<https://leetcode.com/problems/valid-sudoku/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.

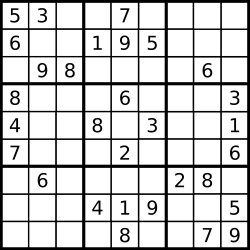
Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

**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:**



**Input:**

[["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:**

[["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 '.'.

**Attempt 1: 2025-08-03**

**Solution 1: Hash Table (10 min)**

**Style 1: 3 loops**

class Solution {

    public boolean isValidSudoku(char[][] board) {

        Set<Character> set = new HashSet<>();

        // Check for each row (fix row number i and increase column number j)

        for(int i = 0; i < 9; i++) {

            for(int j = 0; j < 9; j++) {

                if(board[i][j] != '.' && !set.add(board[i][j])) {

                    return false;

                }

            }

            set.clear();

        }

        // Check for each column (fix column number j and increase row number i)

        for(int j = 0; j < 9; j++) {

            for(int i = 0; i < 9; i++) {

                if(board[i][j] != '.' && !set.add(board[i][j])) {

                    return false;

                }

            }

            set.clear();

        }

        // Check for matrix (there are nine 3 \* 3 matrix need to check)

        for(int k = 0; k < 9; k++) {

            int m = (k / 3) \* 3;

            int n = (k % 3) \* 3;

            for(int i = m; i < m + 3; i++) {

                for(int j = n; j < n + 3; j++) {

                    if(board[i][j] != '.' && !set.add(board[i][j])) {

                        return false;

                    }

                }

            }

            set.clear();

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 2: 3 loops**

/\*\*

\* 这道题利用的是HashSet的唯一性来帮助check。

\* 先按每行check，如果是'.'说明还没填字，是合法的，往下走，如果没在set中存过就加一下，

\* 如果便利过程中出现了在set中存在的key值，说明有重复的数字在一行，不合法，return false。

\* 再按照这个方法check列。

\* 最后按照这个方法check小方块。

\* 注意小方块的ij取法。对于当前这块板子来说，总共有9个小方格，按0~8从左到右依次编号。

\* 按编号求'/'就是求得当前小方格的第一行横坐标，因为每个小方格有3行，所以循环3次。

\* 按编号求'%'就是求得当前小方格的第一列纵坐标，因为每个小方格有3列，所以循环3次。

\* 对9个小方格依次走一边，就完成了检查小方格的工作。

\*

\* There is a more official checking method on matrix, first check 1st dimension as

\* row number by divide 3, then check 2nd dimension as column number by mod 3

\* // Check for each sub-grid(matrix)

for (int k = 0; k < 9; k++) {

for (int i = k / 3 \* 3; i < k / 3 \* 3 + 3; i++) {

for (int j = (k % 3) \* 3; j < (k % 3) \* 3 + 3; j++) {

if (board[i][j] == '.')

continue;

if (set.contains(board[i][j]))

return false;

set.add(board[i][j]);

}

}

set.clear();

}

\*/

class Solution {

    public boolean isValidSudoku(char[][] board) {

        Set<Character> set = new HashSet<Character>();

        // Check for each row (fix row number i and increase column number j)

        for(int i = 0; i < 9; i++) {

            for(int j = 0; j < 9; j++) {

                // Encouter '.' will ignore as only the filled cells need to be validated

                if(board[i][j] != '.' && !set.add(board[i][j])) {

                    return false;

                }

            }

            set.clear();

        }

        // Check for each column (fix column number j and increase row number i)

        for(int j = 0; j < 9; j++) {

            for(int i = 0; i < 9; i++) {

                if(board[i][j] != '.' && !set.add(board[i][j])) {

                    return false;

                }

            }

            set.clear();

        }

        // Check for matrix (there are nine 3 \* 3 matrix need to check)

        for(int m = 0; m < 3; m++) {

            for(int n = 0; n < 3; n++) {

                for(int i = m \* 3; i < m \* 3 + 3; i++) {

                    for(int j = n \* 3; j < n \* 3 + 3; j++) {

                        if(board[i][j] != '.' && !set.add(board[i][j])) {

                            return false;

                        }

                    }

                }

                // For square, not clear after each row or column check, only

                // clear after adding 3 \* 3 elements in one matrix

                set.clear();

            }

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Style 3: 1 loop**

class Solution {

    public boolean isValidSudoku(char[][] board) {

        Set<Character>[] rows = new HashSet[9];

        Set<Character>[] cols = new HashSet[9];

        Set<Character>[] boxes = new HashSet[9];

        for(int i = 0; i < 9; i++) {

            rows[i] = new HashSet<>();

            cols[i] = new HashSet<>();

            boxes[i] = new HashSet<>();

        }

        for(int i = 0; i < 9; i++) {

            for(int j = 0; j < 9; j++) {

                char c = board[i][j];

                if(c == '.') {

                    continue;

                }

                // Check row

                if(!rows[i].add(c)) {

                    return false;

                }

                // Check col

                if(!cols[j].add(c)) {

                    return false;

                }

                // Check box

                int boxIdx = (i / 3) \* 3 + j / 3;

                if(!boxes[boxIdx].add(c)) {

                    return false;

                }

            }

        }

        return true;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

This problem requires us to determine if a given 9x9 Sudoku board is valid according to Sudoku rules. The board doesn't need to be solvable, it just needs to follow the basic rules:

Each row must contain digits 1-9 without repetition

Each column must contain digits 1-9 without repetition

Each of the 9 3x3 sub-boxes must contain digits 1-9 without repetition

**Approach**

We'll use three sets of hash sets to track seen numbers in rows, columns, and sub-boxes:

**Row tracking**: 9 hash sets (one for each row)

**Column tracking**: 9 hash sets (one for each column)

**Sub-box tracking**: 9 hash sets (one for each 3x3 sub-box)

For each cell in the board:

Skip if it's empty ('.')

Check if the number exists in the corresponding row, column, or sub-box set

If it exists, return false (invalid Sudoku)

If not, add it to all three relevant sets

**Solution Code**

import java.util.HashSet;

import java.util.Set;

class Solution {

public boolean isValidSudoku(char[][] board) {

// Initialize sets for rows, columns, and sub-boxes

Set<Character>[] rows = new HashSet[9];

Set<Character>[] cols = new HashSet[9];

Set<Character>[] boxes = new HashSet[9];

for (int i = 0; i < 9; i++) {

rows[i] = new HashSet<>();

cols[i] = new HashSet<>();

boxes[i] = new HashSet<>();

}

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

char num = board[i][j];

if (num == '.') continue;

// Check row

if (rows[i].contains(num)) {

return false;

}

rows[i].add(num);

// Check column

if (cols[j].contains(num)) {

return false;

}

cols[j].add(num);

// Check sub-box

int boxIndex = (i / 3) \* 3 + j / 3;

if (boxes[boxIndex].contains(num)) {

return false;

}

boxes[boxIndex].add(num);

}

}

return true;

}

}

**Explanation**

**Initialization**: We create three arrays of hash sets to track numbers in rows, columns, and sub-boxes.

**Iterating through the board**:

For each cell, if it's empty ('.'), we skip it

For non-empty cells:

Check if the number exists in the current row's set - if yes, return false

Check if the number exists in the current column's set - if yes, return false

Calculate the sub-box index using (i / 3) \* 3 + j / 3 and check if the number exists in that sub-box's set - if yes, return false

**Adding numbers to sets**: If a number passes all checks, it's added to the corresponding row, column, and sub-box sets

**Return true**: If we complete the iteration without finding any duplicates, the board is valid

The time complexity is O(1) since we always process a 9x9 board, and space complexity is O(1) for the same reason (we use a fixed number of hash sets).

This solution efficiently checks all Sudoku validity conditions in a single pass through the board.

**Refer to**

[L37.Sudoku Solver](note://AEF375E022AF421AB858359AF9045537)

[L2133.Check if Every Row and Column Contains All Numbers](note://WEBfdf31b1322c8a4b1fb7c2cf200177f7b)