Problem 36: Valid Sudoku

Problem Information

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

Problem Description

Determine if a

9 x 9

Sudoku board is valid. Only the filled cells need to be validated

according to the following rules

:

Each row must contain the digits

1-9

without repetition.

Each column must contain the digits

1-9

without repetition.

Each of the nine

3 x 3

sub-boxes of the grid must contain the digits

1-9

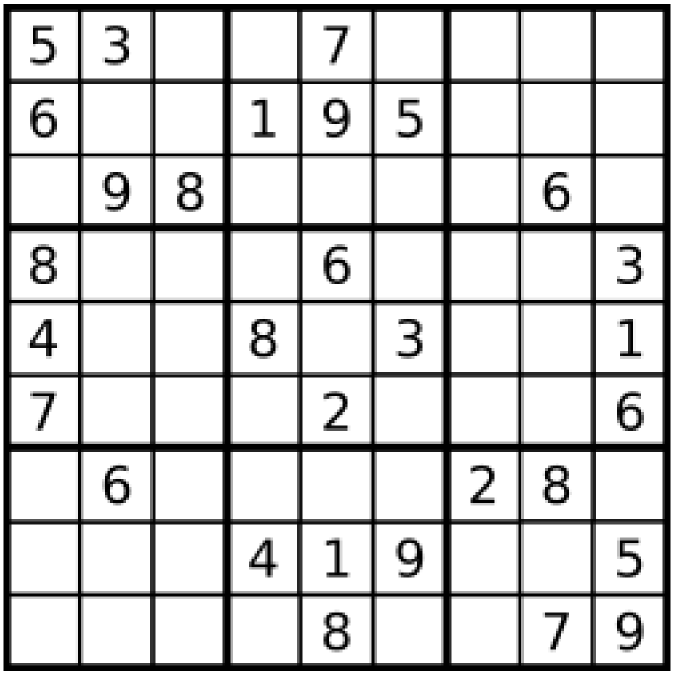
without repetition.

Note:

A Sudoku board (partially filled) could be valid but is not necessarily solvable.

Only the filled cells need to be validated according to the mentioned rules.

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:

true

Example 2:

Input:

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"]]

Output:

false

Explanation:

Same as Example 1, except with the

5

in the top left corner being modified to

8

. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

Constraints:

board.length == 9

board[i].length == 9

board[i][j]

is a digit

1-9

or

'.'

.

Code Snippets

C++:

```
class Solution {
public:
bool isValidSudoku(vector<vector<char>>& board) {

}
};
```

Java:

```
class Solution {
public boolean isValidSudoku(char[][] board) {

}
}
```

Python3:

```
class Solution:
def isValidSudoku(self, board: List[List[str]]) -> bool:
```

Python:

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

JavaScript:

```
/**
* @param {character[][]} board
* @return {boolean}
*/
var isValidSudoku = function(board) {

};
```

TypeScript:

```
function isValidSudoku(board: string[][]): boolean {

};
```

C#:

```
public class Solution {
public bool IsValidSudoku(char[][] board) {

}
}
```

C:

```
bool isValidSudoku(char** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func isValidSudoku(board [][]byte) bool {

}
```

Kotlin:

```
class Solution {
fun isValidSudoku(board: Array<CharArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func isValidSudoku(_ board: [[Character]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_valid_sudoku(board: Vec<Vec<char>>) -> bool {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @return {Boolean}
def is_valid_sudoku(board)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @return Boolean
*/
function isValidSudoku($board) {

}
}
```

Dart:

```
class Solution {
bool isValidSudoku(List<List<String>> board) {

}
}
```

Scala:

```
object Solution {
def isValidSudoku(board: Array[Array[Char]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_valid_sudoku(board :: [[char]]) :: boolean
def is_valid_sudoku(board) do

end
end
```

Erlang:

```
-spec is_valid_sudoku(Board :: [[char()]]) -> boolean().
is_valid_sudoku(Board) ->
.
```

Racket:

```
(define/contract (is-valid-sudoku board)
(-> (listof (listof char?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Sudoku
* Difficulty: Medium
* 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:
bool isValidSudoku(vector<vector<char>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Sudoku
* Difficulty: Medium
* 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 boolean isValidSudoku(char[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Sudoku
Difficulty: Medium
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 isValidSudoku(self, board: List[List[str]]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Valid Sudoku
* Difficulty: Medium
* 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 {boolean}
*/
var isValidSudoku = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Sudoku
* Difficulty: Medium
* 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
```

```
*/

function isValidSudoku(board: string[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Sudoku
* Difficulty: Medium
* 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 bool IsValidSudoku(char[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Valid Sudoku
* Difficulty: Medium
* 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
*/

bool isValidSudoku(char** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Valid Sudoku
// Difficulty: Medium
// 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 isValidSudoku(board [][]byte) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isValidSudoku(board: Array<CharArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isValidSudoku(_ board: [[Character]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Sudoku
// Difficulty: Medium
// 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 is_valid_sudoku(board: Vec<Vec<char>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @return {Boolean}
def is_valid_sudoku(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @return Boolean
*/
function isValidSudoku($board) {

}
}
```

Dart Solution:

```
class Solution {
bool isValidSudoku(List<List<String>> board) {

}
}
```

Scala Solution:

```
object Solution {
def isValidSudoku(board: Array[Array[Char]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_valid_sudoku(board :: [[char]]) :: boolean
def is_valid_sudoku(board) do

end
end
```

Erlang Solution:

```
-spec is_valid_sudoku(Board :: [[char()]]) -> boolean().
is_valid_sudoku(Board) ->
.
```

Racket Solution:

```
(define/contract (is-valid-sudoku board)
(-> (listof (listof char?)) boolean?)
)
```