Problem 37: Sudoku Solver

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

A sudoku solution must satisfy

all of the following rules

:

Each of the digits

1-9

must occur exactly once in each row.

Each of the digits

1-9

must occur exactly once in each column.

Each of the digits

1-9

must occur exactly once in each of the 9

3x3

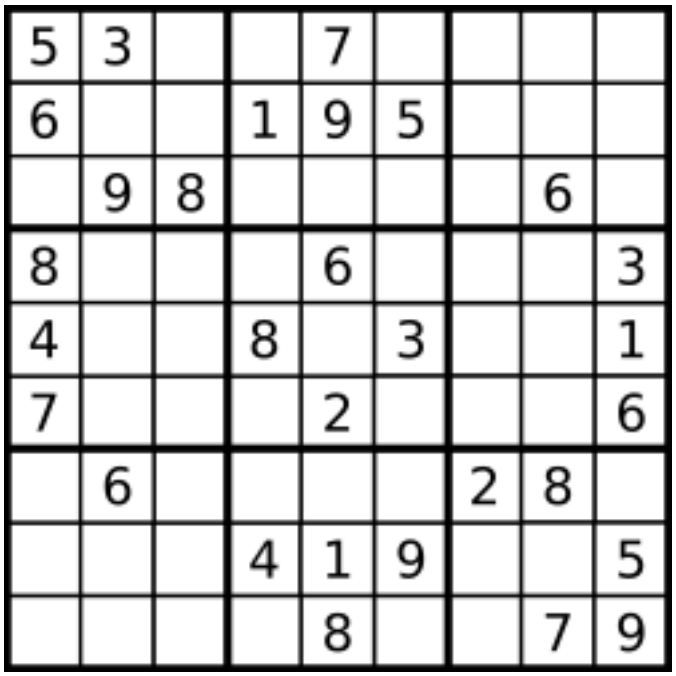
sub-boxes of the grid.

The

'.'

character indicates empty cells.

Example 1:



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

Constraints:

board.length == 9

board[i].length == 9

board[i][j]

is a digit or

'.'

.

It is

guaranteed

that the input board has only one solution.

Code Snippets

C++:

```
class Solution {
public:
void solveSudoku(vector<vector<char>>& board) {

}
};
```

Java:

```
class Solution {
public void solveSudoku(char[][] board) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def solveSudoku(self, board):
"""
:type board: List[List[str]]
:rtype: None Do not return anything, modify board in-place instead.
"""
```

JavaScript:

```
/**
* @param {character[][]} board
* @return {void} Do not return anything, modify board in-place instead.
```

```
*/
var solveSudoku = function(board) {

};
```

TypeScript:

```
/**
Do not return anything, modify board in-place instead.
*/
function solveSudoku(board: string[][]): void {

};
```

C#:

```
public class Solution {
public void SolveSudoku(char[][] board) {

}
}
```

C:

```
void solveSudoku(char** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func solveSudoku(board [][]byte) {

}
```

Kotlin:

```
class Solution {
fun solveSudoku(board: Array<CharArray>): Unit {

}
}
```

Swift:

```
class Solution {
func solveSudoku(_ board: inout [[Character]]) {

}
}
```

Rust:

```
impl Solution {
pub fn solve_sudoku(board: &mut Vec<Vec<char>>) {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @return {Void} Do not return anything, modify board in-place instead.
def solve_sudoku(board)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @return NULL
*/
function solveSudoku(&$board) {

}
}
```

Dart:

```
class Solution {
void solveSudoku(List<List<String>> board) {

}
```

```
}
```

Scala:

```
object Solution {
def solveSudoku(board: Array[Array[Char]]): Unit = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
void solveSudoku(vector<vector<char>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Sudoku Solver
Difficulty: Hard
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def solveSudoku(self, board: List[List[str]]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def solveSudoku(self, board):
"""
:type board: List[List[str]]
:rtype: None Do not return anything, modify board in-place instead.
"""
```

JavaScript Solution:

```
/**
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {character[][]} board
* @return {void} Do not return anything, modify board in-place instead.
*/
var solveSudoku = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
Do not return anything, modify board in-place instead.
*/
function solveSudoku(board: string[][]): void {

};
```

C# Solution:

```
/*
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public void SolveSudoku(char[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Sudoku Solver
* Difficulty: Hard
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

void solveSudoku(char** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Sudoku Solver
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func solveSudoku(board [][]byte) {

}
```

Kotlin Solution:

```
class Solution {
fun solveSudoku(board: Array<CharArray>): Unit {

}
}
```

Swift Solution:

```
class Solution {
func solveSudoku(_ board: inout [[Character]]) {

}
}
```

Rust Solution:

```
// Problem: Sudoku Solver
// Difficulty: Hard
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn solve_sudoku(board: &mut Vec<Vec<char>>) {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @return {Void} Do not return anything, modify board in-place instead.
def solve_sudoku(board)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String[][] $board
* @return NULL
*/
function solveSudoku(&$board) {

}
}
```

Dart Solution:

```
class Solution {
void solveSudoku(List<List<String>> board) {

}
}
```

Scala Solution:

```
object Solution {
def solveSudoku(board: Array[Array[Char]]): Unit = {

}
}
```