|  |  |
| --- | --- |
| **n皇后问题**：在n\*n的棋盘上放置彼此不受攻击的n个皇后。（皇后可以攻击在同一行、同一列、同一斜线上的棋子）  按以上规则：同一行或同一列或同一斜线上只能有一个皇后，同一行或同一列上必须有一个皇后；  n皇后问题的解空间是一棵n叉树，树的深度为n | 1 2 3 4 5 6 7 8  1 . . . . . . . 8  2 . . . 4 . . . .  3 1 . . . . . . .  4 . . 3 . . . . .  5 . . . . . 6 . .  6 . 2 . . . . . .  7 . . . . . . 7 .  8 . . . . 5 . . . |

**回溯法**

1 定义合适的解空间，包括解的组织形式和显约束;

2 确定解空间的组织结构（通常用树静态，如子集树、排列树、m叉树）;

3 搜索解空间：按深度优先搜索策略，根据隐约束（约束函数和限界函数），在解空间中搜索问题的可行解；当发现当前结点不满足求解条件时，就回溯，尝试其他路径；

#include<stdlib.h> // 回溯法（使用栈）

#include<stdio.h>

#include<memory.h>

#define true 1

#define false 0

int n=0; // 解的个数

void printResult(int[]);

void movequeen(int,int,int[],int[],int[]);

void eightqueen( );

void main( )

{

eightqueen( ); // 调用求解八皇后问题

system("pause");

}

void printResult2(int s[] )

{

int arr[9][9]={0};

for(int p=1;p<=8;p++)

{

arr[p][s[p]] = 1;

}

for(int m=1;m<=8;m++)

{

for(int n=1; n<=8; n++)

printf("%2d",arr[m][n]);

printf("\n");

}

printf("\n");

}

void eightqueen( ) // 求解八皇后问题

{

int i,j; // 当前行、列

int s[9]; // 顺序栈s[i]：若皇后放在位置(i,j)上，则将j压入栈内，即s[i] = j

// 检查皇后之间是否冲突的三个数组

int a[9]; // a[j]为真时，表示第j列上无皇后

int b[17]; // "/"方向上，i+j相等，下标为2至16，b[k]为真时表示"/"方向上无皇后

int c[17]; // "\"方向上，i-j相等，下标为-7至7，c[k]为真时表示"\"方向上无皇后

// c[i-j+9]表示下标在2-16范围内

for(i=2; i<=16; i++) { // 初始化，各位置全无皇后

if(i>=2 && i<=9)

a[i-1]=true;

b[i]=true;

c[i]=true;

}

i=1; j=1;

while(i>=1) { // 当i=0时终止循环

while(j<=8) { // 在当前行i上寻找安全位置；

if(a[j] && b[i+j] && c[i-j+9]) break;

j++;

}

if(j<=8) { // 找到安全位置(i,j)

a[j] = false; // 为假时，表示第j列上有皇后

b[i+j] = false;

c[i-j+9] = false;

s[i]=j; // 皇后位置j入栈

if(i==8) { // 找到一个解,输出解

printf("第%d个解：\n",++n);

printResult(s); // 打印输出一个解

movequeen(i,j,a,b,c); // 移去位置(i,j)上的皇后

i--; j=s[i]; // 退栈，回溯到上一个皇后

movequeen(i,j,a,b,c); // 移去位置(i,j)上的皇后

j++; // 修改栈顶皇后的位置

}

else {

i++;j=1; // 准备放置下一个皇后

}

}

else {

i--; // 退栈

if(i>=1) { // 判断一下是否越界

j=s[i];

movequeen(i,j,a,b,c); // 移去皇后

j++;

}

} // if(j<=8)

} // while(i>=1)

}

// 移去位置(i,j)上的皇后函数

void movequeen(int i, int j, int a[], int b[], int c[])

{

a[j] = 1; // j列上无皇后

b[i+j] = 1; // "/"方向上无皇后

c[i-j+9] = 1; // "\"方向上无皇后

}

void printResult(int s[] ) // 打印输出一个解的函数

{

int i,j; // 循环变量

printf("%8c",' ');

for(i=1;i<=8;++i)

printf("%4d",i);

for(i=1;i<=8;++i)

{

printf("\n%4c",' ');

for(j=1;j<=8;++j)

{

if(j==1)

printf("%4d",i);

if(j==s[i])

printf("%4d",s[i]);

else

printf("%4c",'.');

}

printf("\n");

}

printf("\n\n");

}

void printResult3(int s[] )

{

int k;

printf("行： 1 2 3 4 5 6 7 8\n");

printf("列：");

for(k=1; k<=8; k++)

printf("%2d",s[k]);

printf("\n");

}

第1个解：

行： 1 2 3 4 5 6 7 8

列： 1 5 8 6 3 7 2 4

第2个解：

行： 1 2 3 4 5 6 7 8

列： 1 6 8 3 7 4 2 5

#include <iostream> // 回溯法（递归）

#include <cmath> // 求绝对值函数需要引入该头文件

#define M 105

using namespace std;

int n; // n表示n个皇后

int x[M]; // x[i]表示第i个皇后放置在第i行第x[i]列，表示解空间

int countn; // countn表示n皇后问题可行解的个数

bool Place(int t) // 判断第t个皇后(在第t行)能否放置在第j(j<t)个位置

{

bool ok=true;

for(int j=1;j<t;j++) // 判断该位置的皇后是否与前面t-1个已经放置的皇后冲突

{

if(x[t]==x[j]||t-j==fabs(x[t]-x[j]))// 判断列、对角线是否冲突

{

ok=false;

break;

}

}

return ok;

}

void Backtrack(int t) // t表示当前扩展结点在第t层

{

if(t>n) // 如果当前位置为n,则表示已经找到了问题的一个解

{

countn++;

for(int i=1; i<=n;i++) // 打印选择的路径

cout<<x[i]<<" ";

cout<<endl;

cout<<"----------"<<endl;

}

else

for(int i=1;i<=n;i++) // 分别判断n个分支,特别注意i不要定义为全局变量,否则递归调用有问题

{

x[t]=i;

if(Place(t)) // 如果不冲突的话进行下一行的搜索，否则考察下一个分支（兄弟结点）

Backtrack(t+1);

}

}

void main()

{

cout<<"请输入皇后的个数 n:";

cin>>n;

countn=0;

Backtrack(1);

cout <<"答案的个数是："<<countn<< endl;

}