**8-皇后问题**：

初始状态：整个棋盘无皇后

操作：在棋盘上的空位置任意添加一个皇后

过度模型：返回指定方格

目标测试：棋盘上有八个皇后 且未受到人身攻击

路径成本：实验次数

状态：棋盘上存在八个皇后

状态空间 ：从棋盘上任选1/2/3/4/5/6/7/8个位置放皇后 约5100000000

解空间：实际上 第一个皇后有64个位置 第二个有63个位置以此类推 故为64\*63\*62\*61\*60\*59\*58\*57

为你要将每一种情况都要将其模拟出来，那么其实就需要建立一个解空间树来进行模拟。从根节点出发，一直往下，看看这个方向行不行的通不行的话就返回，剪枝，如果可以拿就继续深度搜索，一直到能无冲突的摆放四个皇后为止，用到的是递归回溯。

C语言代码实现：

#include<stdio.h>

int count1=0;

int facs(int i,int j,int (\*Q)[4])//这个函数就是判断同一行，同一列行不能放元素，以及四个方向

{

int s,t;

for(s=i,t=0;t<4;t++)

{

if(Q[s][t]==1&&t!=j)//判断行

return 0;

}

for(t=j,s=0;s<4;s++)

{

if(Q[s][t]==1&&s!=i)//判断列

return 0;

}

for(s=i-1,t=j-1;s>=0&&t>=0;s--,t--)

{

if(Q[s][t]==1)//判断左上方

return 0;

}

for(s=i-1,t=j+1;s>=0&&t<4;s--,t++)

{

if(Q[s][t]==1)//判断右上方

return 0;

}

for(s=i+1,t=j+1;s<4&&t<4;s++,t++)

{

if(Q[s][t]==1)//判断右下方

return 0;

}

for(s=i+1,t=j-1;s<4&&t>=0;s++,t--)

{

if(Q[s][t]==1)//判断左下方

return 0;

}

return 1;

}

void queen(int j,int (\*Q)[4])

{

int i,k;

if(j==4)

{

for(i=0;i<4;i++)

{

for(k=0;k<4;k++)

printf("%d ",Q[i][k]);

printf("\n");

}

printf("\n");

count1++;

return;

}

for(i=0;i<4;i++)

{

if(facs(i,j,Q))

{

Q[i][j]=1;

queen(j+1,Q);//由此往下，初始位置为（0,0）

Q[i][j]=0;

}

}

}

int main()

{

int Q[4][4];

int i,j;

for(i=0;i<4;i++)

{

for(j=0;j<4;j++)

Q[i][j]=0;

}

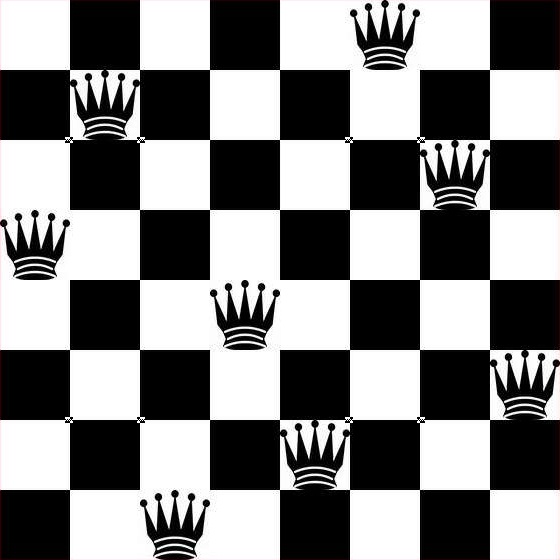
queen(0,Q);

printf("%d",count1);

return 0;

}

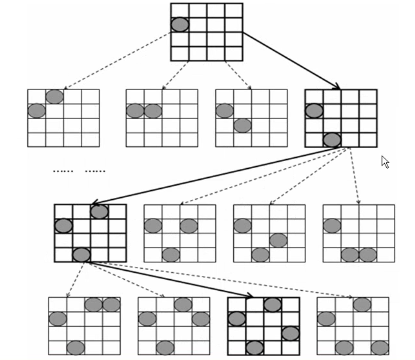
一个解空间：



     以上图为例，当在第i层出现非法的棋盘局面时，就跳回第i-1层，继续探索第i-1层的那个节点的下一个分支；或者在第4层探索到合法的局面就进行记录并跳回上一层，继续探索下一个分支。其他三个解空间树同理。

     以上图为例，就单看探索的第四层节点的个数。使用回溯法，就只需探索第4层中的4个节点，而如果使用穷举法，就要探索玩第4层的所有64个节点，显而易见，哪一个方法更有效。

    其实在解决四皇后问题的时候，并不一定要真的构建出这样的一棵解空间树，它完全可以通过一个递归回溯来模拟。所谓的解空间树只是一个逻辑上的抽象。当然也可以用树结构来真实的创建出一棵解空间树，不过那样会比较浪费空间资源，也没有那个必要



**Completeness** :完整性：如果问题存在解，一定能找到最优解

如果存在 总能找到解决方案吗？

不一定

Optimality：最优性：

使目标函数达到最优值（最大值或最小值）的任一可行解，都称为一个最优解

找得到的解是最优解

总是有一个最优解吗？ 不一定

时间复杂度：耗费的时间

空间复杂度：耗费的内存