，我们只需保证每行每列完全覆盖，而每一斜行是肯定不能完全覆盖的（因为一共2\*n-1个斜行）。

下附代码：（由于是高中信息学竞赛时编写，命名规范、面向对象等方面完成的不佳，可读性不高，请老师海涵）

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

#define mx 100100//要大于(n\*n)\*(6\*n-2)解释见主函数

long long sum=0;//方案总数

int nn;

struct out{

int ans[14];

}ss[100000];//记录答案的数组

struct DLX{

int n,m,cnt;//矩阵的长，宽，点的数量

int l[mx],r[mx],u[mx],d[mx],row[mx],col[mx];//每个点的左，右，上下，行，列信息

int h[mx];//每行的头结点

int s[mx];//每列的结点数

int ansk[mx];//选了ansk[]

void init(int \_n,int \_m){

n=\_n,m=\_m;

int i;

for(i=0;i<=m;i++){

r[i]=i+1;l[i]=i-1;u[i]=d[i]=i;

}

r[m]=0;//m右边是0

l[0]=m;//0左边是m

memset(h,-1,sizeof(h));

memset(s,0,sizeof(s));

cnt=m+1;//开始时有m个结点（0结点和各列头结点）

}//初始化，生成每列的头

inline void link(int R,int C){

s[C]++;

row[cnt]=R;

col[cnt]=C;

u[cnt]=C;

d[cnt]=d[C];

u[d[C]]=cnt;

d[C]=cnt;

if(h[R]<0)h[R]=r[cnt]=l[cnt]=cnt;//该行没有别的点,把第一个加入的点作为该行的行头结点

else{

r[cnt]=h[R];

l[cnt]=l[h[R]];

r[l[h[R]]]=cnt;

l[h[R]]=cnt;

}

cnt++;

}//在r行c列插入点

inline void remove(int c){

r[l[c]]=r[c],l[r[c]]=l[c];

for(int i=d[c];i!=c;i=d[i]){

for(int j=r[i];j!=i;j=r[j]){

u[d[j]]=u[j];

d[u[j]]=d[j];

s[col[j]]--;

}

}

}//删除c列和c列上有点的行

inline void resume(int c){

for(int i=u[c];i!=c;i=u[i]){

for(int j=l[i];j!=i;j=l[j]){

u[d[j]]=j;

d[u[j]]=j;

s[col[j]]++;

}

}

r[l[c]]=c;

l[r[c]]=c;

}//恢复c列和c列上有点的行

void dance(int deep){

if(r[0]>nn){//前nn列都已删除表示已经放完了

sum++;

int x,y;

for(register int i=0;i<deep;i++){

x=ansk[i]%nn;

y=(ansk[i]-1)/nn+1;

if(x==0)x=nn;

ss[sum].ans[x]=y;

}

return;

}

int c=r[0];

register int i,j;

for(i=r[0];i<=nn;i=r[i])if(s[i]<s[c])c=i;//找到点最少的列，只需看前nn列是否删完

remove(c);

for(i=d[c];i!=c;i=d[i]){

ansk[deep]=row[i];

for(j=r[i];j!=i;j=r[j]) remove(col[j]);

dance(deep+1);

for(j=l[i];j!=i;j=l[j]) resume(col[j]);

}

resume(c);

return ;

}

}dlx;

bool cap(out a,out b){

int i=1;

while(a.ans[i]==b.ans[i]&&i<=nn){

i++;

}

return a.ans[i]<b.ans[i];

}//根据题目要求找出前3种解法

int main(){

scanf("%d",&nn);

dlx.init(nn\*nn,6\*nn-2);

//一共n\*n格对应n\*n行

//对于第m列

//若[1,n]对应在第m行放皇后

//若[n+1,2\*n]对应在第m-n列放皇后

//若[2\*n+1,4\*n-1]对应在第m-2\*n“\”斜行放皇后 （共2\*n-1个“\”斜行）

//若[4\*n,6\*n-2]对应在第m-4\*n+1“/”斜行放皇后（共2\*n-1个“/”斜行）

int x;

register int i,j;

int o=0;

for(i=1;i<=nn;i++){

for(j=1;j<=nn;j++){

o++;//第o行对应坐标(o/n+1,o%n)

dlx.link(o,i);

dlx.link(o,j+nn);

dlx.link(o,i-j+3\*nn);

dlx.link(o,i+j+4\*nn-2);

}

}

dlx.dance(0);

sort(ss+1,ss+1+sum,cap);

for(i=1;i<=3;i++){

for(j=1;j<=nn;j++)printf("%d ",ss[i].ans[j]);

printf("\n");

}

printf("%lld",sum);

return 0;

}

由于篇幅原因，报告内还有很多内容与解释没有展示，请老师和助教老师再移步源程序，在其中的注释写了每一步过程的详解。