0037. 解数独

ITCharge 大约 3 分钟

- 标签：数组、哈希表、回溯、矩阵
- 难度：困难

题目链接

- 0037. 解数独 - 力扣

题目大意

描述：给定一个二维的字符数组 $board$ 用来表示数独，其中数字 $1 \sim 9$ 表示该位置已经填入了数字，`.` 表示该位置还没有填入数字。

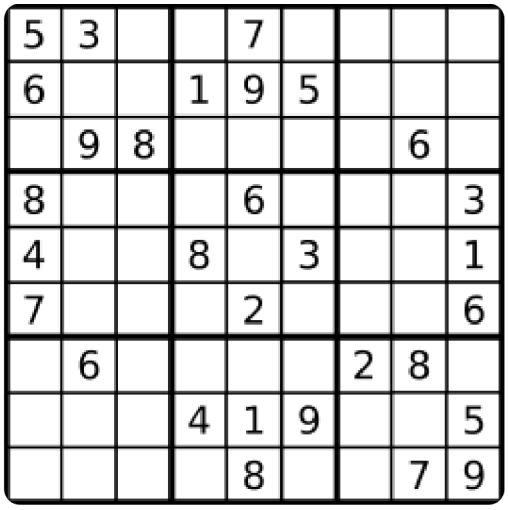
要求：现在编写一个程序，通过填充空格的方式来解决数独问题，最终不用返回答案，将题目给定 $board$ 修改为可行的方案即可。

说明：

- 数独解法需遵循如下规则：
  - 数字 $1 \sim 9$ 在每一行只能出现一次。
  - 数字 $1 \sim 9$ 在每一列只能出现一次。
  - 数字 $1 \sim 9$ 在每一个以粗直线分隔的 3×3 宫格内只能出现一次。
- $board.length == 9$。
- $board[i].length == 9$。
- $board[i][j]$ 是一位数字或者 `.`。
- 题目数据保证输入数独仅有一个解。

示例：

- 示例 1：



py

输入：board = [["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],
["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]
输出：[["5","3","4","6","7","8","9","1","2"],
["6","7","2","1","9","5","3","4","8"],["1","9","8","3","4","2","5","6","7"],
["8","5","9","7","6","1","4","2","3"],["4","2","6","8","5","3","7","9","1"],
["7","1","3","9","2","4","8","5","6"],["9","6","1","5","3","7","2","8","4"],
["2","8","7","4","1","9","6","3","5"],["3","4","5","2","8","6","1","7","9"]]
解释：输入的数独如上图所示，唯一有效的解决方案如下所示：

解题思路

思路 1：回溯算法

对于每一行、每一列、每一个数字，都需要一重 `for` 循环来遍历，这样就是三重 `for` 循环。

对于第 i 行、第 j 列的元素来说，如果当前位置为空位，则尝试将第 k 个数字置于此处，并检验数独的有效性。如果有效，则继续遍历下一个空位，直到遍历完所有空位，得到可行方案或者遍历失败时结束。

遍历完下一个空位之后再将此位置进行回退，置为 `.`。

思路 1：代码

py

```
class Solution:
    def backtrack(self, board: L    List[str]]):
        for i in range(len(board,,
            for j in range(len(board[0])):
                if board[i][j] != '.':
                    continue
                for k in range(1, 10):
                    if self.isValid(i, j, k, board):
                        board[i][j] = str(k)
                        if self.backtrack(board):
                            return True
                        board[i][j] = '.'
                return False
        return True

    def isValid(self, row: int, col: int, val: int, board: List[List[str]]) ->
bool:
        for i in range(0, 9):
            if board[row][i] == str(val):
                return False

        for j in range(0, 9):
```

```
            if board[j][col] == str(val):
                return False

        start_row = (row // 3) * 3
        start_col = (col // 3) * 3

        for i in range(start_row, start_row + 3):
            for j in range(start_col, start_col + 3):
                if board[i][j] == str(val):
                    return False
        return True

    def solveSudoku(self, board: List[List[str]]) -> None:
        self.backtrack(board)
        """
        Do not return anything, modify board in-place instead.
        """
```

思路 1：复杂度分析

- **时间复杂度**：$O(9^m)$，m 为棋盘中 `.` 的数量。
- **空间复杂度**：$O(9^2)$。