**数据结构实验报告**

**实验二 求解皇后问题**

学 生 姓 名 ：杨仕振

学 号 ：2015301500349

二○一六年十月

**一、简介**

编写一个程序求解皇后问题：在n\*n的方格棋盘上，放置n个皇后，要求每个皇后不同行、不同列、不同左右对角线。要求由用户来输入皇后的个数，不能超过20；并且要求输出所有的解。（提示：采用类似于栈求解迷宫问题的方法）

**二、算法说明**

程序中使用的数据结构如下：

typedef struct {

int x;

int y;

}chess;

“棋子”，用它在“棋盘”上的坐标来表示

typedef struct {

chess c[Maxsize];

int top;

}stack;

用来记录可行组合的栈

程序中用到的函数：

void printpath(stack \*&s, int n);//用以打印符合条件的路径

void Push(stack \*&s, int a, int b);//用以入栈（出栈用s->top—代替）

void scangrid(stack \*&s, int n);//进行扫描和判断是否可以入栈或打印的核心函数

基本思想：

假象一个n\*n的棋盘,从左向右为横轴，坐标为0~n-1；

从上而下为纵轴，坐标为0~n-1；以左上角为（0,0）

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | .. | n-1 |
| 0 |  |  |  |
| .. |  |  |  |
| n-1 |  |  |  |

下面是对核心的scangrid的分析：

void scangrid(stack \*&s, int n)

{

用a记录当前位置横坐标，b记录纵坐标

while(!(栈空且当前位置“达到”第一行第n+1个格子))

{

for (a = 0; a < n; a++)//每次将当前格子的横坐标向右移动一个坐标

{

遍历栈中每个已经存入的格子，//备注：这里用计数变量i来判断是否和已存格子存在冲突

if 格子不与任何一个已有格子同列或同对角线，

则入栈，并将当前位置推向下一行，

{

if 栈已满

则打印，

{

while 当前位置不是第一行第n个格子时

将当前位置置为栈顶格子，并出栈；

}

continue

else 跳出a的循环，从下一行开始扫描“棋盘”；

}

else 说明冲突

{

while 当前位置不是第一行第n个格子时

将当前位置置为栈顶格子，并出栈；

}

}

}

}

**三、测试结果**

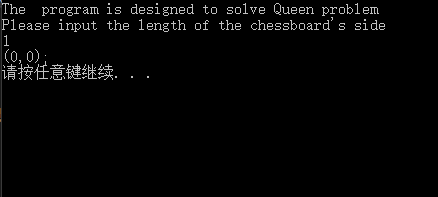


图1 当n=1时的输出

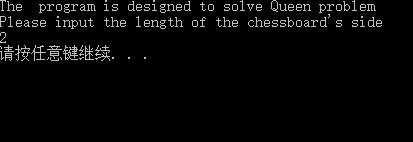


图2 当n=2时的输出

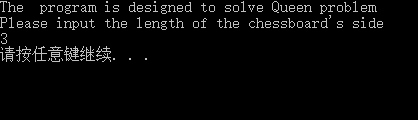


图3 当n=3时的输出

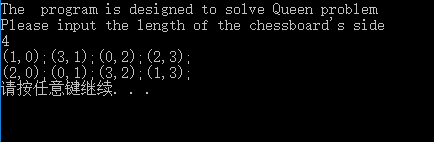


图4 当n=4时的输出

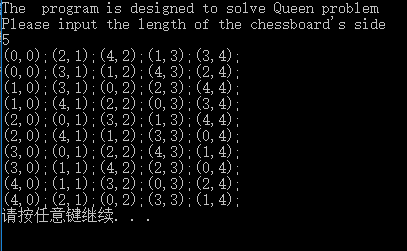


图5 当n=5时的输出

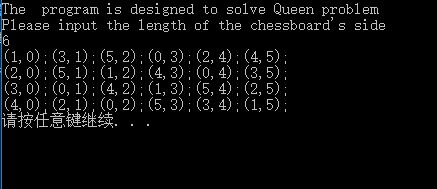


图6 当n=6时的输出

其中：

1输入的目的是为了检测只有一个棋子的特殊情况。

5输入的目的是为了检测当棋盘右下角在路径中时能否正常的输出和终止循环。

所有的输出均符合预期输出的坐标结果，均通过。

当n逐渐增大时，程序运行所耗费的时间将会急剧上升，结果也很多，这里就不一一列出了。

**四、分析与探讨**

程序在n较小时可以正常输出，但是当n过大时，就不能在短时间内输出所有结果

由于本例源代码中使用的循环个数过多，所以当n增大时，将根本无法计算其时间复杂度。所以这里，非常遗憾，不能进行探讨。

而对于空间复杂度，本例的源代码中只声明了有限个变量，而且充分借助了储存组合的栈所记录的坐标，因此为O(1);

**附录：源代码**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define Maxsize 20

typedef struct {

int x;

int y;

}chess;//规定记录棋子的坐标方式

typedef struct {

chess c[Maxsize];

int top;

}stack;//储存棋子组合的栈

void printpath(stack \*&s, int n)//打印组合

{

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

printf("(%d,%d);", s->c[i].x, s->c[i].y);

printf("\n");

}

void Push(stack \*&s, int a, int b)//入栈操作

{

s->top++;

s->c[s->top].x = a;

s->c[s->top].y = b;

}

void scangrid(stack \*&s, int n)

{

int a = 0;

int b = 0;

int i;

while (!((s->top == -1) && (a == n)&&(b==0)))//判断是否已经扫描完了所有的情况

{

for (a = 0; a < n; a++)

{

for (i = 0; i <= s->top; i++)//判断当前的格子是否能够入栈

if (a == s->c[i].x || abs(s->c[i].x - a) == abs(s->c[i].y - b) || s->top == n - 1)//条件：不同对角线且不同列

break;//若冲突，则中止循环

if (i > s->top)//如果i比栈顶指针大，说明可以入栈

{

Push(s, a, b);//入栈

b++;//将当前位置推向下一行

if (s->top == n - 1)//如果栈满

{

printpath(s, n);//打印组合

while (a == n - 1&&b!=0)//当达到了最右端的格子（不包含第一行的情况）

{

a = s->c[s->top].x;

b = s->c[s->top].y;//将当前格子置为栈顶格子

s->top--;//出栈

}

continue;//继续向右扫描

}

break;//跳出该行，扫描下一行

}

else//如果不能入栈

{

while (a == n - 1&&b!=0)//当达到了最右端的格子（不包含第一行的情况）

{

a = s->c[s->top].x;

b = s->c[s->top].y;//将当前格子置为栈顶格子

s->top--;//出栈

}

}

}

}

}

int main()

{

int n;

stack \*s = (stack \*)malloc(sizeof(stack));

s->top = -1;

printf("The program is designed to solve Queen problem\n");

printf("Please input the length of the chessboard's side(larger than zero)\n");

do{

scanf\_s("%d", &n);

}while(n<1);

scangrid(s, n);

system("PAUSE");

return 0;

}