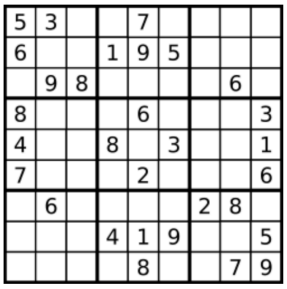
1 有效的数独

36. 有效的数独

难度 中等 413

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。



上图是一个部分填充的有效的数独。

数独部分空格内已填入了数字，空白格用 `'.'` 表示。

示例 1：

```
输入：
[
  ["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：
[
  ["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，因此这个数独是无效的。
```

说明：

- 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。
- 给定数独序列只包含数字 `1-9` 和字符 `'.'` 。
- 给定数独永远是 `9x9` 形式的。

思路：

给数独的每一行，每一列，每一个块，都设置为集合，当board判断不为空，则是否在对应行，对应列，对应块，在就返回false，

不在就添加进去

```python
def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """

        cow = [set() for _ in range(9)]    # 行
        col = [set() for _ in range(9)]    # 列
        block = [set() for _ in range(9)]  # 块

         for i in range(9):
             for j in range(9):
                 if board[i][j] != '.':

                     if board[i][j] in cow[i]:
                         return False
                     if board[i][j] in col[j]:
                         return False
                     if board[i][j] in block[(i // 3) * 3 + j //3 ]:
                         return False
                     cow[i].add(board[i][j])
                     col[j].add(board[i][j])
                     block[(i // 3) * 3 + j //3 ].add(board[i][j])
         return True
```

时间复杂度：O(1)

空间复杂度：O(1)

解数独：

编写一个程序，通过已填充的空格来解决数独问题。

一个数独的解法需**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。

空白格用 `'.'` 表示。

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

一个数独。

答案被标成红色。

Note:

- 给定的数独序列只包含数字 `1-9` 和字符 `'.'` 。
- 你可以假设给定的数独只有唯一解。
- 给定数独永远是 `9x9` 形式的。

思路：

1 每一行，每一列，每一块都设置成set，在里面加入应添加的数：1，2，。。。9

2 先遍历，如果board[i][j]不为空，说明那里不能添加了，就在对应的行，列，块，中把对应元素删除

并且把空的位置，添加到 empty列表里

3 递归操作，就是在空的位置添加数，

递归终止条件：

全部填满，iter = len(empty)

递归函数：

1 从empty队列中找到对应的位置，(i,j),从而确定块b，如果数val在行列，块的集合中，就把那个位置设置为那个数，递归。

```
for val in row[i] & col[j] & block[b]:
    board[i][j] = str(val)
```

因为执行这个操作，把`val`添加到了`board`里，所以需要在`row`，`col`，`block`中把对应值删除，递归结束后，进行回溯，把值再添加进去

```
def solveSudoku(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        row = [set(range(1,10)) for _ in range(9)]
        col = [set(range(1,10)) for _ in range(9)]
        block = [set(range(1,10)) for _ in range(9)]

        empty = []
         for i in range(9):
             for j in range(9):
                 if board[i][j] != '.':
                     val = int(board[i][j])
                     row[i].remove(val)
                     col[j].remove(val)
                     block[(i//3) * 3 + j//3].remove(val)
                 else:
                     empty.append((i,j))


         def backtrack(iter = 0):
```

```
            # 递归终止条件:
            if iter == len(empty):
                return True
            i, j = empty[iter]
            b = (i // 3) * 3 + j // 3
            for val in row[i] & col[j] & block[b]:
                row[i].remove(val)
                col[j].remove(val)
                block[b].remove(val)
                board[i][j] = str(val)
                if backtrack(iter + 1):
                    return True
                row[i].add(val)
                col[j].add(val)
                block[b].add(val)
            return False

        backtrack()
```

太牛了