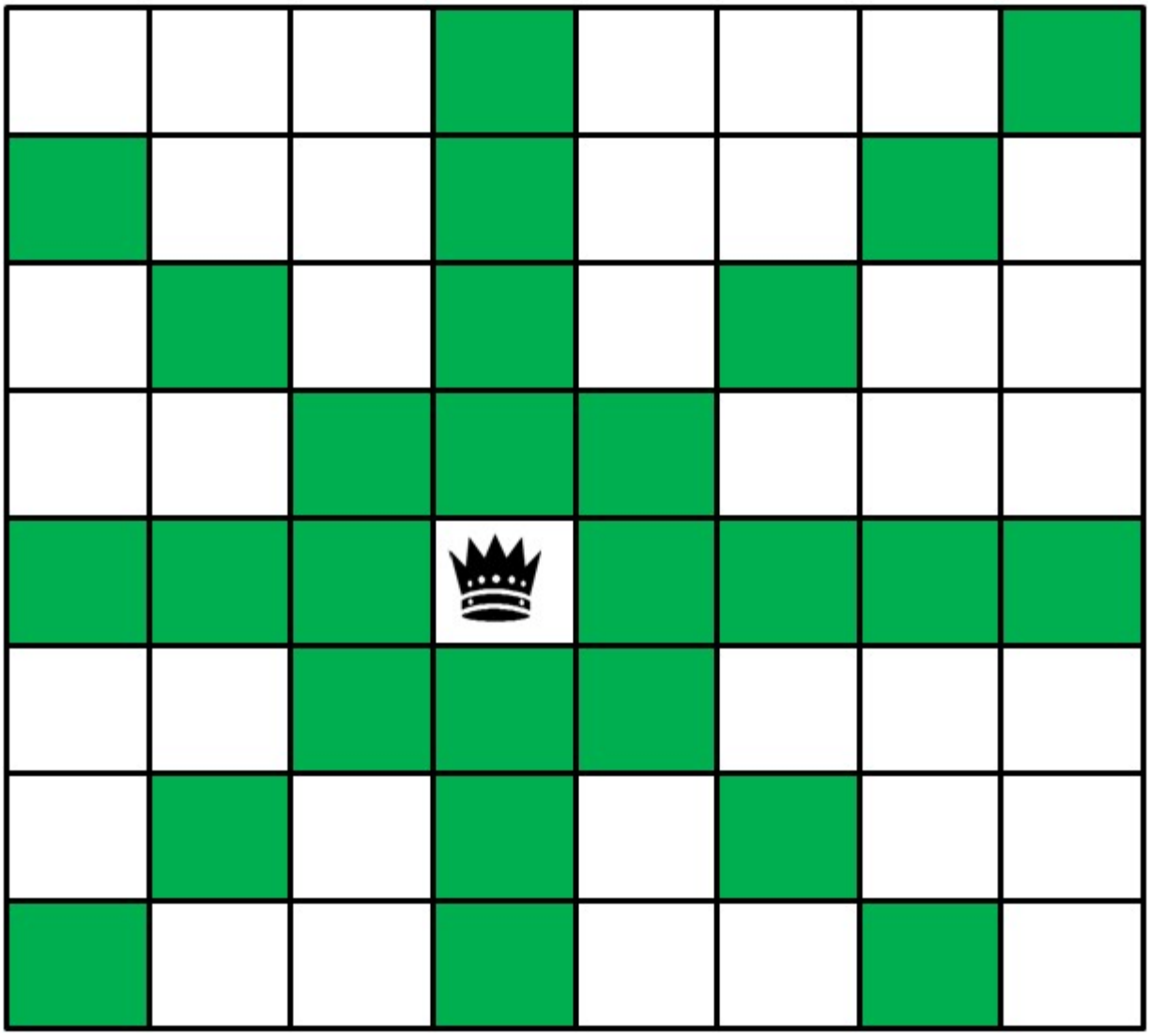
漫画：什么是八皇后问题?

——————— 第二天 ———————

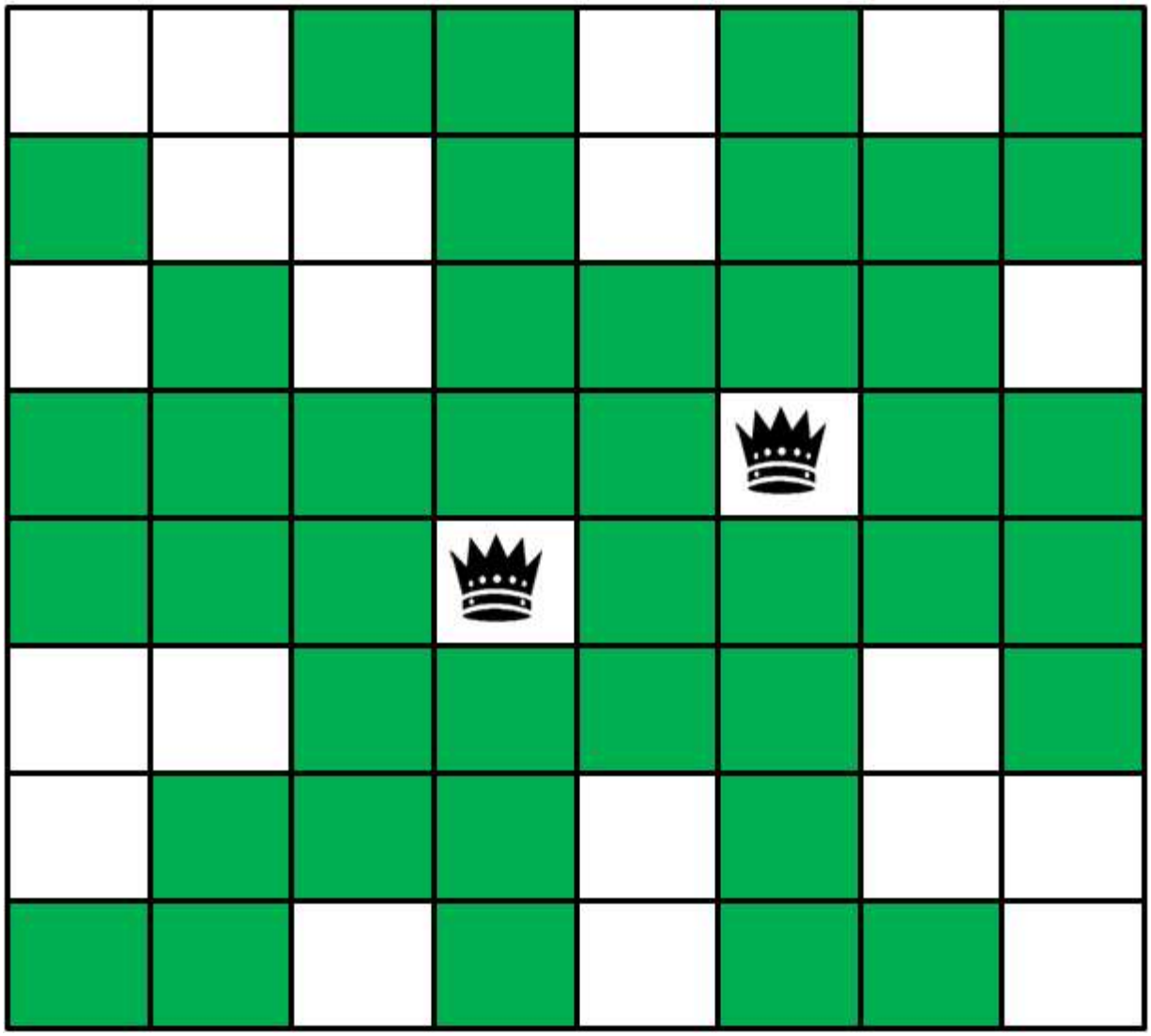
题目是什么意思呢?

国际象棋中的皇后，可以横向、纵向、斜向移动。如何在一个8X8的棋盘上放置8个皇后，使得**任意两个皇后都不在同一条横线、竖线、斜线方向上**?

让我们来举个栗子，下图的绿色格子是一个皇后在棋盘上的“封锁范围”，其他皇后不得放置在这些格子:



下图的绿色格子是两个皇后在棋盘上的“封锁范围”，其他皇后不得放置在这些格子：



那么，如何遵循规则，同时放置这8个皇后呢？让我们来看看小灰的回答。

那个...什么是皇后啊？象棋里不是只有车马炮吗？

呵呵，没关系，回家学下国际象棋吧。

小灰，听说你去面试了？结果怎么样？

哎.....

大黄，你教教我下象棋...啊不，教教我怎样解决八皇后问题呗？

好呀，八皇后问题是一道古老而又著名的问题，甚至早在计算机发明之前就出现了。

什么是八皇后问题?

八皇后问题是一个古老的问题，于1848年由一位国际象棋棋手提出：在8×8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，如何求解?

以高斯为代表的许多数学家先后研究过这个问题。后来，当计算机问世，通过计算机程序的运算可以轻松解出这个问题。

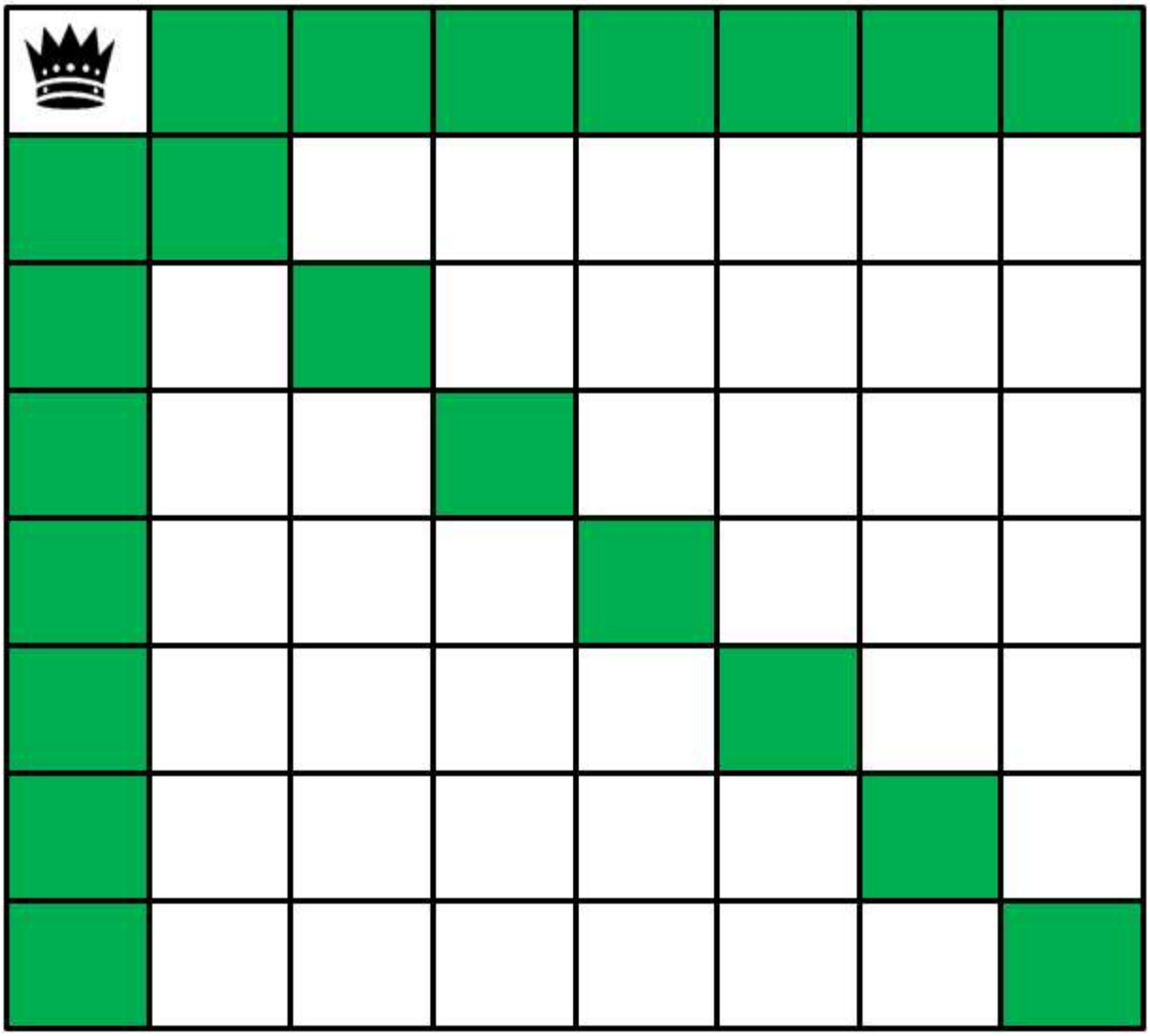
如何解决八皇后问题?

所谓递归回溯，本质上是一种枚举法。这种方法从棋盘的第一行开始尝试摆放第一个皇后，摆放成功后，递归一层，再遵循规则在棋盘第二行来摆放第二个皇后。如果当前位置无法摆放，则向右移动一格再次尝试，如果摆放成功，则继续递归一层，摆放第三个皇后……

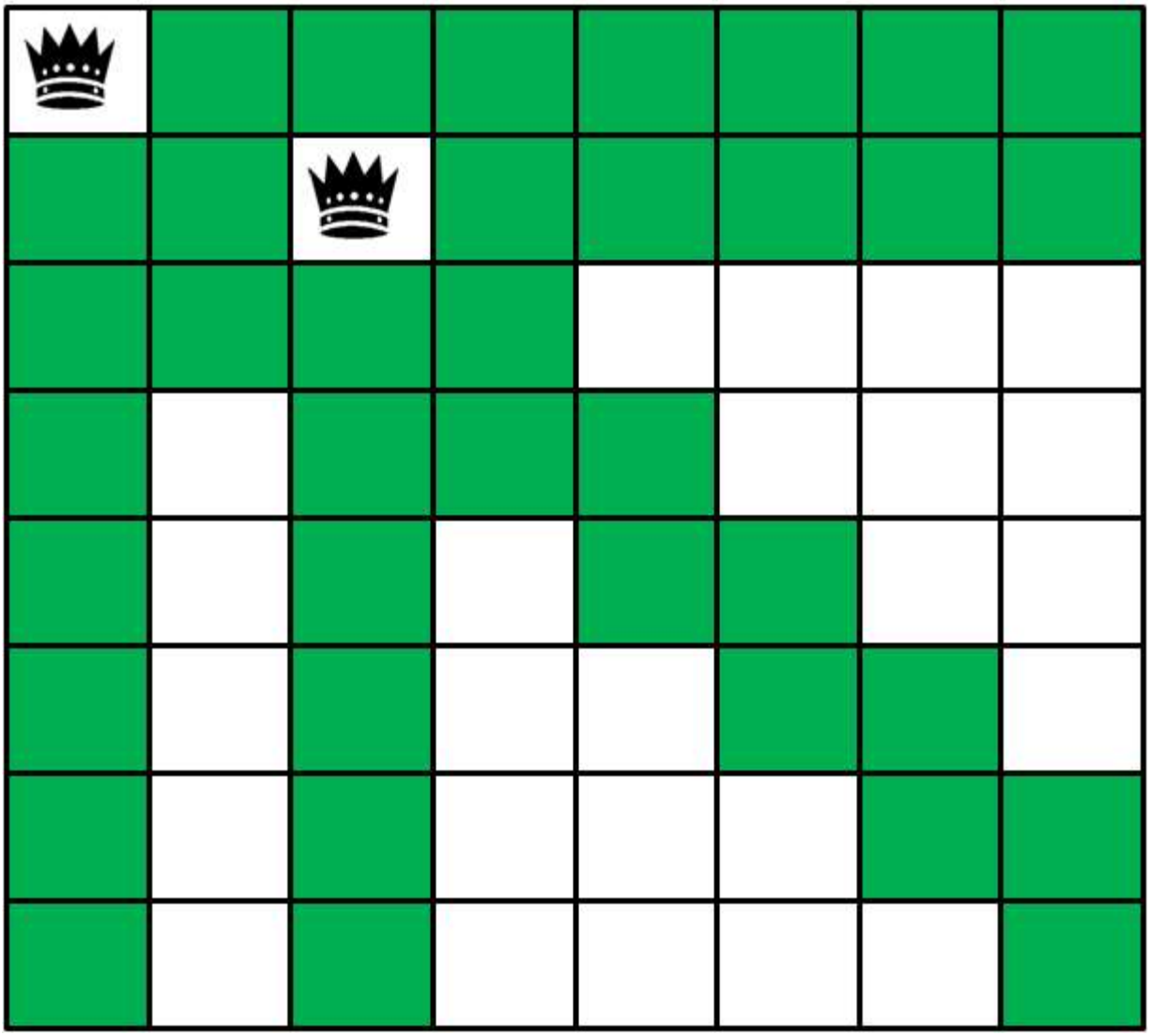
如果某一层看遍了所有格子，都无法成功摆放，则回溯到上一个皇后，让上一个皇后右移一格，再进行递归。如果八个皇后都摆放完毕且符合规则，那么就得到了其中一种正确的解法。

说起来有些抽象，我们来看一看递归回溯的详细过程。

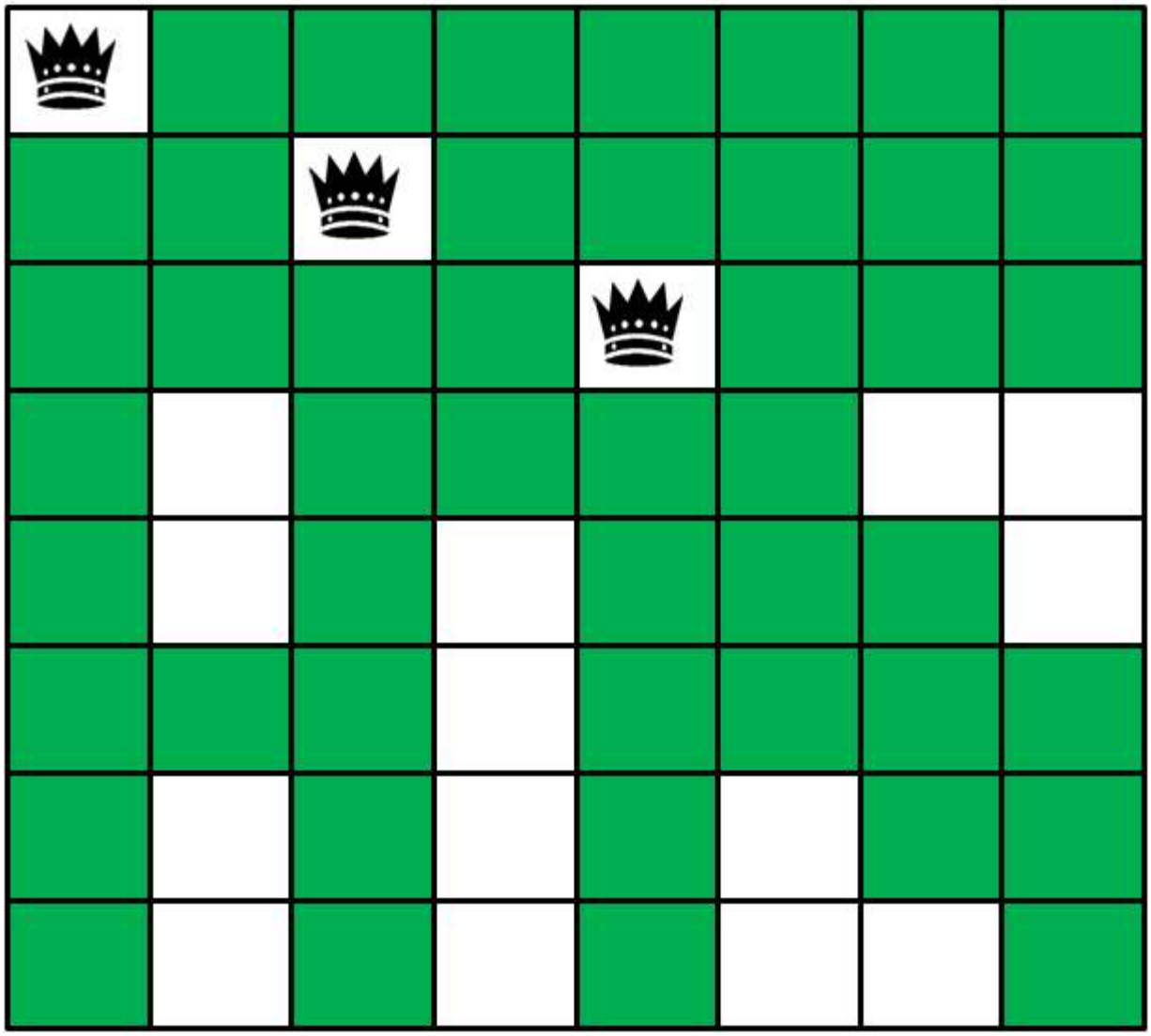
1. 第一层递归，尝试在**第一行**摆放**第一个皇后**：



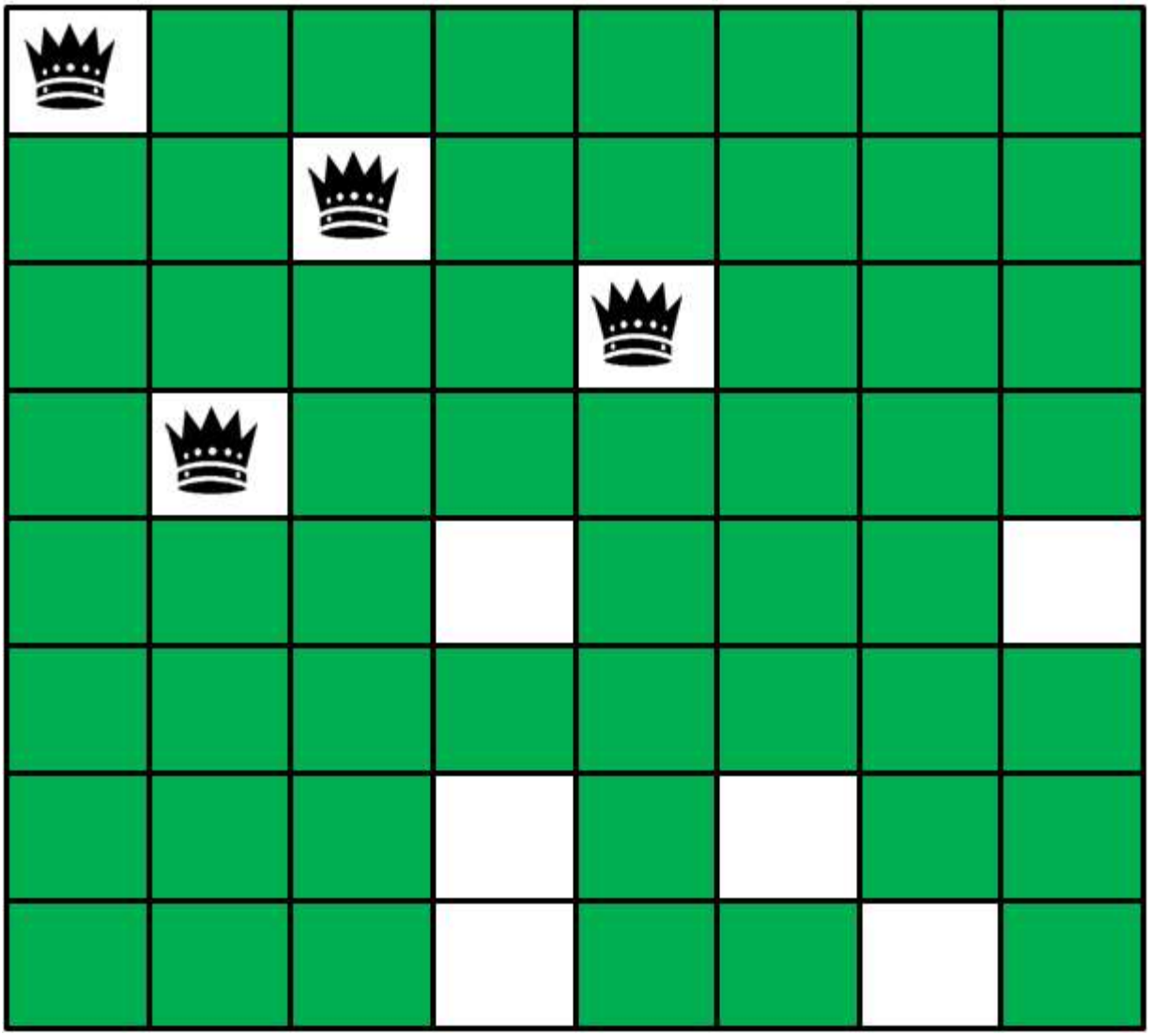
2. 第二层递归，尝试在**第二行**摆放**第二个皇后**（前两格被第一个皇后封锁，只能落在第三格）：



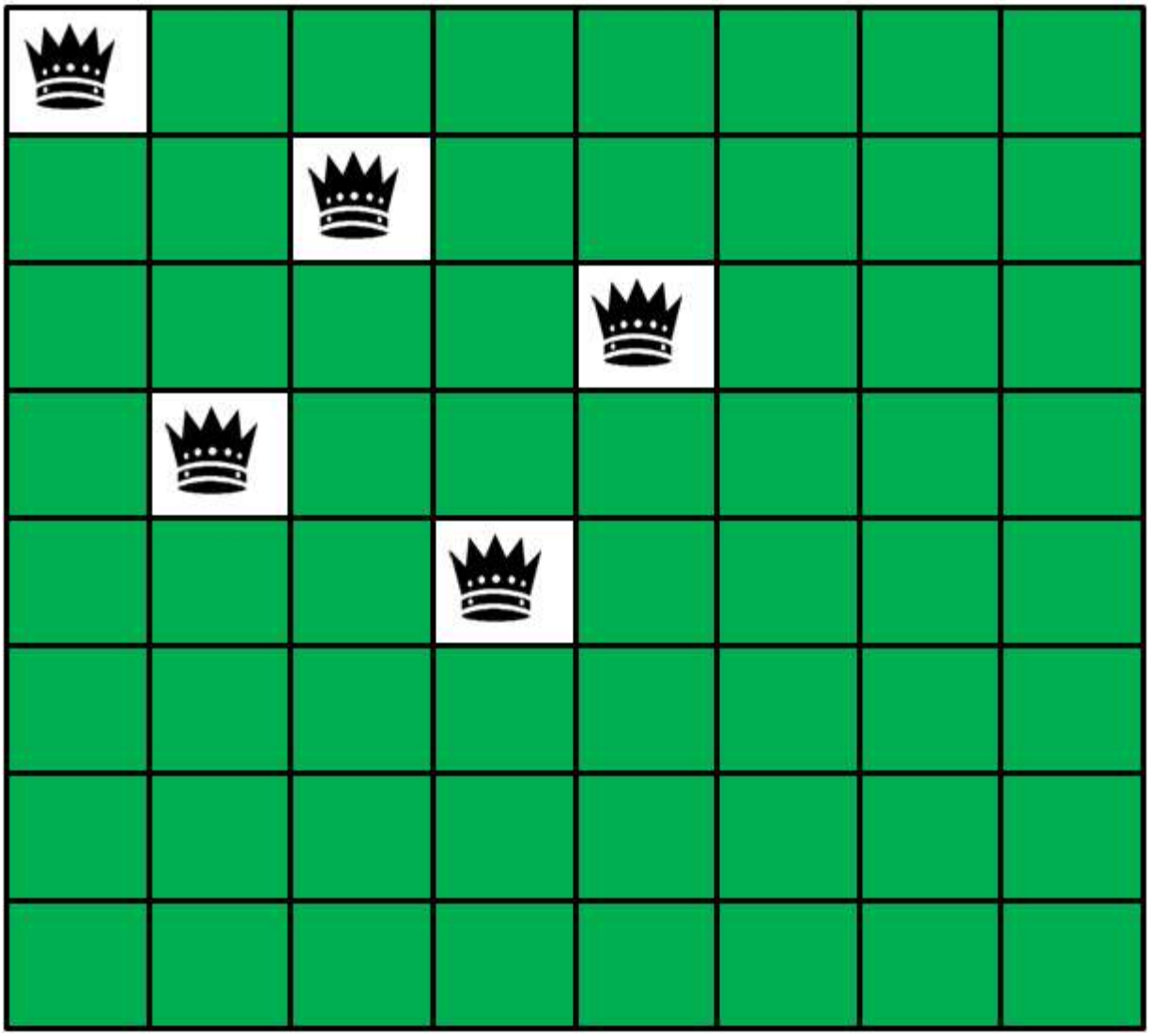
3. 第三层递归，尝试在**第三行**摆放**第三个皇后**（前四格被第一第二个皇后封锁，只能落在第五格）：



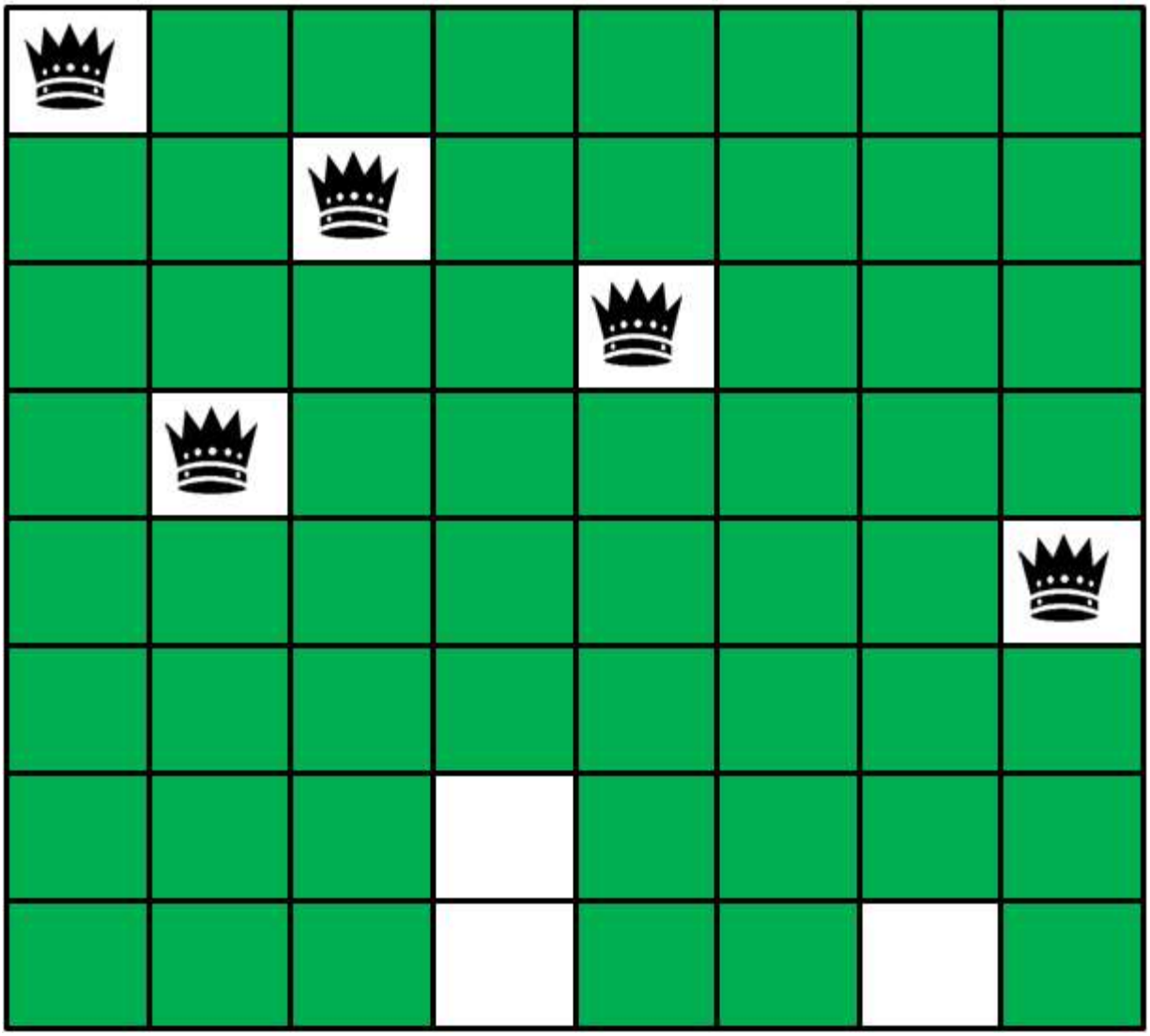
4. 第四层递归，尝试在**第四行**摆放**第四个皇后**（第一格被第二个皇后封锁，只能落在第二格）：



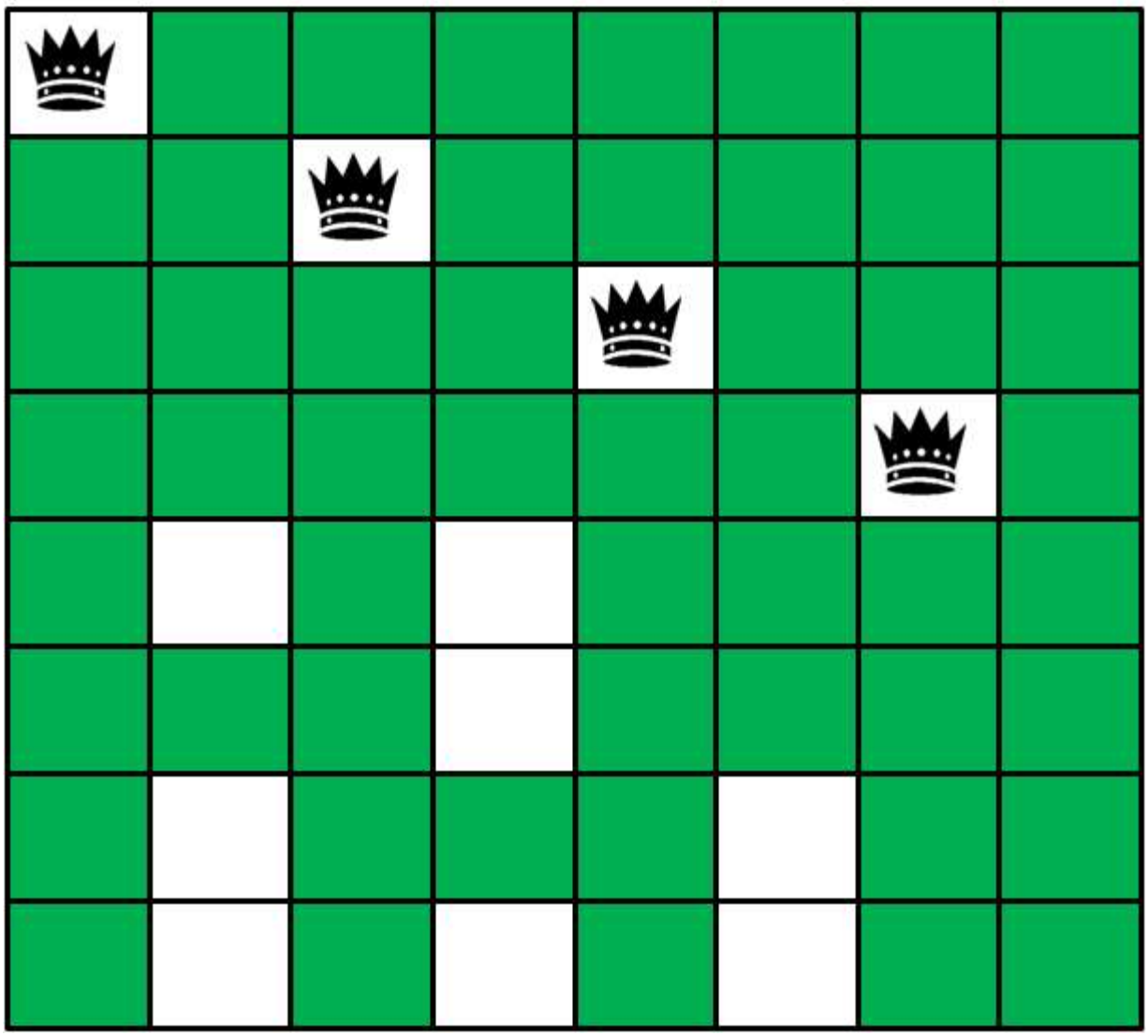
5. 第五层递归，尝试在**第五行**摆放**第五个皇后**（前三格被前面的皇后封锁，只能落在第四格）：



6. 由于所有格子都“绿了”，第六行已经没办法摆放皇后，于是进行回溯，重新摆放**第五个皇后**到**第八格**：



7. 第六行仍然没有办法摆放皇后，第五行也已经尝试遍了，于是回溯到**第四行**，重新摆放**第四个皇后**到**第七格**：



8. 继续摆放第五个皇后，以此类推.....

八皇后问题的代码实现?

解决八皇后问题，可以分为两个层面：

1. 找出第一种正确摆放方式，也就是深度优先遍历。

2. 找出全部的正确摆放方式，也就是广度优先遍历。

由于篇幅优先，我们本篇只介绍如何找出第一种正确摆放方式。

在研究代码实现的时候，我们需要解决几个问题：

1. 国际象棋的棋盘如何表示?

很简单，用一个长度是8的二维数组来表示即可。

```
//棋盘格子的范围，以及皇后数量（应该分开定义，这里偷懒了）
static final int MAX_NUM = 8;
//二维数组作为棋盘
int chessBoard[][] = new int[MAX_NUM][MAX_NUM];
```

由于这里使用的是int数组，int的初始值是0，代表没有落子。当有皇后放置的时候，对应的元素值改为1。

在这里，二维数组的第一个维度代表横坐标，第二个维度代表纵坐标，并且从0开始。比如chessBoard[3][4]代表的是棋盘第四行第五列格子的状态。

2. 如何判断皇后的落点是否合规?

定义一个check方法，传入新皇后的落点，通过纵向和斜向是否存在其他皇后来判断是否合规。

```
//检查落点是否符合规则
boolean check(int x, int y){
    for(int i=0; i<y; i++){
        //检查纵向
        if(chessBoard[x][i] == 1){
            return false;
        }
        //检查左侧斜向
        if(x-1-i >= 0 && chessBoard[x-1-i][y-1-i] == 1){
            return false;
        }
        //检查右侧斜向
        if(x+1+i < MAX_NUM && chessBoard[x+1+i][y-1-i] == 1){
            return false;
        }
    }
    return true;
}
```

3. 如何进行递归回溯?

递归回溯是本算法的核心，代码逻辑有些复杂

```
boolean settleQueen(int y){
    //行数超过8，说明已经找出答案
    if(y == MAX_NUM){
        return true;
    }
    //遍历当前行，逐一格子验证
    for(int i=0; i<MAX_NUM; i++){
        //为当前行清零，以免在回溯的时候出现脏数据
        for(int x=0; x<MAX_NUM; x++){
            chessBoard[x][y] = 0;
        }
        //检查是否符合规则，如果符合，更改元素值并进一步递归
        if(check(i,y)){
            chessBoard[i][y] = 1;
            //递归如果返回true，说明下层已找到解法，无需继续循环
            if(settleQueen(y+1)){
                return true;
            }
        }
    }
    return false;
}
```

4. 如何输出结果?

这个问题很简单，直接遍历二维数组并输出就可以。

```
//打印棋盘当前值
void printChessBoard(){
    for(int j=0; j<MAX_NUM; j++){
        for(int i=0 ; i<MAX_NUM; i++){
            System.out.print(chessBoard[i][j]);
        }
        System.out.println();
    }
}
```

5. 如何把这些方法串起来?

在main函数里分三步来调用：

第一步：初始化

第二步：递归摆放皇后

第三步：最后输出结果。

其中Queen8是整个类的名字。

```
public static void main(String[] args) {
    Queen8 queen8 = new Queen8();
    queen8.settleQueen(0);
    queen8.printChessBoard();
}
```

最终输出如下：

10000000

00001000

00000001

00000100

00100000

00000010

01000000

00010000

这下明白了，八皇后问题的解法还真是有趣！

好了，关于八皇后问题，我们今天就介绍到这里，感谢大家的支持！

由于篇幅原因，这一篇只讲了如何找出第一种正确的八皇后摆放。大家如果有兴趣，可以对文中的代码稍作改动，实现找出所有八皇后摆放的代码。