JavaScript Valid Sudoku

Challenge

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.

1st Example

```
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
```

2nd Example

```
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 `'.'`.

Solution

```
const isValidSudoku = (board) => {
    const map = {};
```

Solution continues on next page...

```
    for (let i = 0; i < 9; i++) {
        for (let j = 0; j < 9; j++) {
            let num = board[i][j],
                x   = Math.floor(i / 3),
                y   = Math.floor(j / 3),
                err = (map['r' + i + num] ||
                       map['c' + j + num] ||
                       map['b' + x + y + num]);

            if (board[i][j] === '.') {
                continue;
            }

            if (err) {
                return false;
            }

            map['r' + i + num] = 1;

            map['c' + j + num] = 1;

            map['b' + x + y + num] = 1;
        }
    }

    return true;
};
```

Explanation

I've defined a function called `isValidSudoku` that takes a Sudoku board as input and checks if it is a valid Sudoku solution. It returns `true` if the board is valid and `false` otherwise.

Inside the function, an empty object called `map` is initialized. This object will be used to keep track of the numbers that have already appeared in each row, column, and `3x3` box of the Sudoku board.

Two nested loops are used to iterate over each cell of the Sudoku board. The outer loop iterates over the rows (`i`) and the inner loop iterates over the columns (`j`).

Inside the loop, several variables are defined: `num` represents the number in the current cell of the board, `x` represents the index of the `3x3` box in the row, `y` represents the index of the `3x3` box in the column, and `err` is a flag indicating whether there is an error (duplicate number) in the current row, column, or `3x3` box.

The function checks if the number in the current cell is a dot (`'.'`). If it is, it means the cell is empty, and the rest of the code for this cell is skipped using the `continue` statement.

If there is an error (`err` is truthy), it means the current number has already appeared in the same row, column, or `3x3` box. In this case, the function returns `false` to indicate that the Sudoku board is not valid.

If there is no error, the `map` object is updated by setting the corresponding keys to `1`. These keys are constructed using the row index (`r`), column index (`c`), and `3x3` box indices (`b`) concatenated with the number (`num`). This marks the number as seen in the respective row, column, and `3x3` box.

After the loops have finished iterating over all the cells, the function returns `true` to indicate that the Sudoku board is valid.

In summary, the `isValidSudoku` function checks if a given Sudoku board is valid by ensuring that no number is repeated in the same row, column, or `3x3` box. It uses an object (`map`) to keep track of the numbers that have already appeared in each row, column, and box.

Author: Trevor Morin

Copyright 2024 Superklok Labs