宫水三叶的刷题日记

数独问题

Author ：宫水三叶
Date ：2021/10/07
QQ Group: 703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「数独问题」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「数独问题」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「数独问题」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **36. 有效的数独** ，难度为 **中等**。

Tag：「哈希表」、「数组」、「位运算」、「数独问题」

请你判断一个 9x9 的数独是否有效。只需要 根据以下规则 ，验证已经填入的数字是否有效即可。

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。（请参考示例图）

数独部分空格内已填入了数字，空白格用 ‘.’ 表示。

注意：

- **一个有效的数独（部分已被填充）不一定是可解的。**
- **只需要根据以上规则，验证已经填入的数字是否有效即可。**

示例 1：

```
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：true
```

示例 2：

```
输入：board =
[["8","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：false

解释：除了第一行的第一个数字从 5 改为 8 以外，空格内其他数字均与 示例1 相同。 但由于位于左上角的 3x3 宫内有两个 8 存在，因此
```

提示：

- board.length == 9
- board[i].length == 9
- board[i][j] 是一位数字或者 '.'

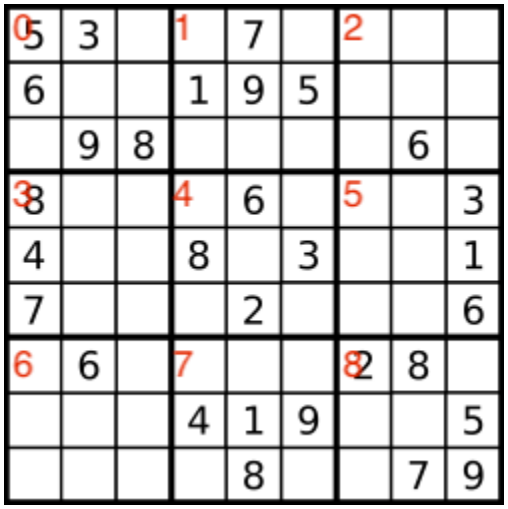
哈希表

由于只要我们判断是否为有效的数独。

所以我们只需要对 `board` 中出现的数进行判断，如果 `board` 中有数违反了数独的规则，返回 `false`，否则返回 `true`。

直观上，我们很容易想到使用 **哈希表** 来记录某行/某列/某个小方块出现过哪些数字，来帮助我们判断是否符合「有效数独」的定义。

这道题唯一的难点可能是在于如何确定某个数落在哪个小方块中，我们可以去小方块进行编号：



然后推导出小方块编号和行列的关系为：$idx = \lfloor i/3 \rfloor * 3 + \lfloor j/3 \rfloor$。

代码：

```Java
class Solution {
    public boolean isValidSudoku(char[][] board) {
        Map<Integer, Set<Integer>> row  = new HashMap<>(), col = new HashMap<>(), area = n
        for (int i = 0; i < 9; i++) {
            row.put(i, new HashSet<>());
            col.put(i, new HashSet<>());
            area.put(i, new HashSet<>());
        }
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                char c = board[i][j];
                if (c == '.') continue;
                int u = c - '0';
                int idx = i / 3 * 3 + j / 3;
                if (row.get(i).contains(u) || col.get(j).contains(u) || area.get(idx).cont
                row.get(i).add(u);
                col.get(j).add(u);
                area.get(idx).add(u);
            }
        }
        return true;
    }
}
```

- 时间复杂度：在固定 $9 * 9$ 的问题里，计算量不随数据变化而变化。复杂度为 $O(1)$
- 空间复杂度：在固定 $9 * 9$ 的问题里，存储空间不随数据变化而变化。复杂度为 $O(1)$

数组

大多数的哈希表计数问题，都能转换为使用数组解决。

虽然时间复杂度一样，但哈希表的更新和查询复杂度为均摊 $O(1)$，而定长数组的的更新和查询复杂度则是严格 $O(1)$。

因此从执行效率上来说，数组要比哈希表快上不少。

代码：

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {
        boolean[][] row = new boolean[10][10], col = new boolean[10][10], area = new boole
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                int c = board[i][j];
                if (c == '.') continue;
                int u = c - '0';
                int idx = i / 3 * 3 + j / 3;
                if (row[i][u] || col[j][u] || area[idx][u]) return false;
                row[i][u] = col[j][u] = area[idx][u] = true;
            }
        }
        return true;
    }
}
```

- 时间复杂度：在固定 $9 * 9$ 的问题里，计算量不随数据变化而变化。复杂度为 $O(1)$
- 空间复杂度：在固定 $9 * 9$ 的问题里，存储空间不随数据变化而变化。复杂度为 $O(1)$

位运算

更进一步，我们可以使用一个 int 来记录 某行/某列/某个小方块 的数值填入情况：使用从低位开始的 $[1, 9]$ 位来记录该数值是否已被填入。

例如 $(...111000111.)_2$ 代表数值 $[1, 3]$ 和 $[7, 9]$ 均被填入。

代码：

```java
class Solution {
    public boolean isValidSudoku(char[][] board) {
        int[] row = new int[10], col = new int[10], area = new int[10];
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                char c = board[i][j];
                if (c == '.') continue;
                int u = c - '0';
                int idx = i / 3 * 3 + j / 3;
                if ((((row[i] >> u) & 1) == 1) || (((col[j] >> u) & 1) == 1) || (((area[id
                row[i] |= (1 << u);
                col[j] |= (1 << u);
                area[idx] |= (1 << u);
            }
        }
        return true;
    }
}
```

- 时间复杂度：在固定 $9 * 9$ 的问题里，计算量不随数据变化而变化。复杂度为 $O(1)$
- 空间复杂度：在固定 $9 * 9$ 的问题里，存储空间不随数据变化而变化。复杂度为 $O(1)$

题目描述

这是 LeetCode 上的 **37. 解数独** ，难度为 **困难**。

Tag：「回溯算法」、「DFS」、「数独问题」

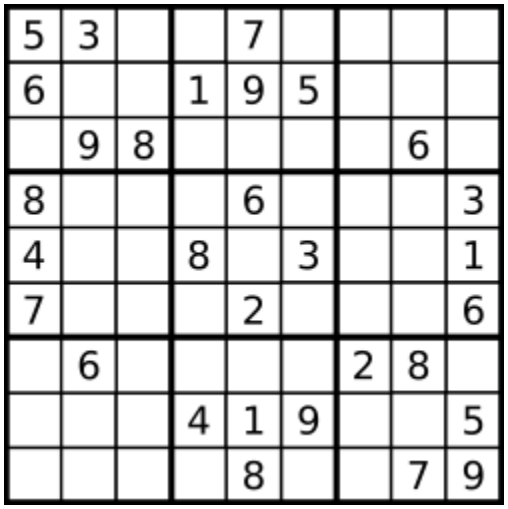
编写一个程序，通过填充空格来解决数独问题。

数独的解法需 遵循如下规则：

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。（请参考示例图）

数独部分空格内已填入了数字，空白格用 ‘.’ 表示。

示例：



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：

提示：

- 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

回溯解法

和 N 皇后一样，是一道回溯解法裸题。

上一题「36. 有效的数独（中等）」是让我们判断给定的 `borad` 是否为有效数独。

这题让我们对给定 `board` 求数独，由于 `board` 固定是 `9*9` 的大小，我们可以使用回溯算法去做。

这一类题和 N 皇后一样，属于经典的回溯算法裸题。

这类题都有一个明显的特征，就是数据范围不会很大，如该题限制了范围为 `9*9`，而 N 皇后的 N 一般不会超过 13。

对每一个需要填入数字的位置进行填入，如果发现填入某个数会导致数独解不下去，则进行回溯。

代码：

```Java
class Solution {
    boolean[][] row = new boolean[9][9];
    boolean[][] col = new boolean[9][9];
    boolean[][][] cell = new boolean[3][3][9];
    public void solveSudoku(char[][] board) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if (board[i][j] != '.') {
                    int t = board[i][j] - '1';
                    row[i][t] = col[j][t] = cell[i / 3][j / 3][t] = true;
                }
            }
        }
        dfs(board, 0, 0);
    }
    boolean dfs(char[][] board, int x, int y) {
        if (y == 9) return dfs(board, x + 1, 0);
        if (x == 9) return true;
        if (board[x][y] != '.') return dfs(board, x, y + 1);
        for (int i = 0; i < 9; i++) {
            if (!row[x][i] && !col[y][i] && !cell[x / 3][y / 3][i]) {
                board[x][y] = (char)(i + '1');
                row[x][i] = col[y][i] = cell[x / 3][y / 3][i] = true;
                if (dfs(board, x, y + 1)) {
                    break;
                } else {
                    board[x][y] = '.';
                    row[x][i] = col[y][i] = cell[x / 3][y / 3][i] = false;
                }
            }
        }
        return board[x][y] != '.';
    }
}
```

- 时间复杂度：在固定 `9*9` 的棋盘里，具有一个枚举方案的最大值（极端情况，假设我们的棋盘刚开始是空的，这时候每一个格子都要枚举，每个格子都有可能从 1 枚举到 9，所以枚举次数为 $9^{9} = 729$），即复杂度不随数据变化而变化。复杂度为 $O(1)$
- 空间复杂度：在固定 `9*9` 的棋盘里，复杂度不随数据变化而变化。复杂度为 $O(1)$

💡**更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。**

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「数独问题」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：

"给作者手机充个电"

YOLO 的赞赏码