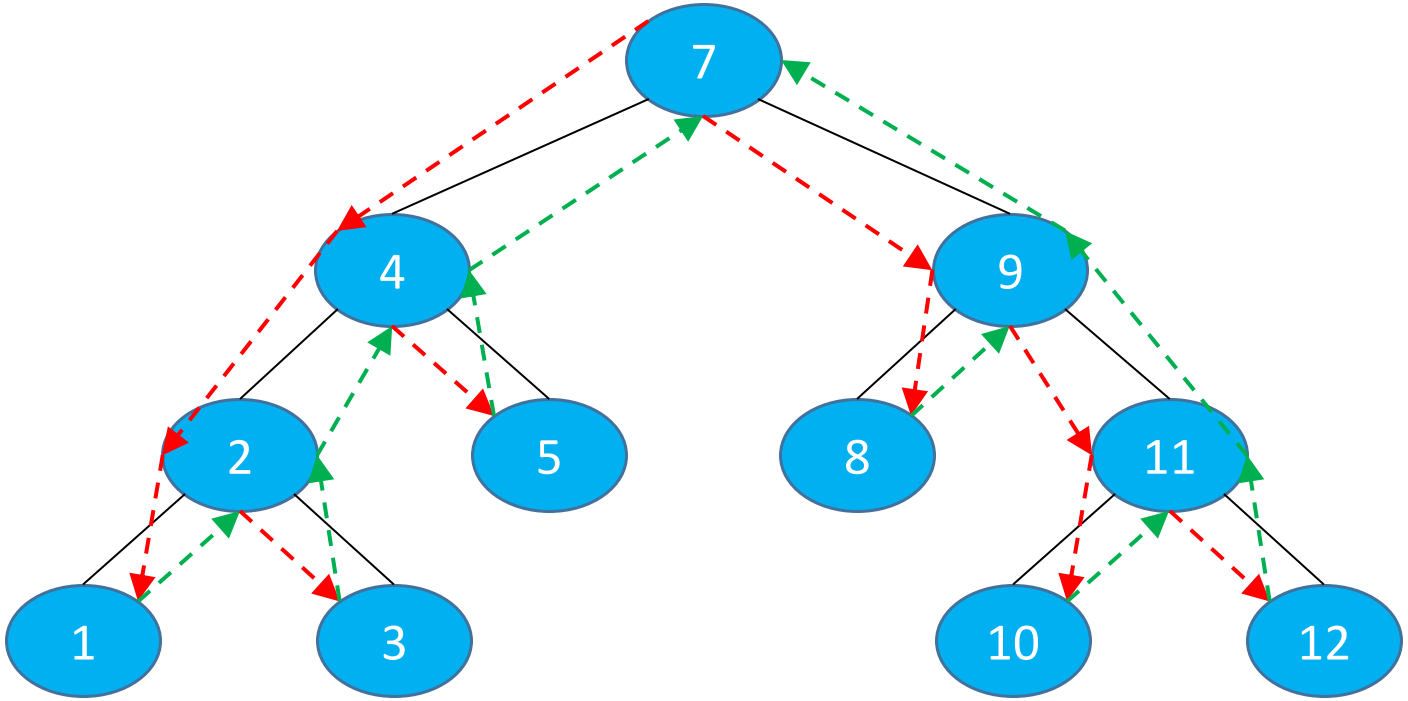
回溯（Back Tracking）

回溯（Back Tracking）

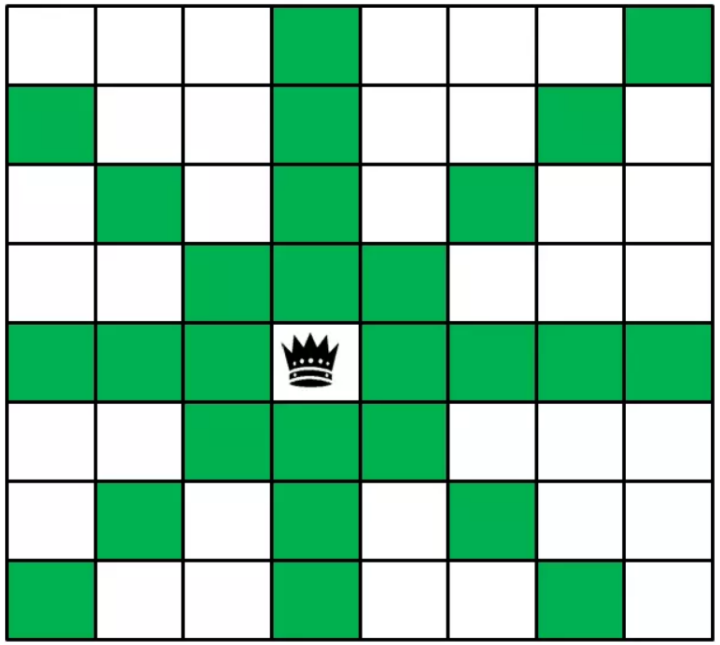
- 回溯可以理解为：通过选择不同的岔路口来通往目的地（找到想要的结果）
  - 每一步都选择一条路出发，能进则进，不能进则退回上一步（回溯），换一条路再试
- 树、图的深度优先搜索（DFS）、八皇后、走迷宫都是典型的回溯应用

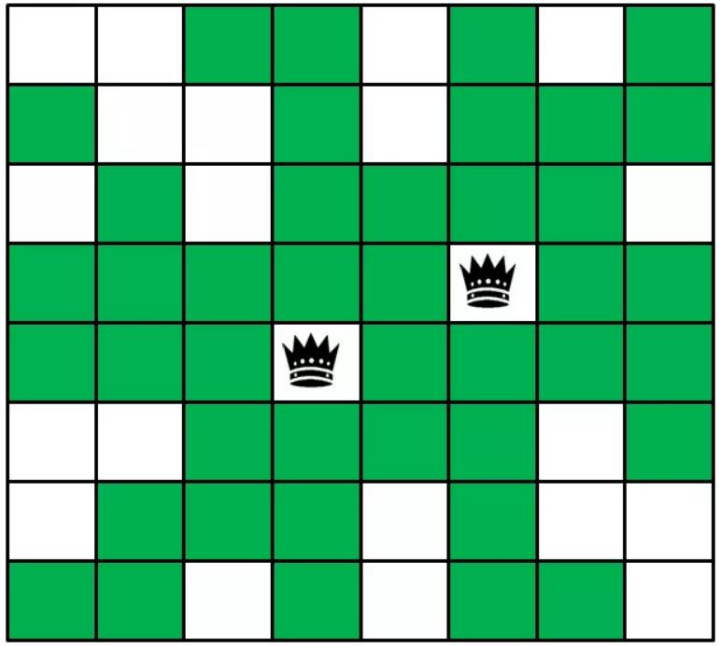


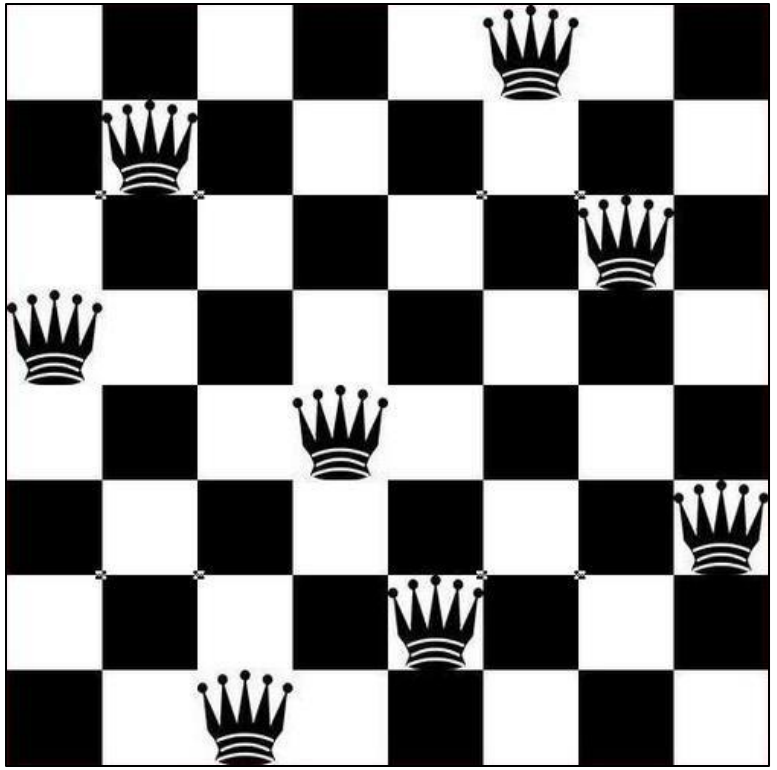
- 不难看出来，回溯很适合使用递归

练习 – 八皇后问题（Eight Queens）

- 八皇后问题是一个古老而著名的问题
- 在8x8格的国际象棋上摆放八个皇后，使其不能互相攻击：任意两个皇后都不能处于同一行、同一列、同一斜线上
- 请问有多少种摆法?







- leetcode_51_N皇后：https://leetcode-cn.com/problems/n-queens/
- leetcode_52_N皇后 II： https://leetcode-cn.com/problems/n-queens-ii/

八皇后问题的解决思路

- 思路一：暴力出奇迹
  - 从 64 个格子中选出任意 8 个格子摆放皇后，检查每一种摆法的可行性
  - 一共 C_{64}^{8} 种摆法（大概是 $4.4 * 10^9$ 种摆法）

$$A_n^m = n(n-1)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

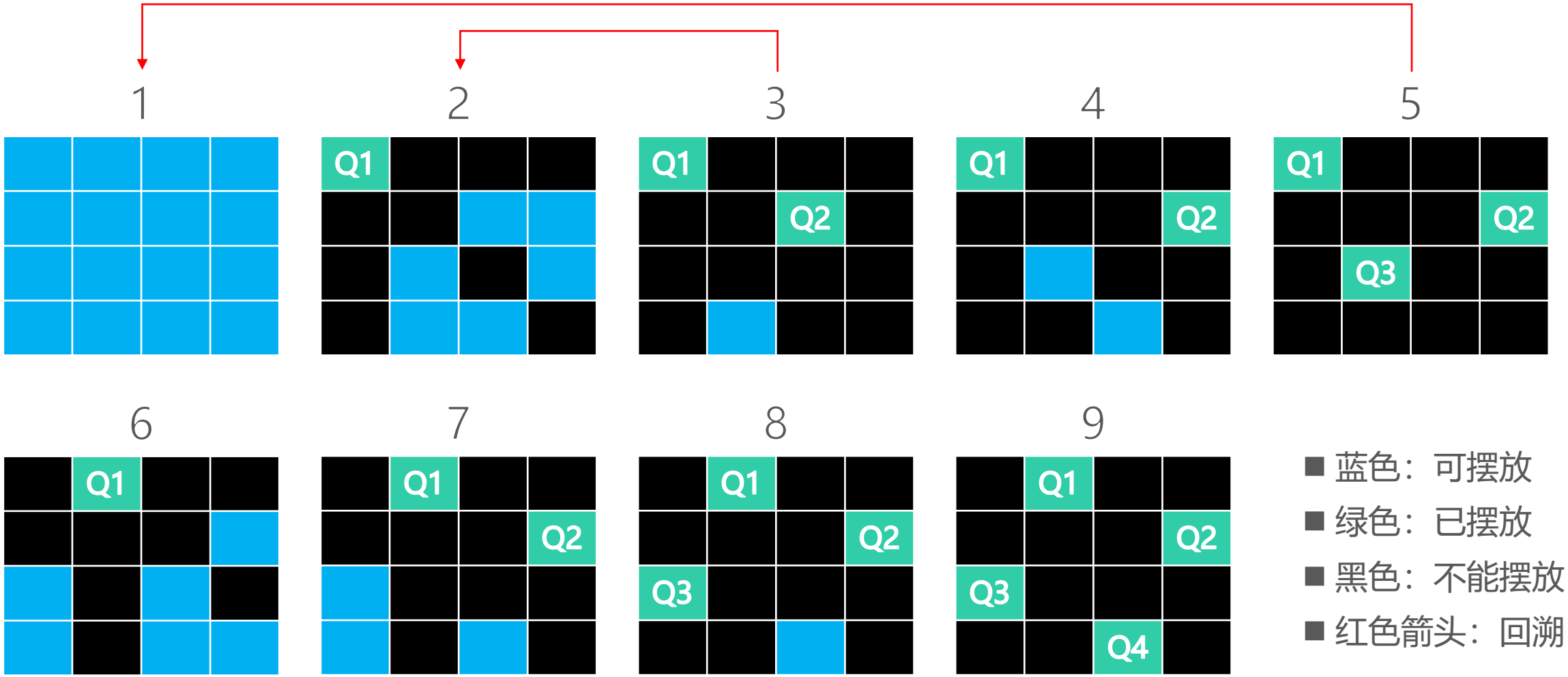
$$C_n^m = \frac{A_n^m}{m!} = \frac{n!}{m!(n-m)!} = C_n^{n-m}$$

- 思路二：根据题意减小暴力程度
  - 很显然，每一行只能放一个皇后，所以共有 8^8 种摆法（16777216 种），检查每一种摆法的可行性

- 思路三：回溯法
  - 回溯 + 剪枝

四皇后 – 回溯法

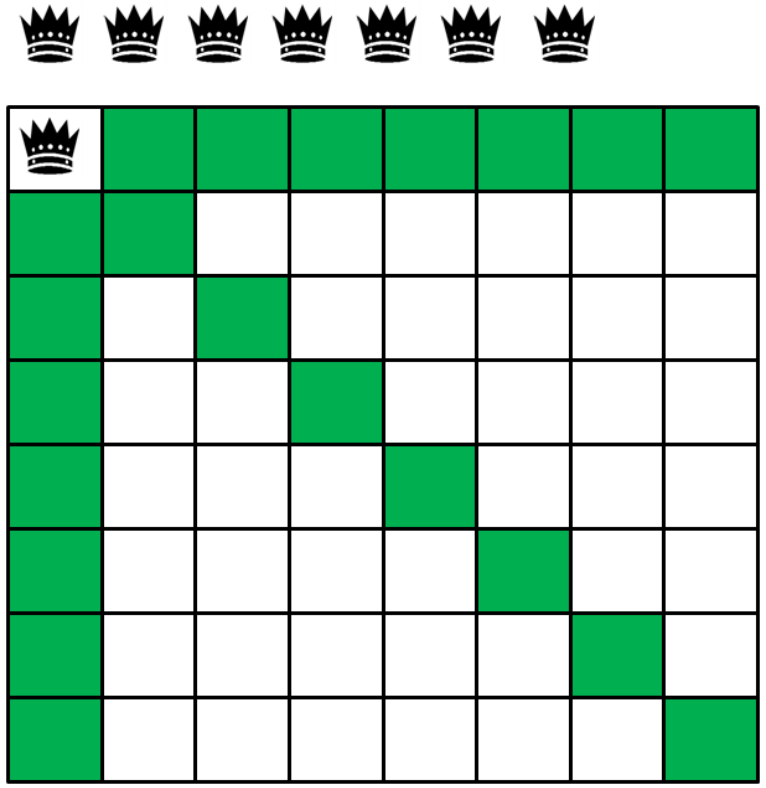
- 在解决八皇后问题之前，可以先缩小数据规模，看看如何解决四皇后问题

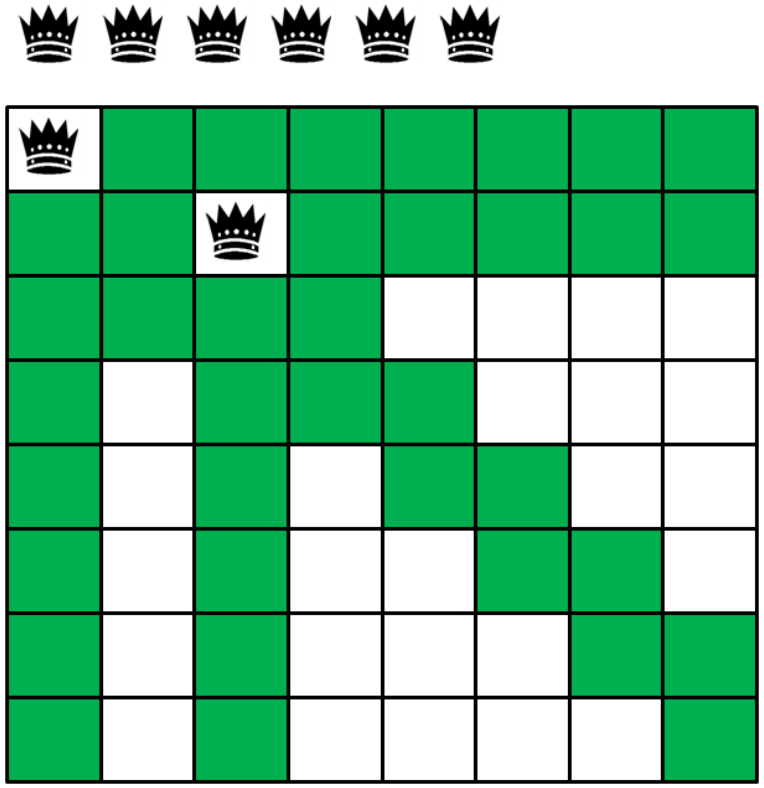


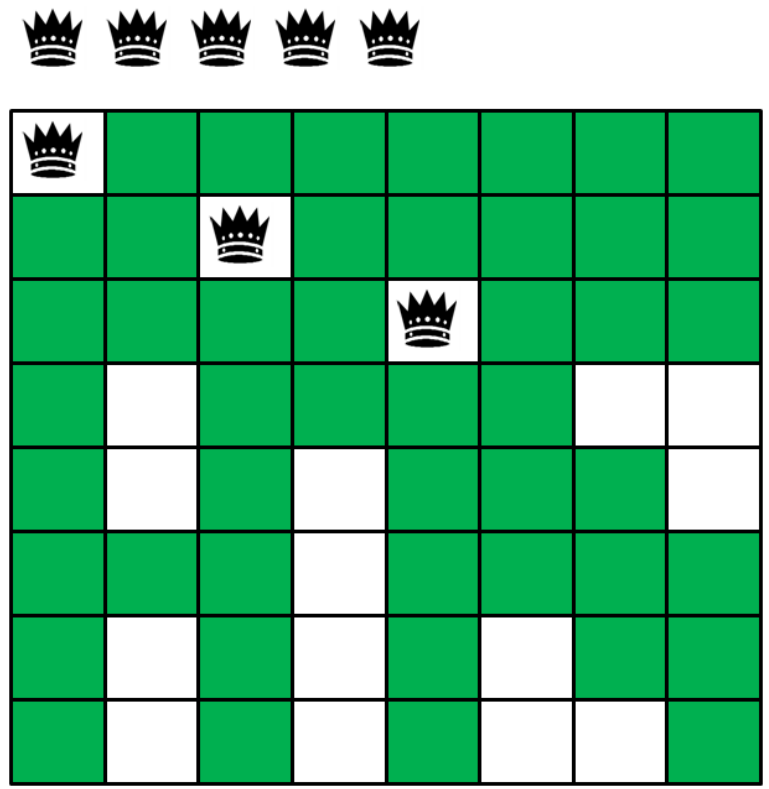
- 蓝色：可摆放
- 绿色：已摆放
- 黑色：不能摆放
- 红色箭头：回溯

四皇后 – 剪枝（Pruning）

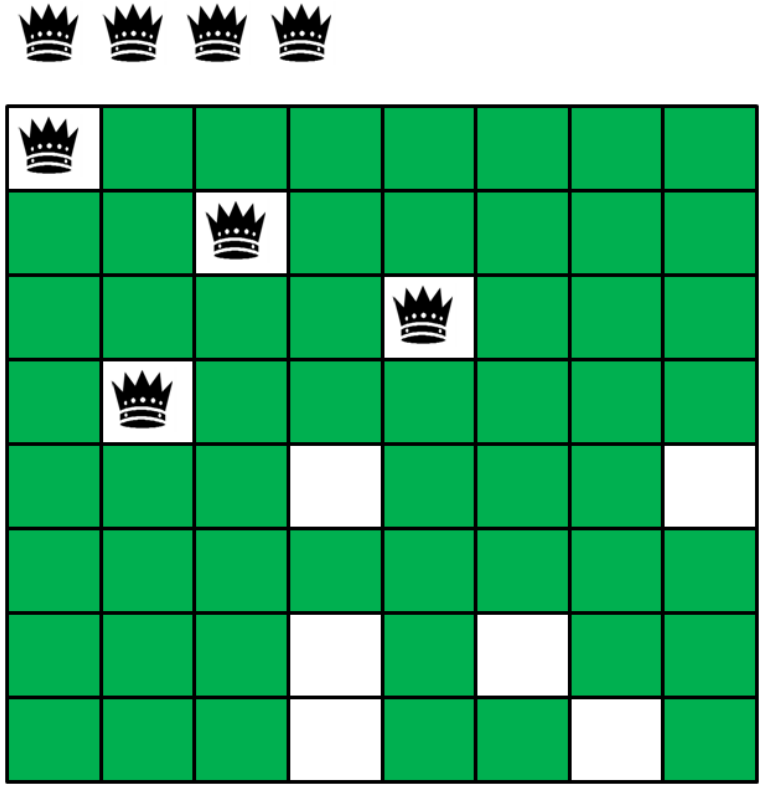
八皇后 – 回溯法1

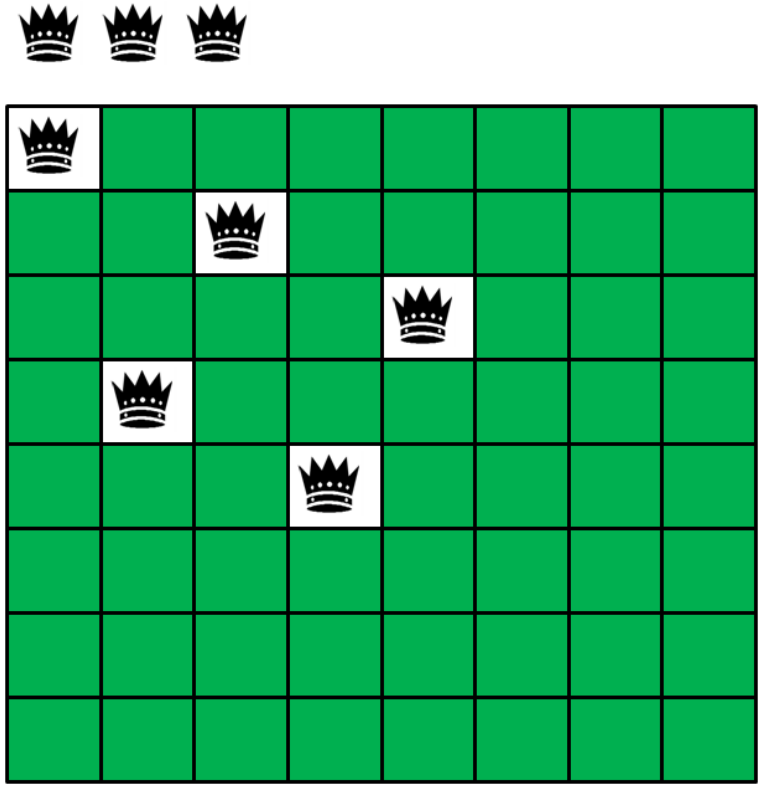


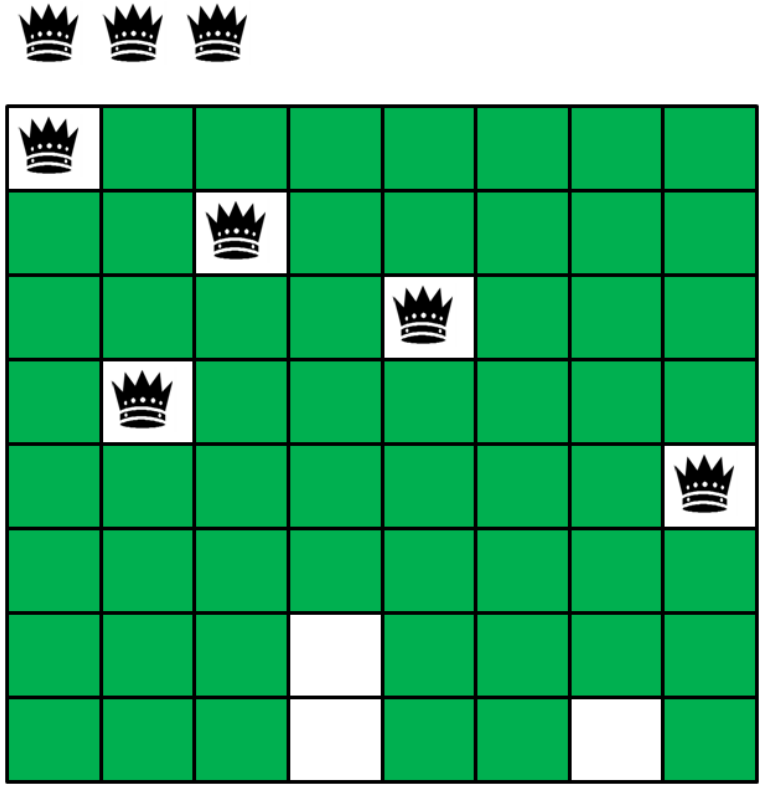




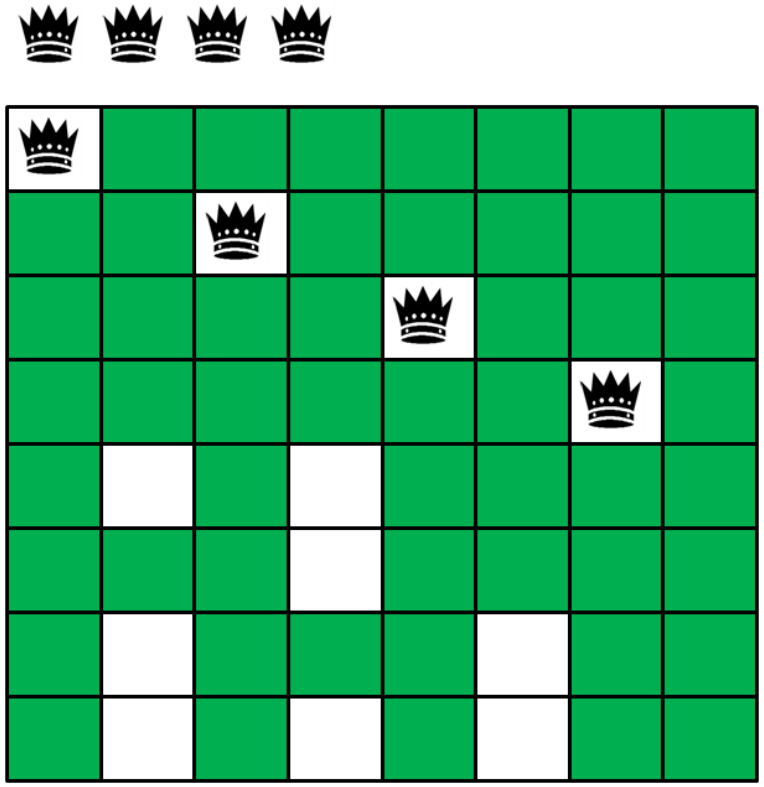
八皇后 – 回溯法2







八皇后 – 回溯法3



八皇后实现 – 合法性检查

```java
/**
 * 存放每一个皇后的列号（在第几列）
 */
private int[] queues;
/**
 * 一共有多少种合理的摆法
 */
private int ways;
```

```java
/**
 * 检查第row行第col列是否可以摆放皇后
 */
private boolean isValid(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (queues[i] == col) return false;
        if (row - i == Math.abs(col - queues[i])) return false;
    }
    return true;
}
```

八皇后实现 – 打印

```
/**
 * 显示皇后的摆放情况
 */
void show() {
    for (int row = 0; row < queues.length; row++) {
        for (int col = 0; col < queues.length; col++) {
            if (queues[row] == col) {
                System.out.print(1);
            } else {
                System.out.print(0);
            }
            System.out.print(" ");
        }
        System.out.println();
    }
    System.out.println("--------------------------");
}
```

八皇后实现 – 从某一行开始摆放皇后

```
/**
 * 从第 row 行开始摆放皇后
 */
void place(int row) {
    if (row == queues.length) {
        ways++;
        show();
        return;
    }
    for (int col = 0; col < queues.length; col++) {
        if (isValid(row, col)) {
            queues[row] = col;
            place(row + 1);
        }
    }
}
```

八皇后实现 – 摆放所有皇后

```
void placeQueens(int count) {
    if (count < 1) return;
    queues = new int[count];

    place(0);
    System.out.println(count + "皇后一共有" + ways + "种摆法");
}
```

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
----------------------------
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
----------------------------
4皇后一共有2种摆法
```

八皇后优化 – 成员变量

```
/**
 * 标记着某一列是否有皇后了
 */
boolean[] cols;
/**
 * 标记着某一对角线是否有皇后了（左上角->右下角，left top -> right bottom）
 */
boolean[] leftTop;
/**
 * 标记着某一对角线是否有皇后了（右上角->左下角，right top -> left bottom）
 */
boolean[] rightTop;
/**
 * 一共有多少种合理的摆法
 */
int ways;
```

八皇后优化 – 从某一行开始摆放皇后

```
/**
 * 从第 row 行开始摆放皇后
 */
void place(int row) {
    if (row == cols.length) {
        ways++;
        return;
    }
    for (int col = 0; col < cols.length; col++) {
        if (cols[col]) continue;
        int ltIndex = row - col + cols.length - 1;
        if (leftTop[ltIndex]) continue;
        int rtIndex = row + col;
        if (rightTop[rtIndex]) continue;
        cols[col] = leftTop[ltIndex] = rightTop[rtIndex] = true;
        place(row + 1);
        cols[col] = leftTop[ltIndex] = rightTop[rtIndex] = false;
    }
}
```

八皇后优化 – 摆放所有皇后

```
/**
 * 摆放count个皇后
 */
void placeQueens(int count) {
    if (count < 1) return;
    cols = new boolean[count];
    leftTop = new boolean[(count << 1) - 1];
    rightTop = new boolean[leftTop.length];

    place(0);
    System.out.println(count + "皇后一共有" + ways + "种摆法");
}
```

八皇后优化 – 对角线

■ 左上角 -> 右下角的对角线索引：row – col + 7

	0	1	2	3	4	5	6	7
0	7	6	5	4	3	2	1	0
1	8	7	6	5	4	3	2	1
2	9	8	7	6	5	4	3	2
3	10	9	8	7	6	5	4	3
4	11	10	9	8	7	6	5	4
5	12	11	10	9	8	7	6	5
6	13	12	11	10	9	8	7	6
7	14	13	12	11	10	9	8	7

■ 右上角 -> 左下角的对角线索引：row + col

0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	0
1	2	3	4	5	6	7	8	1
2	3	4	5	6	7	8	9	2
3	4	5	6	7	8	9	10	3
4	5	6	7	8	9	10	11	4
5	6	7	8	9	10	11	12	5
6	7	8	9	10	11	12	13	6
7	8	9	10	11	12	13	14	7

八皇后优化 – 位运算

- 可以利用位运算进一步压缩八皇后的空间复杂度

```
/**
 * 标记着某一列是否有皇后了
 */
byte cols;
/**
 * 标记着某一对角线是否有皇后了（左上角->右下角，left top -> right bottom）
 */
short leftTop;
/**
 * 标记着某一对角线是否有皇后了（右上角->左下角，right top -> left bottom）
 */
short rightTop;
/**
 * 一共有多少种合理的摆法
 */
int ways;
```

八皇后优化 – 位运算

```
/**
 * 从第 row 行开始摆放皇后
 */
void place(int row) {
    if (row == 8) {
        ways++;
        return;
    }
    for (int col = 0; col < 8; col++) {
        int colV = 1 << col;
        if ((cols & colV) != 0) continue;
        int ltV = 1 << (row - col + 7);
        if ((leftTop & ltV) != 0) continue;
        int rtV = 1 << (row + col);
        if ((rightTop & rtV) != 0) continue;
        cols |= colV;
        leftTop |= ltV;
        rightTop |= rtV;
        place(row + 1);
        cols &= ~colV;
        leftTop &= ~ltV;
        rightTop &= ~rtV;
    }
}
```

```
void place8Queens() {
    place(0);
    System.out.println("8皇后一共有" + ways + "种摆法");
}
```

作业

- 全排列：https://leetcode-cn.com/problems/permutations
- 全排列 II：https://leetcode-cn.com/problems/permutations-ii/
- 组合总和：https://leetcode-cn.com/problems/combination-sum/
- 组合总和 II：https://leetcode-cn.com/problems/combination-sum-ii/
- 子集：https://leetcode-cn.com/problems/subsets/
- 子集 II：https://leetcode-cn.com/problems/subsets-ii/