51-N皇后

题述

51. N 皇后

难度 困难 1362

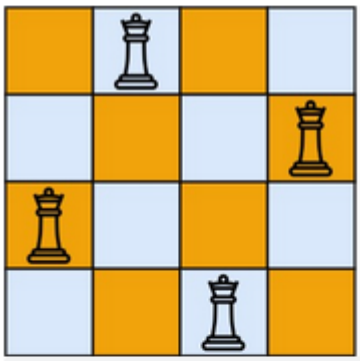
按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。

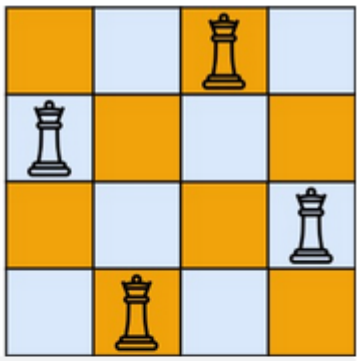
n 皇后问题 研究的是如何将 `n` 个皇后放置在 `n×n` 的棋盘上，并且使皇后彼此之间不能相互攻击。

给你一个整数 `n` ，返回所有不同的 **n 皇后问题** 的解决方案。

每一种解法包含一个不同的 **n 皇后问题** 的棋子放置方案，该方案中 `'Q'` 和 `'.'` 分别代表了皇后和空位。

示例 1：





输入： n = 4
输出： [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
解释： 如上图所示，4 皇后问题存在两个不同的解法。

示例 2：

输入： n = 1
输出： [["Q"]]

思路

N皇后问题是回溯的经典问题之一，期末考试一定会涉及！

看到这个题目，一定要先分析清楚题目的约束条件，即皇后什么情况下彼此间会有冲突：

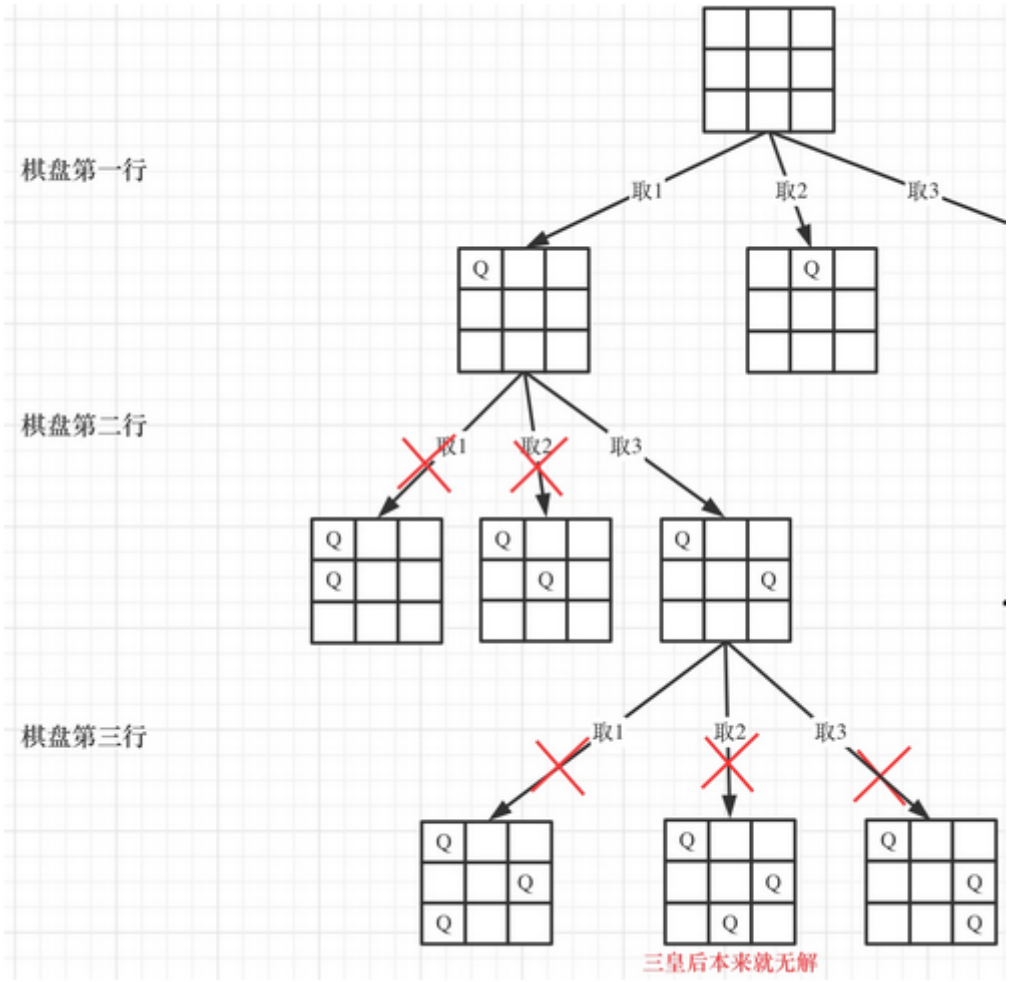
- 皇后不能在同一行
- 皇后不能在同一列
- 皇后不能在同一45度角斜线

OK，然后来分析这道题目的输入输出，输入为一个整数n，代表的是你要研究如何将n个皇后放置在n * n的棋盘上，并且他们彼此之间不能相互攻击（有冲突）。

输出是一个二维数组，返回所有不同的N皇后问题的解决方案，

```
.代表空棋格
Q代表放置皇后
```

同样的，回溯问题最关键的解题出发点是先构想出搜索树，下面我们先来看3皇后问题，构造如下搜索树：



依次进行横向纵向遍历，可以看出，二维矩阵中矩阵的高就是这颗树的高度，矩阵的宽就是搜索树每一个结点的宽度，那么我们适用皇后的约束条件，进行回溯遍历搜索，只要搜索到了树的叶子结点，那么即一个可行解。

回溯三段论

```
void backtracking(参数) {
    if (终止条件) {
        存放结果;
        return;
    }
    for (选择：本层集合中元素（树中节点孩子的数量就是集合的大小）) {
        处理节点;
        backtracking(路径，选择列表); // 递归
        回溯，撤销处理结果
    }
}
```

之前提过，横向遍历使用for循环，纵向遍历使用递归方法，回溯的搜索优先度是深度优先，即先深度后广度。

1、递归函数参数

和前面做过的组合问题一样，我们使用二维数组来记录最终结果。

在Python中，提到二维数组的添加新元素问题，就要联想到深复制，使用:

```
def backTracking(self,chessBoard,row,n):
```

- row用来记录当前遍历到棋盘的第几层了
- n是棋盘的规格 即n*n的棋盘
- chessBoard为棋盘 是一个二维数组

2、递归终止条件

回溯方法的终止条件一般都是遍历到叶子结点即可终止本次递归，进行回溯。

当递归到棋盘最底层，也就是叶子结点时，就可以收集结果并返回了。

```
def backTracking(self,chessBoard,row,n):
        #如果走到最后一行，就说明已经找到了一个可行解
        if row == n:
            tempResult = []
            for temp in chessBoard:
                tempStr = "".join(temp)
                tempResult.append(tempStr)
            self.result.append(tempResult)
            return
```

3、横向遍历逻辑（单层搜索）

递归深度：row控制棋盘的行，每一层里for循环的col控制棋盘的列，一行一列确定放置皇后的位置。

每次都是从新的一行的其实位置开始搜索，所以从0开始。

```
        for col in range(n):
                if self.isVaild(chessBoard, row, col) == False:
                    continue
                chessBoard[row][col] = 'Q'
                self.backTracking(chessBoard, row+1, n)
                chessBoard[row][col] = '.'
```

4、校验是否冲突

- 不能同行
- 不能同列
- 不能共斜线（45或135度角）

```
def isVaild(self,board,row, col):
            #判断同一列是否冲突
            for i in range(len(board)):
```

```
                if board[i][col] == 'Q':
                    return False
            # 判断左上角是否冲突
            i = row -1
            j = col -1
            while i>=0 and j>=0:
                if board[i][j] == 'Q':
                    return False
                i -= 1
                j -= 1
            # 判断右上角是否冲突
            i = row - 1
            j = col + 1
            while i>=0 and j < len(board):
                if board[i][j] == 'Q':
                    return False
                i -= 1
                j += 1
            return True
```

题解

Python

```
class Solution:
    def __init__(self):
        self.result =[]

    def isVaild(self,board,row, col):
            #判断同一列是否冲突
            for i in range(len(board)):
                if board[i][col] == 'Q':
                    return False
            # 判断左上角是否冲突
            i = row -1
            j = col -1
            while i>=0 and j>=0:
                if board[i][j] == 'Q':
                    return False
                i -= 1
                j -= 1
            # 判断右上角是否冲突
            i = row - 1
            j = col + 1
            while i>=0 and j < len(board):
                if board[i][j] == 'Q':
                    return False
                i -= 1
                j += 1
            return True

    def backTracking(self,chessBoard,row,n):
        #如果走到最后一行，就说明已经找到了一个可行解
        if row == n:
```

```
            tempResult = []
            for temp in chessBoard:
                tempStr = "".join(temp)
                tempResult.append(tempStr)
            self.result.append(tempResult)
            return
        for col in range(n):
                if self.isVaild(chessBoard, row, col) == False:
                    continue
                chessBoard[row][col] = 'Q'
                self.backTracking(chessBoard, row+1, n)
                chessBoard[row][col] = '.'

    def solveNQueens(self, n: int) -> List[List[str]]:
        self.chessBoard = [['.'] * n for i in range(n)]
        self.backTracking(self.chessBoard,0,n)
        return self.result
```

思考