
# LeetCode51，52 别再问我N皇后了

**承志**

技术公众号：TechFlow，B站：梁上唐前



本文始发于个人公众号：**TechFlow，原创不易，求个关注**

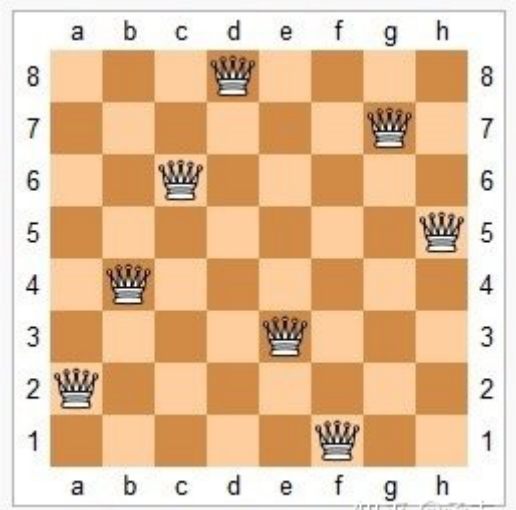
今天是**LeetCode专题第32篇**，我们来看看八皇后问题的进阶版——**N皇后**问题。

今天的文章对应LeetCode当中的51和52两题，这两题的题面几乎完全一样，都是N皇后问题，不同的是51题要求的是所有N皇后的摆放的情况，而52题只需要求所有摆放的种数。所以我们把这两题合并在一篇文章当中分享。

## N皇后问题

N皇后问题是**非常经典**的算法问题，也是面试当中的常客。早年许多面试官喜欢考察N皇后问题，本质上是想要通过这个问题考察候选人对于递归和搜索的掌握程度。递归和搜索可以说是算法的基础，也是一个高阶工程师必须掌握的内容。因此它非常的重要，现在虽然在面试当中出现得少了，但是它背后的算法的精髓却一直没有变。

我们来回顾一下N皇后的题面，在国际象棋的规则当中，皇后是最强大的。既可以横着走，也可以竖着走，还能斜着走。如果我们要在棋盘上摆放多个皇后，只要其中两个皇后**同行或者同列或者同对角线**，那么就认为她们的行动范围产生了重叠，就会产生“冲突”。然而我们不希望这样的事情发生，所以请问给定一个n*n的棋盘，要求在其中摆放n个皇后，有哪些摆放的方法？



One solution to the eight queens puzzle

当n=8的时候，就是大名鼎鼎的八皇后问题。我们也曾经在文章当中分享过，不熟悉或者新关注的同学可以点击下方传送门：

LeetCode 31：递归、回溯、八皇后、全排列一篇文章全讲清楚
mp.weixin.qq.com

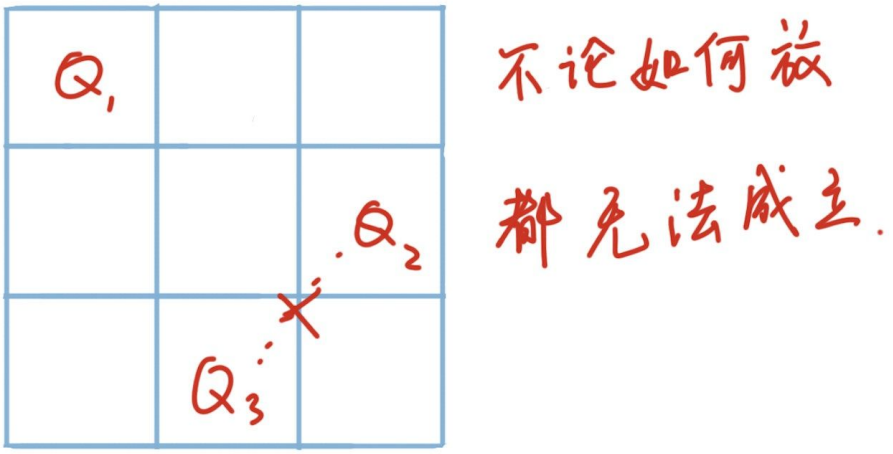
## 思路分析

八皇后问题已经是老生常谈了，在我们探讨解法之前，先来思考一个问题，用递归或者搜索解决的问题很多，**为什么只有N皇后问题如此经典呢？**

是因为国际象棋比较流行吗？还是因为这个问题很困难吗？还是它的思路很巧妙吗？由于这个问题是我自己提出的，书本上并没有相关的答案，需要我们自己思考。我们先不公布答案，带着这个问题来分析一下这道题的思路。

我个人在解题的时候喜欢问问题，很多时候看似破朔迷离找不到头绪的题目，**多问几个问题也许就能找到灵感**。如果我们对数字敏感的话，很容易发现一个大问题，为什么题目会让我们在n*n的棋盘上摆放n个皇后呢？为什么是n个，而不是2n个，也不是n+1个呢？

这个问题不难回答，因为题目当中规定了**皇后不能同行摆放**，所以每一行最多摆放一个皇后，一共有n行，那么显然最多只能摆放n个皇后。但是如果我们继续提问，既然这么多限制条件，**那为什么一定能找到答案呢**，会不会摆放n个皇后的解不存在呢？这当然是有可能的，我们很容易发现，当n=2和3的时候就没有解法。






如果我们顺着这个思路问下去，还可以挖掘出许多问题来。比如到底什么样的n可以使得一定有解呢？每一个n对应的解有没有规律呢？这样一直问下去，如果所有的问题都能解答，就说明这个问题就彻底吃透了。实际上这个问题**背后是一个非常复杂的数学问题**，会在之后开一篇文章单独讲解。

虽然不去深究这些问题，也一样可以把题目做出来，但很多时候思维和能力的差距就体现在这些看似无用的思考上。

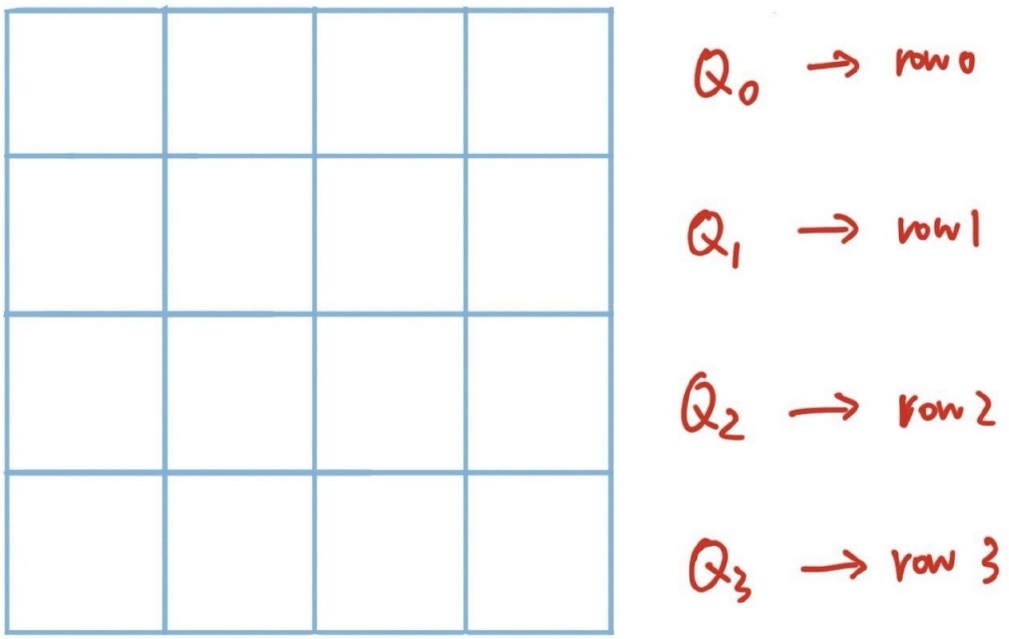
我们先把问题简化，把解的存在性问题先放一放，既然题目要我们求，一定会给我们有解的n。而且我们也可以大概猜测得出结论，当n大于等于4的时候，N皇后问题一定有解。那么，我们怎么求解呢?

## 问题建模

我们干想是没有结果的，要对问题进行**建模**。建模这个词很玄乎，听起来很高端，而且在很多场景当中都会出现。比如机器学习当中，我们需要建模，还有专门的数学建模大赛，但是少有人会对建模这个词进行解释。

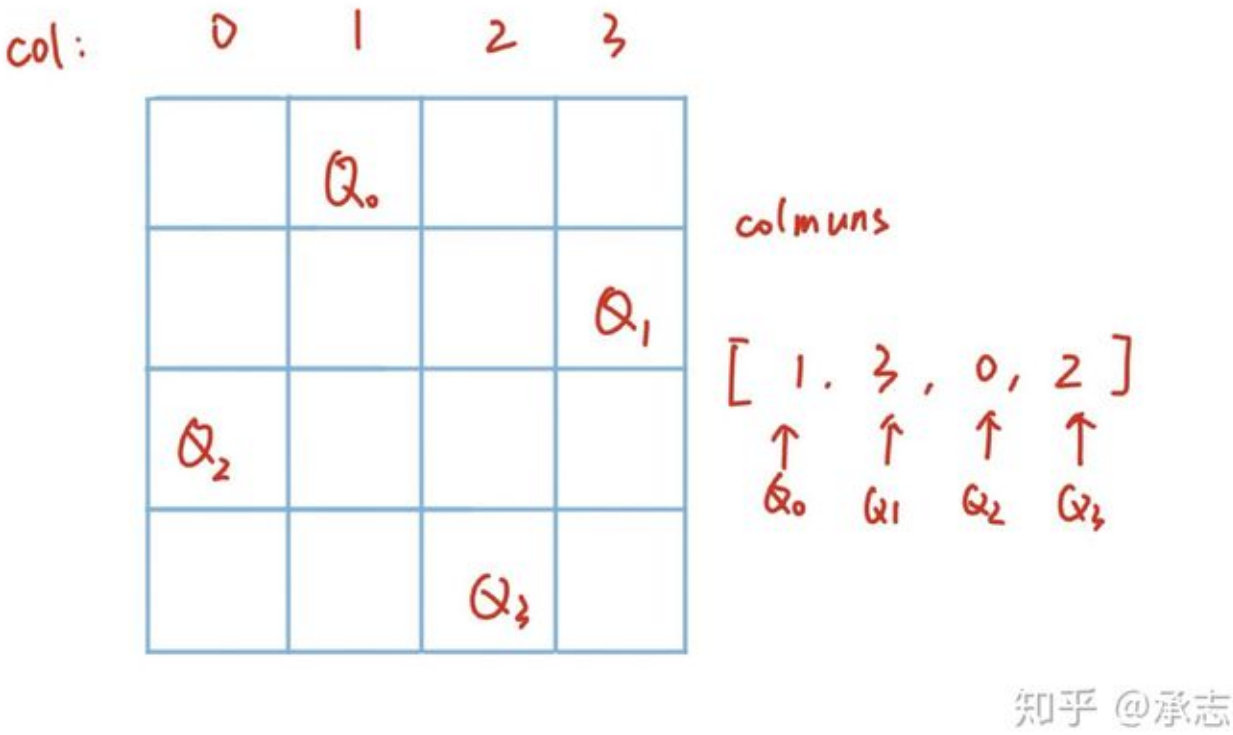
经过了一系列思考，我个人总结，所谓的建模，其实就是一个**寻找和设计适应问题的解法的过程**。模型就是从问题当中抽象出来的逻辑，比如N皇后摆放是问题本身，但是摆放的方法的逻辑才是模型。模型不是凭空出现的，是我们一点点构建的。这个过程有点像是搭积木，从无到有，从易到难，一点点将模型完善。

n*n的棋盘上摆放n个皇后，这个是问题本身，我们做第一层抽象。显然，由于皇后之间不能同行也不能同列，那么每一行和每一列只能摆放一个皇后。我们不能同时枚举一个皇后摆放的行和列，我们优先考虑其中的行。不如做一个假设，由于皇后之间没有差别，我们可以**假设每一行摆放的皇后是固定的**。第一个皇后就摆放在第一行，第二个皇后就摆放在第二行。




进行了第一层抽象之后，问题清晰了许多，但是还是无法得出答案。所以我们还需要做第二层抽象和分析，每行固定一个皇后之后可以保证皇后之间不会同行发生冲突，但是不能保证不同列以及不同对角线。所以我们必须设计一个机制，来保证这一点。我们需要枚举皇后所有摆放的情况，所以不能再固定皇后摆放的列，既然不能固定，但是可以记录。由于我们已经确定了每一个皇后摆放的行，只要**记录下它们摆放的列**，就可以判断是否会构成同列以及同对角线。

到这里，我们已经找到解法了，但是我们还可以再做第三层抽象。由于皇后已经固定了行号，我们可以**用数组当中的下标代替皇后**。下标0存储的位置就是皇后0摆放的列号，0就是皇后0的行号，那么我们用一个一维数组就存储了皇后摆放的二维信息。




也就是说我们在递归的时候，只用一个数组就记录了整个棋盘的情况，这个时候用代码实现起来就要容易很多。

```
# code for leetcode 51
class Solution:

    def dfs(self, n, queen, ret):
        if len(queen) == n:
            ret.append(queen[:])
            return

        for i in range(n):
            # 如果同列
            if i in queen:
                continue
            flag = False
            # 判断是否存在同对角线
            for j, idx in enumerate(queen):
                # len(queen)表示当前是第几个皇后
                if abs(len(queen) - j) == abs(idx - i):
                    flag = True
                    break
            if flag:
                continue
            # 合法则放入i列
```

```
            queen.append(i)
            self.dfs(n, queen, ret)
            queen.pop()

    def transform(self, n, ret):
        res = []
        # 根据每个皇后摆放的列号还原棋盘
        for arr in ret:
            s = []
            for i, idx in enumerate(arr):
                row = ['.' for _ in range(n)]
                row[idx] = 'Q'
                s.append(''.join(row))
            res.append(s)
        return res

    def solveNQueens(self, n: int) -> List[List[str]]:
        ret = []
        self.dfs(n, [], ret)
        return self.transform(n, ret)
```

## 总结

最后，我们再回到一开始的问题，为什么递归求解的问题这么多，只有N皇后成为经典呢？

我觉得很重要的一个原因就在于这道题对应的**建模过程**，我们从无到有，抽丝剥茧，一点点将整个问题搭建起来，构建出了一个适配于当前问题的模型。并且经过我们的优化，这也是用递归实现的最佳模型。对于我们而言，把问题AC了其实并不重要，重要的是能够**掌握这个思路构建的能力**，这样以后我们就可以很方便地迁移到其他的问题场景当中，这才是学习的精髓。

吐槽一下，LeetCode把题目稍微变一下就成为一种新题的做法实在是……

今天的文章就是这些，如果觉得有所收获，请顺手点个**关注或者转发**吧，你们的举手之劳对我来说很重要。

weixin.qq.com/r/kiqWjj7... (二维码自动识别)

发布于 2020-04-26

力扣（LeetCode） 递归



## 文章被以下专栏收录

TechFlow
简单文章，复杂算法

## 推荐阅读


LeetCode 题解 | -
官：N 皇后最优解

力扣（Le... 发

## 还没有评论

写下你的评论...

1 条评论被折叠（为什么？）