四皇后问题分析

17硬件一班 汪昊程 171491205

## 引言

### 什么是四皇后问题？N皇后问题？

### 四皇后问题不同情况下的状态空间和解空间？

### 四皇后问题搜索树的分析举例？

经过调查分析，我得到以下结论：

1.所谓4皇后问题就是求解如何在4×4的棋盘上无冲突的摆放4个皇后棋子。在国际象棋中，皇后的移动方式为横竖交叉的，因此在任意一个皇后所在位置的水平、竖直、以及45度斜线上都不能出现皇后的棋子，N皇后以此类推。

2.四皇后问题不同情况下的状态空间和解空间情况如下：

1. 当不考虑棋子的摆放位置时：

状态空间：C16/0+C16/1+C16/2+C16/3+C16/4 ==47297

解空间：16\*15\*14\*13==43680

1. 当考虑以列为核心棋子的摆放位置时：

状态空间：4+4\*4+4\*4\*4+4\*4\*4\*4==341

解空间：4\*4\*4\*4==256

1. 当以四皇后规则为核心考虑棋子的摆放位置时：

状态空间：无皇后1种，一皇后4种，二皇后6种，三皇后4种，四皇后2种，共17种。

解空间：2种

## 正文部分

得到了四皇后四种情况下的搜索空间和解空间树，我们如何实现？

首先，我们先用算法考虑棋子摆放的位置是否合理。

判断一个棋子是否合理，需要判断即将落下的棋子的左，右，上，下，左前，右前，左后，右后，是否有棋子？

具体实现代码如下：

**# include <stdio.h>**

**# define N 4**

**int** count=0; *// 判断 Q[i][j] 是否存在冲突*

**int** **isCorrect**(**int** i, **int** j, **int** (\*Q)[N])

{

**int** s,t; *// 判断行*

**for**(s=i,t=0; t<N; t++) **if**(Q[s][t]==1 && t!=j)

**return** 0; *// 判断列*

**for**(s=0,t=j; s<N; s++) **if**(Q[s][t]==1 && s!=i)

**return** 0; *// 判断左上角*

**for**(s=i-1,t=j-1; s>=0&&t>=0; s--,t--) **if**(Q[s][t]==1)

**return** 0; *// 右下角*

**for**(s=i+1,t=j+1; s<N&&t<N; s++,t++) **if**(Q[s][t]==1)

**return** 0; *// 右上角*

**for**(s=i-1,t=j+1; s>=0&&t<N; s--,t++) **if**(Q[s][t]==1)

**return** 0; *// 左下角*

**for**(s=i+1,t=j-1; s<N&&t>=0; s++,t--) **if**(Q[s][t]==1)

**return** 0;

**return** 1;

}

此函数判断棋子落下的位置是否合理。

接下来，我们用递归实现。

**void** **Queue**(**int** i, **int** (\*Q)[N]) {

**int** j,k; *// 第四行已经遍历完，打印符合条件的结果，结束*

**if**(i==N) { count++;

printf("No.%d slove way\n",count);

**for**(k=0;k<N; k++) {

**for**(j=0; j<N; j++)

printf("%-5d", Q[k][j]);

printf("\n"); *//return;* }

printf("\n"); } *// 遍历第 i+1 行的 j+1 列*

**for**(j=0;j<N; j++)

{ **if**(isCorrect(i,j,Q))

{ Q[i][j]=1;

Queue(i+1,Q);*// 遍历下一行*

Q[i][j]=0; *// 如果下一行不成立，则回溯，重置 0* }

}

}

在实现算法的过程中，我有如下收获：

1. 整个算法是用回溯法实现
2. 采用深度优先遍历
3. 虽然看上去是用树，但由于四皇后问题的特殊性，我们采用二维数组方式解决。

如此，再加上网上文献的帮助，我得到的深度优先遍历算法结论如下：

1. 完整性：不能保证算法完整性。一旦深度无限长，或者有循环，计算机就会一直在一条路上一直遍历，无法回溯，导致问题无法解决
2. 最优性：不能保证最优性。完整性无法保证的话，最优性也无法保证
3. 时间复杂度：O（b的m次幂）时间复杂度偏高（如果m很大的话）
4. 空间复杂度：O（bm）线性空间回溯。

通过本人实践，发现此问题用深度优先算法空间复杂度低，虽然算法完整性和最优性都无法保证，但是解决某些某些问题问题有奇效（此问题如果用广度优先的话，空间复杂度极高）