L18 Sudoku Solver

Monday, January 16, 2023 12:56 AM

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules:**

1. Each of the digits 1-9 must occur exactly once in each row.
2. Each of the digits 1-9 must occur exactly once in each column.
3. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

The '.' character indicates empty cells.

↳ This problem is the **Standard** based on the **Recursion and Backtracking**

Example 1:

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

Input: 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"]]

Output:
[["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"]]

Explanation: The input board is shown above and the only valid solution is shown below:

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

↳ Sudoku Solver is the **9X9 Matrix** and we placed the Numbers in **3X3 Matrix** 1 to 9 with out Repetation

We have some Standard Rule that we follow to Solve this problem

→ The digit 1 to 9 only appears once at every row
→ The digit 1 to 9 only appears once at any Column
→ The digit 1 to 9 all element appears at once time in any particular 3X3 Matrix

→ That is the same sets of Rule that we Remembered during to Solved this problem

Let is Understand

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

→ As we can see in 3X3 Matrix every Number 1 to 9 only appears at once time

← Every Column every element 1 to 9 appears at once

↑ Every Row have 1 to 9 only one time appears

That is the fundamental of the Sudoku Solver

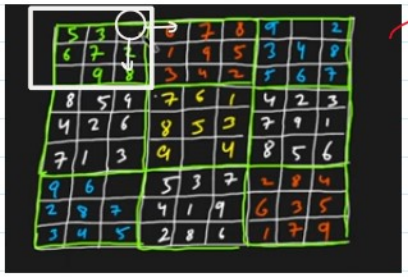
Every Row have 1 to 9 only one time appears one Value

3	4	5	2	8	6	1	7	9

That is the fundamental of the Sudoku Solver

Thought Process

→ The first Approach that Comes into the my mind **Try every possible way (Recursion)**



→ Now what can I do is I will find the first empty block and try to Apply our Rules on that

Now try to Apply our all 3 Rules

if I put 1

→ Not any 1 present into that Column
→ Not any 1 present into the that Row
→ Not any 1 present into the 3x3 unit Matrix

→ All 3 condition are true So we placed 1 in the cell

→ Basically Our thought Process is try to finding the empty cell and apply our 3 Rules if all those Rules are Valid then put the element into the box

→ Try all possible way (1 to 9) digit of the Number

Recursion Tree

Answer

Now we can not check for the 5 to 9 element

→ Column Condition
↓ Row Condition
3x3 Matrix Condition

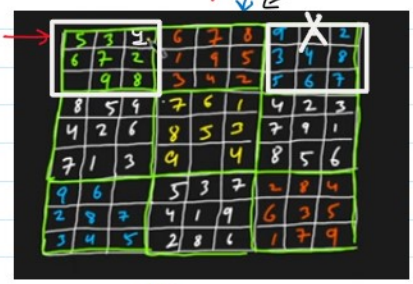
→ Because the Question Says only one Valid Sudoku that's way

→ if Question Says All the Valid Sudoku then Check All the Combination (1 to 9)

false

1 2 3 4 5 6 7 8 9

true



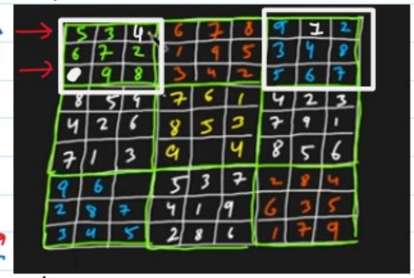
→ that Means if I placed 1 into the 3x3 Matrix then we can not placed any further element into the next Row

true

any further element into the next row

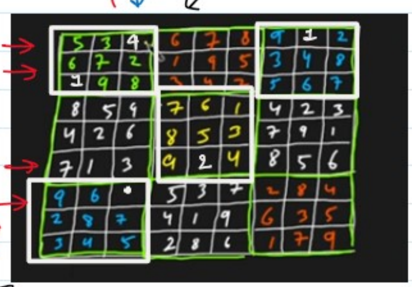
→ that means that is not a Valid Sudoko Return False

True

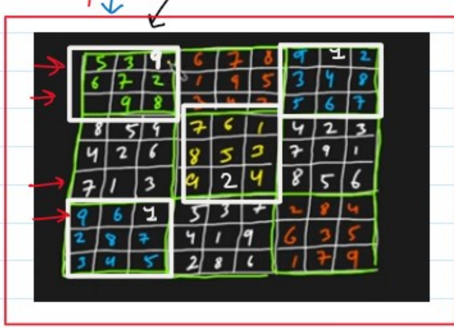


True

True



True



→ that is the my Valid Sudoko and Return True

Time Complexity: O(9(n ^ 2)), in the worst case, for each cell in the n2 board, we have 9 possible numbers.

Space Complexity: O(1), since we are refilling the given board itself, there is no extra space required, so constant space complexity.

Time Complexity = $O(9^{(n^2)})$

Space Complexity = $O(1)$

```python
class Solution:

    def isValid(self, board, row, coll, ele):

        # my all 3 conditions
        for i in range(9):

            # for coll condition
            if board[i][coll] == ele:
                return False

            # for row condition
            if board[row][i] == ele:
                return False

            # for 3X3 matrix Condition
            if board[3 * (row // 3) + i // 3][3 * (coll // 3) + i % 3] == ele:
                return False

        return True

    def solved(self, board):

        # traverse in the matrix
        for i in range(len(board)):
            for j in range(len(board[0])):

                # finding the empty place
                if board[i][j] == '.':

                    for ele in range(1, 10):

                        # try to put the 1 to 9 all the combination for finding the valid Sudoko
                        if self.isValid(board, i, j, str(ele)):
                            board[i][j] = str(ele)

                            if self.solved(board) == True:
                                return True
                            else:
                                # if false so backtrack and removed previus element
                                board[i][j] = "."
                        # for invalid sudoko
                        return False

        # if any empty position is not found
        return True

    def solveSudoku(self, board):

        self.solved(board)
```

```java
public class L18_Sudoku_Solver {

    public static void main(String[] args) {
        System.out.println("L18_Sudoku_Solver");
    }
}



class Solution12 {


    public static boolean solved(char[][] board) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if (board[i][j] == '.') {

                    for (char c = '1'; c <= '9'; c++) {
                        if (isValid(board, i, j, c)) {
                            board[i][j] = c;

                            if (solved(board))
                                return true;
                            else
                                board[i][j] = '.';
                        }
                    }

                    return false;
                }
            }
        }
        return true;
    }


    public static boolean isValid(char[][] board, int row, int col, char c) {
        for (int i = 0; i < 9; i++) {
            if (board[i][col] == c)
                return false;

            if (board[row][i] == c)
                return false;

            if (board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
                return false;
        }
        return true;
    }


    public void solveSudoku(char[][] board) {

        solved(board);

    }
}
```