回溯：八皇后

困难/递归、回溯

学习目标

拉勾教育
—互联网人实战大学—

- 掌握回溯算法的特点
- 掌握回溯算法的应用

题目描述

设计一种算法，打印 N 皇后在 N × N 棋盘上的各种摆法，其中每个皇后都不同行、不同列，也不在对角线上。这里的“对角线”指的是所有的对角线，不只是平分整个棋盘的那两条对角线。

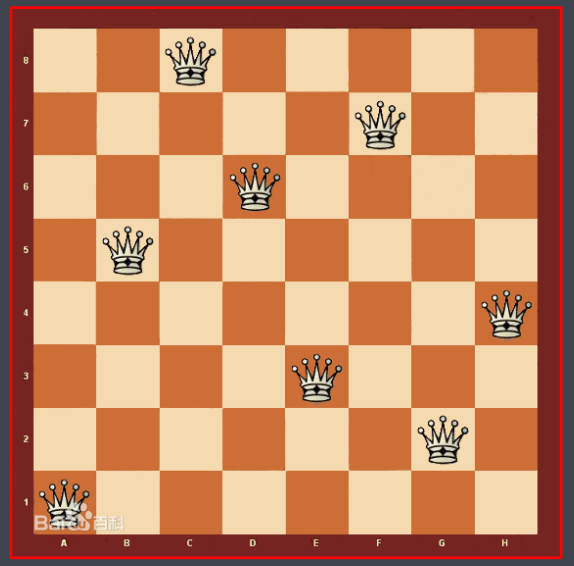
输入：4
输出：[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
解释：4 皇后问题存在如下两个不同的解法。

```
[
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

一. Comprehend 理解题意

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。皇后的走法是：可以横直斜走，格数不限。因此要求皇后彼此之间不能相互攻击，等价于要求任何两个皇后都不能在同一行、同一列以及同一条斜线上。



二. Choose 数据结构及算法思维选择

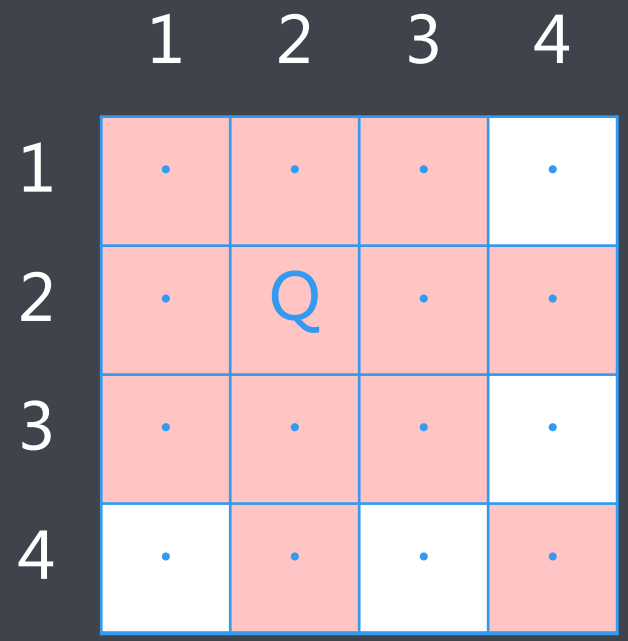
基础解法：递归（暴力解法）

- 使用递归遍历所有可能
- 数据结构：二维数组
- 算法思维：递归

三. Code 基本解法及编码实现

基础解法： 递归

我们先不看8皇后，我们先来看一下4皇后，其实原理都是一样的，比如在下面的4*4的格子里，如果我们在其中一个格子里输入了皇后，那么在这一行这一列和这左右两边的对角线上都不能有皇后。



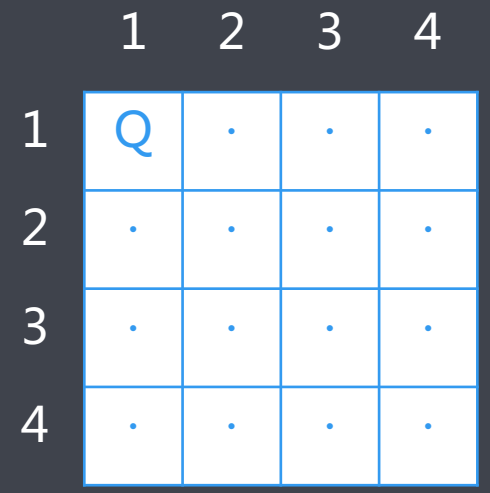
遍历每一种情况即可求解

三. Code 基本解法及编码实现

基础解法： 递归

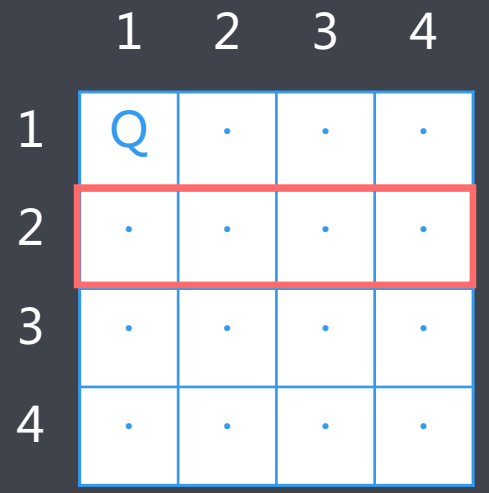
第一行

比如我们在第一行第1列输入了一个皇后



第二行

第二行我们就不能在第一列和第2列输入皇后了，因为有冲突了。但我们可以在第3列输入皇后



三. Code 基本解法及编码实现

基础解法： 递归

第三行

我们发现在任何位置输入都会有冲突。这说明我们之前选择的是错误的，再回到上一步，我们发现第二步不光能选择第3列，而且还能选择第4列，既然选择第3列不合适，那我们就选择第4列吧

第二行（重新选择）

第二行我们选择第4列

	1	2	3	4
1	Q	·	·	·
2	·	·	·	Q
3	·	·	·	·
4	·	·	·	·

第三行（重新选择）

第3行我们只有选择第2列不会有冲突

	1	2	3	4
1	Q	·	·	·
2	·	·	·	Q
3	·	Q	·	·
4	·	·	·	·

三. Code 基本解法及编码实现

基础解法： 递归

第四行

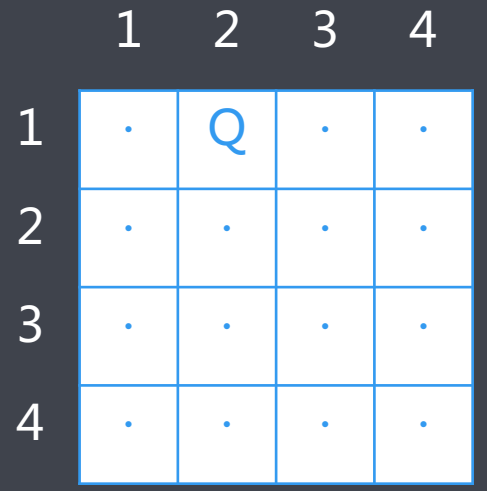
我们发现第4行又没有可选择的了。第一次重试失败

第二次重试

到这里我们只有重新回到第一步了，这说明我们之前第一行选择第一列是无解的，所以我们第一行不应该选择第一列，我们再来选择第二列来试试

第一行

这一行我们选择第2列

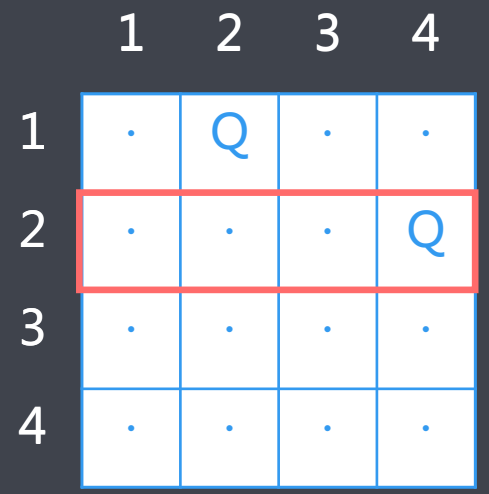


三. Code 基本解法及编码实现

基础解法： 递归

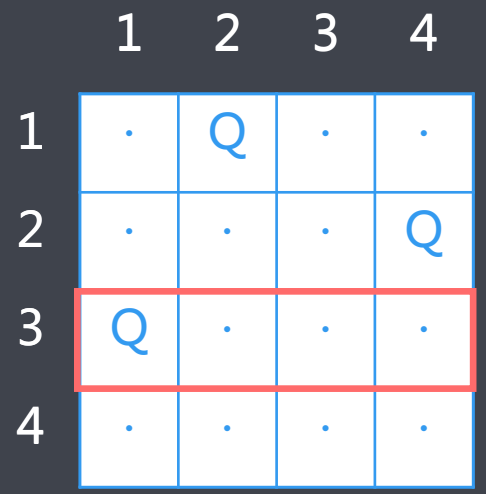
第二行

第二行我们前3个都不能选，只能选第4列



第三行

第三行我们只能选第1列



三. Code 基本解法及编码实现

基础解法： 递归

第四行

第四行我们只能选第3列

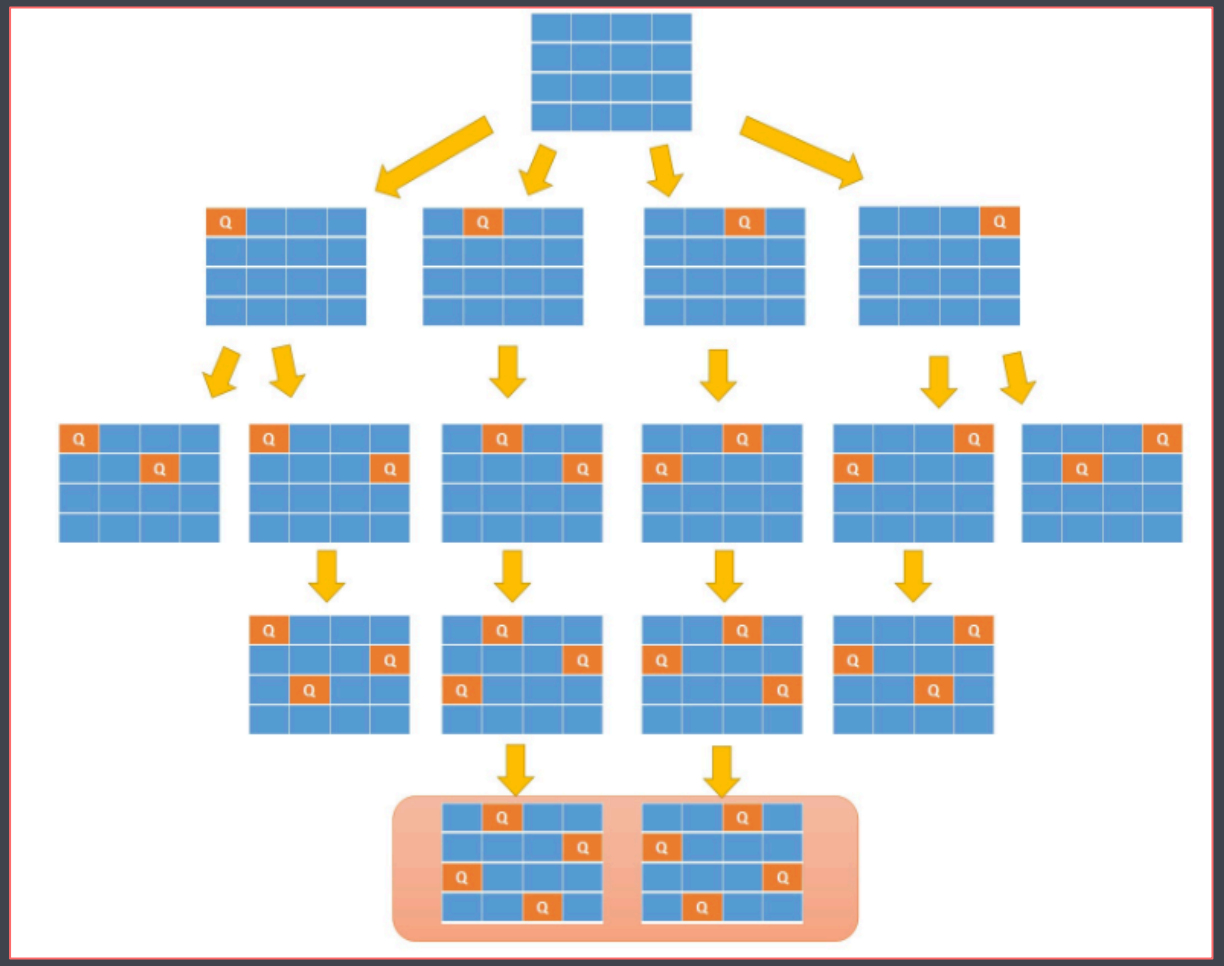
	1	2	3	4
1	·	Q	·	·
2	·	·	·	Q
3	Q	·	·	·
4	·	·	Q	·

最后我们终于找到了一组解。除了这组解还有没有其他解呢，肯定还是有的，因为4皇后是有两组解的，这里我们就不在一个个试了。

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

基础解法： 递归



最终得到两个搜索结果

三. Code 基本解法及编码实现

基础解法：递归编码实现

```
class Solution {
    public List<List<String>> solveNQueens(int n) {
        char[][] chess = new char[n][n];//初始化数组
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                chess[i][j] = '.';
        List<List<String>> res = new ArrayList<>();
        solve(res, chess, 0);
        return res;
    }
    private void solve(List<List<String>> res, char[][] chess, int row) {
        // 递归出口，最后一行都走完了，说明找到了一组，把它加入到集合res中
        if (row == chess.length) { res.add(construct(chess)); return; }
        for (int col = 0; col < chess.length; col++) { //遍历每一列
            if (valid(chess, row, col)) { // 判断这个位置是否可以放皇后
                char[][] temp = copy(chess); //数组复制一份
                temp[row][col] = 'Q';//在当前位置放个皇后
                solve(res, temp, row + 1); //递归到下一行继续
            }
        }
    }
}
```

```
class Solution {
    //把二维数组chess中的数据测下copy一份
    private char[][] copy(char[][] chess) {
        char[][] temp = new char[chess.length][chess[0].length];
        for (int i = 0; i < chess.length; i++) {
            for (int j = 0; j < chess[0].length; j++) {
                temp[i][j] = chess[i][j];
            }
        }
        return temp;
    }
    //把数组转为list
    private List<String> construct(char[][] chess) {
        List<String> path = new ArrayList<>();
        for (int i = 0; i < chess.length; i++) {
            path.add(new String(chess[i]));
        }
        return path;

    }

}
```

三. Code 基本解法及编码实现

基础解法：递归编码实现

```
class Solution {
    private boolean valid(char[][] chess, int row, int col) {
        //判断当前列有没有皇后,因为他是一行一行往下走的，
        //我们只需要检查走过的行数即可，通俗一点就是判断当前
        //坐标位置的上面有没有皇后
        for (int i = 0; i < row; i++) {
            if (chess[i][col] == 'Q') {
                return false;
            }
        }
        //判断当前坐标的右上角有没有皇后
        for (int i = row - 1, j = col + 1; i >= 0 && j < chess.length; i--, j++) {
            if (chess[i][j] == 'Q') {
                return false;
            }
        }
        //判断当前坐标的左上角有没有皇后
        for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {
            if (chess[i][j] == 'Q') {
                return false;
            }
        }
        return true;
    }

}
```

执行结果： 通过 显示详情 ›

执行用时： 9 ms，在所有 Java 提交中击败了 18.69% 的用户

内存消耗： 39.3 MB，在所有 Java 提交中击败了 25.63% 的用户

三. Code 最优解思路及编码实现

基础解法：递归复杂度分析

时间复杂度：O(N!)

- N是皇后的数量

空间复杂度：O(N^2)

- 空间复杂度取决于递归调用栈的深度+ 存储的数组空间
- 栈的最大深度是N
- 题目中用到的数组空间为O(N^2)

四. Consider 思考更优解

思考

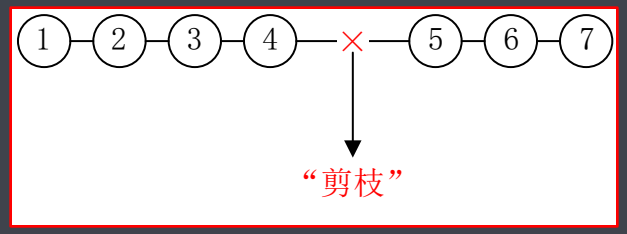
- 在基础解法中，使用一个二维数组来记录整个棋盘的状态，来判断某个位置是否可以放皇后
- 某个位置是否可以放皇后，取决于同一行、同一列、同一斜线上的状态，与其余位置无关
- 只需记录这三个方向上的状态即可，从而降低空间复杂度

四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

关键知识点：回溯

- 递归是一种算法结构，而回溯是一种算法思想，可以用递归实现
- 回溯就是一种试探，类似于穷举，但回溯有“剪枝”功能
- 如在求和问题。给定7个数字，1 2 3 4 5 6 7求和等于7的组合
- 从小到大搜索，选择1+2+3+4 =10>7，则5 6 7就没必要再继续了，这就是剪枝
- 从问题的某一种可能出发, 搜索从这种情况出发所能达到的所有可能, 当这一条路走到“尽头”的时候, 再倒回出发点, 从另一个可能出发, 继续搜索。这就是回溯



五. Code 最优解思路及编码实现

最优解：基于集合的回溯

- 使用回溯探索可能情况，用集合记录每一列以及两个方向的每条斜线上的情况
- 数组结构：集合
- 算法思维：回溯

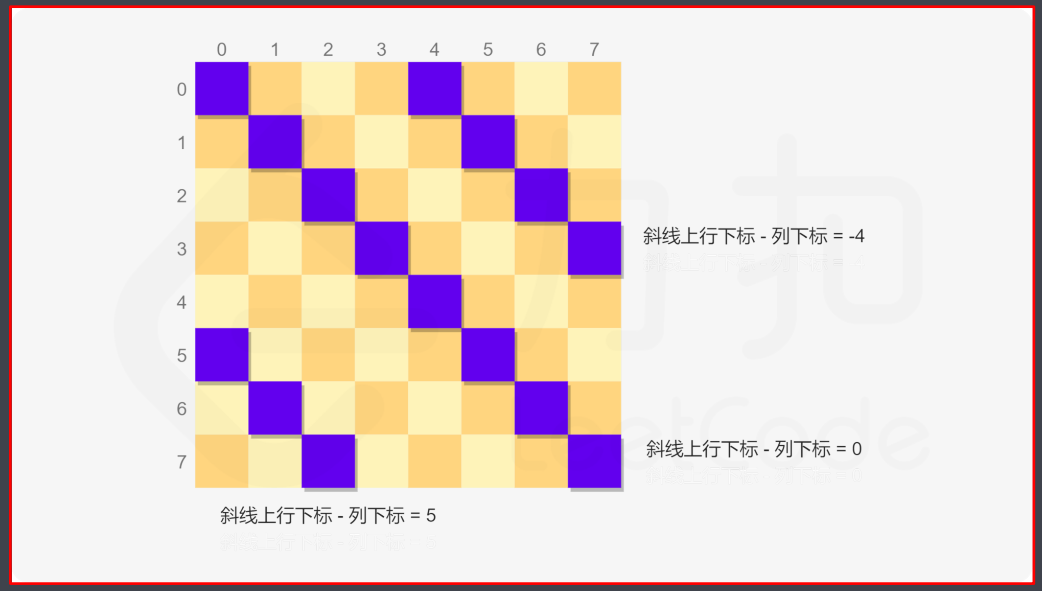
五. Code 最优解思路及编码实现

最优解：基于集合的回溯

- 只需使用三个集合columns, diagonals1, diagonals2分别记录同一列以及两条斜线上是否有皇后
- columns[i]表示第i列上是否放置了皇后
- 但是如何表示两个方向上的斜线呢？

五. Code 最优解思路及编码实现

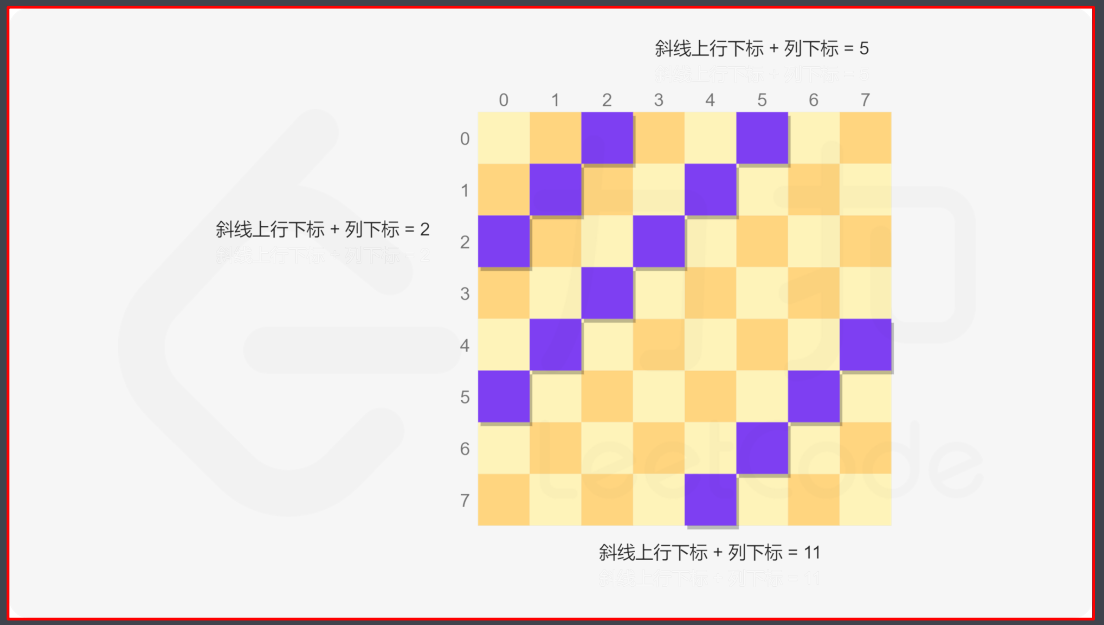
最优解：基于集合的回溯



- 方向一的斜线为从左上到右下方向，
- 同一条斜线上的每个位置满足行下标与列下标之差相等
- 因此使用行下标与列下标之差即可明确表示每一条方向一的斜线。

五. Code 最优解思路及编码实现

最优解：基于集合的回溯



- 方向二的斜线为从右上到左下方向，
- 同一条斜线上的每个位置满足行下标与列下标之和相等
- 例如 (3,0)和 (1,2)在同一条方向二的斜线上
- 用行下标与列下标之和即可明确表示每一条方向二的斜线

每次放置皇后时，对于每个位置判断其是否在三个集合中

如果三个集合都不包含当前位置，则当前位置是可以放置皇后的位置

五. Code 最优解思路及编码实现

最优解：基于集合的回溯参考代码

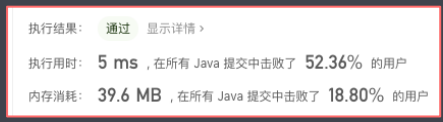
```
class Solution {
    public List<List<String>> solveNQueens(int n) {
        List<List<String>> solutions = new ArrayList<List<String>>();// 用来记录结果的数组
        int[] queens = new int[n];// (i,queens[i])放置皇后
        Arrays.fill(queens, -1);// 初始化为-1
        Set<Integer> columns = new HashSet<Integer>();// columns[i]表示第i列上以及放置了皇后
        Set<Integer> diagonals1 = new HashSet<Integer>();// 表示左上到右下的斜线，同一斜线上行列坐标之差相等
        Set<Integer> diagonals2 = new HashSet<Integer>();// 表示右上到左下的斜线，同一斜线上行列坐标之和相等
        backtrack(solutions, queens, n, 0, columns, diagonals1, diagonals2);// 开始递归
        return solutions;
    }
    //生成最终的结果
    public List<String> generateBoard(int[] queens, int n) {
        List<String> board = new ArrayList<String>();
        for (int i = 0; i < n; i++) {
            char[] row = new char[n];
            Arrays.fill(row, '.');
            row[queens[i]] = 'Q';
            board.add(new String(row));
        }
        return board;
    }
}
```

五. Code 最优解思路及编码实现

拉勾教育
—互联网人实战大学—

最优解：基于集合的回溯参考代码

```
class Solution {
    // n表示是n皇后问题；row 表示当前将要填充的行号
    public void backtrack(List<List<String>> solutions, int[] queens, int n, int row, Set<Integer> columns, Set<Integer> d
iagonals1, Set<Integer> diagonals2) {
        if (row == n) {// 递归出口
            List<String> board = generateBoard(queens, n);
            solutions.add(board);
        } else {
            for (int i = 0; i < n; i++) {// 对于当前的第row行，试图去填充每一列
                if (columns.contains(i)) {continue; } // 如果当前列上已经有皇后了，退出此次循环，填充下一列
                int diagonal1 = row - i;// 使用行列坐标只差来表示方向一(左上到右下)的斜线
                if (diagonals1.contains(diagonal1)) { continue; } // 如果方向一已经有皇后了，退出此次循环，填充下一列
                int diagonal2 = row + i;// 使用行列坐标只和来表示方向二(右上到左下)的斜线
                if (diagonals2.contains(diagonal2)) { continue; } // 如果方向二已经有皇后了，退出此次循环，填充下一列
                // 如果当前位置(row,i)可以放置皇后，更新状态
                queens[row] = i; columns.add(i); diagonals1.add(diagonal1); diagonals2.add(diagonal2);
                //继续遍历下一行
                backtrack(solutions, queens, n, row + 1, columns, diagonals1, diagonals2);
                // 上面的递归完成后，在(row,i)放置皇后对的前提下所有可能已经记录在solutions里面了，复原状态
                queens[row] = -1; columns.remove(i); diagonals1.remove(diagonal1); diagonals2.remove(diagonal2);
            }
        }
    }
}
```



五. Code 最优解思路及编码实现

最优解：基于集合的回溯复杂度分析

时间复杂度：O(N!)

- N是皇后的数量

空间复杂度：O(N)

- 空间复杂度取决于递归调用栈的深度+ 存储的数组空间
- 栈的最大深度是N
- 题目中用到的数组空间为O(N)

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

- N皇后有几种摆法

延伸扩展

- 回溯算法在解决寻找最优解的问题如N皇后、迷宫问题等应用广泛
- 回溯算法关键点是寻找剪枝条件

本题来源

- 面试题 08.12 https://leetcode-cn.com/problems/eight-queens-lcci/

总结

- 掌握回溯算法的特点
- 掌握回溯算法的应用

课后练习

1. 串联字符串最大长度(Leetcode1239/中等)
2. 计算各个位数不同的数子个数（ Leetcode 357 /中等 ）
3. 优美的排列（ Leetcode 526 /中等 ）
4. 贴纸拼词 （ Leetcode 691/困难 ）

课后练习

1. 串联字符串最大长度(Leetcode1239/中等)

   提示：给定一个字符串数组 arr，字符串 s 是将 arr 某一子序列字符串连接所得的字符串，如果 s 中的每一个字符都只出现过一次，那么它就是一个可行解。

   请返回所有可行解 s 中最长长度。

```
输入: arr = ["un","iq","ue"]
输出: 4
解释: 所有可能的串联组合是 "","un","iq","ue","uniq" 和
"ique"，最大长度为 4。
```

课后练习

2. 计算各个位数不同的数子个数（Leetcode 357 /中等）

提示：给定一个非负整数 n，计算各位数字都不同的数字 x 的个数，其中 0 ≤ x < 10^n 。

```
输入：2
输出：91
解释：答案应为除去 11,22,33,44,55,66,77,88,99 外，在
[0,100) 区间内的所有数字。
```

课后练习

3. 优美的排列（Leetcode 526 /中等）

提示：假设有从 1 到 N 的 N 个整数，如果从这 N 个数字中成功构造出一个数组，使得数组的第 i 位 (1 <= i <= N) 满足如下两个条件中的一个，我们就称这个数组为一个优美的排列。条件：

- 第 i 位的数字能被 i 整除
- i 能被第 i 位上的数字整除

现在给定一个整数 N，请问可以构造多少个优美的排列？

```
输入: 2
输出: 91
解释: 答案应为除去 11,22,33,44,55,66,77,88,99 外，在
[0,100) 区间内的所有数字。
```

课后练习

拉勾教育
—互联网人实战大学—

4. 贴纸拼词（Leetcode 691/困难）

提示：我们给出了 N 种不同类型的贴纸。每个贴纸上都有一个小写的英文单词。你希望从自己的贴纸集合中裁剪单个字母并重新排列它们，从而拼写出给定的目标字符串 target。

如果你愿意的话，你可以不止一次地使用每一张贴纸，而且每一张贴纸的数量都是无限的。

拼出目标 target 所需的最小贴纸数量是多少？如果任务不可能，则返回 -1。

=

输入：

```
["with", "example", "science"], "thehat"
```

输出：

```
3
```

解释：

```
我们可以使用 2 个 "with" 贴纸，和 1 个 "example" 贴纸。
把贴纸上的字母剪下来并重新排列后，就可以形成目标 “thehat“ 了。
此外，这是形成目标字符串所需的最小贴纸数量。
```

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容