人工智能练习一

班级：硬件一班

姓名：于华宇

学号：171491120

## 四皇后问题分析

1. 初始状态：4×4棋盘上没有皇后
2. 动作：向棋牌上没有棋子的位置放置皇后
3. 转移函数：向某个位置放置皇后，若会产生攻击则回溯选择另一个位置，直到不产生攻击为止。
4. 目标：棋盘上放置4个皇后，且彼此不产生攻击。
5. 花费：试验次数

### 四皇后问题实现措施

1. 在4×4方格上任意放4个皇后

状态空间：C160+C161+C162+C163+C164=47297

解空间：16×15×14×13=43680

2、4×4方格上，每一列放1个皇后

状态空间：1+4+4×4+4×4×4+4×4×4×4=341

解空间：4×4×4×4=256

3、4×4方格上，每一列放1个皇后，且互不攻击

一列一列放置皇后，若出现攻击则回溯到上一列找下一个位置

状态空间：

无皇后时1种

1个皇后时4种

2个皇后时2+1+1+2=6种

3个皇后时1+1+1+1=4种

4个皇后时：2种

状态空间有1+4+6+4+2=17种

解空间：2种

|  |  |  |  |
| --- | --- | --- | --- |
|  | √ |  |  |
|  |  |  | √ |
| √ |  |  |  |
|  |  | √ |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | √ |  |
| √ |  |  |  |
|  |  |  | √ |
|  | √ |  |  |

### 四皇后实现代码

思路：一列一列向4×4棋盘上放皇后。若产生攻击，则放置在同一列下一行，若整列都无法找到合适的位置则回溯（列数减一，上一行放皇后的位置下移一行），直到找到合适位置，count++，并输出解。

回溯利用递归实现。当满足解，会一直调用函数到最后，若不满足解，则弹栈回到上一层。

**#include<iostream>**

**using namespace std;**

**#define ATTACK 0**

**#define OK 1**

**#define N 4**

**int count=0; //记录符合要求解的个数**

**int space[N][N]; //构造4×4棋盘空棋盘**

**/\*int isAttacked(int row,int column,int (\*space)[N])**

**{**

**int i,j;**

**for(i=0;i<column;i++)**

**//判断同一行是否有皇后已经放置**

**if(space[row][i]==1)**

**return ATTACK;**

**for(i=1;row-i>=0,column-i>=0;i++)**

**//判断是否在同一斜线**

**if(space[row-i][column-j]==1||space[row+i][column-j]==1)**

**return ATTACK;**

**return OK;**

**}\*/**

**int isAttacked(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;**

**}**

**void backTrack(int column,int (\*space)[N]){**

**int i,j;**

**if(column==N){**

**count++;**

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

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

**cout<<space[i][j]<<" ";**

**cout<<endl;**

**}**

**cout<<endl;**

**}**

**else{**

**for(int row=0;row<N;row++){**

**if(isAttacked(row,column,space)){**

**space[row][column]=1;**

**backTrack(column+1,space);**

**space[row][column]=0;**

**}**

**}**

**}**

**}**

**int main(){**

**for(int m=0;m<N;m++){**

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

**space[m][n]=0;**

**}**

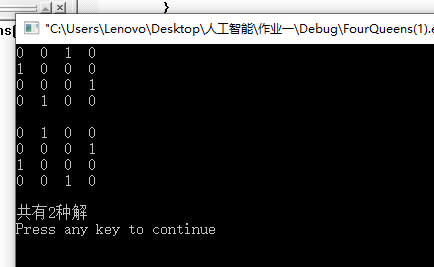
**}**

**backTrack(0,space);**

**cout<<"共有"<<count<<"种解"<<endl;**

**return 0;**

**}**



参考资料：<https://www.cnblogs.com/pinsily/p/7587178.html>

其中注释掉的第一个判断是否攻击函数是在查阅参考资料之前写的。我认为放置皇后时是一列一列进行放置的，不需要检查还未放置位置的攻击情况。但是运行时总是有问题。

经过对正确代码的更改，发现。若不检测列，右上角，右下角，结果依然正确。若不检测行，左上角，左下角则结果错误。证明参考资料中的isAttacked（）函数可以精简。所以我之前写的isAttacked（）函数在想法和逻辑上没有错误，但在具体实现上出了问题。

经过比对，行判断没有问题。斜线判断出现了错误。

错误代码：  
 **for(i=1;row-i>=0,column-i>=0;i++)**

**//判断是否在同一斜线**

**if(space[row-i][column-j]==1||space[row+i][column-j]==1)**

**return ATTACK;**

改进如下：

**for(i=1,j=1;row-i>=0&&column-j>=0;i++,j++)**

**//判断是否在同一斜线**

**if(space[row-i][column-j]==1)**

**return ATTACK;**

**for(i=1,j=1;row+i<N&&column-j>=0;i++,j++)**

**//判断是否在同一斜线**

**if(space[row+i][column-j]==1)**

**return ATTACK;**

错误原因：row+i 与 row-i的判断条件需要分开。所以判断斜线上的时候需要分开判断。i，j变量的增减变化也要注意。判断条件的逻辑也要注意。

### 深度优先遍历

1. 完整性：不能保证完整性。若路径产生循环，或者为无限长，会一直在某一条路径中遍历，无法回溯到其他分支。无法保证可以找到问题的解。
2. 最优性：由于无法保证找到问题的解。同样，也无法保证能找到解决问题花费最少的解。（完整性不能保证，则最优性也无法保证）
3. 时间复杂度：O(bm)，由于每一条路径都要走到底，所以当解的度数比整棵树的度数多很多的时候，时间上会比较浪费。
4. 空间复杂度：O(bm)，回溯的时候，只需要记住当前路径上节点的子节点即可，空间是线性空间。如果是二叉树，则空间复杂度为O(m)

总结：深度优先搜索由于完整性最优性都不具备，很难应用在实践中，但由于空间复杂度很优秀，所以可以在其他算法中借鉴深度优先搜索的思想，节省空间。