Check if given Sudoku solution is valid or not

Given a 2D array, **board[][]** of size **9 × 9**, which represents a solution to the Sudoku puzzle, the task is to check if the given representation of a solved Sudoku puzzle is valid or not.

Examples:

Input:

```
board[][] = {{7, 9, 2, 1, 5, 4, 3, 8, 6},
             {6, 4, 3, 8, 2, 7, 1, 5, 9},
             {8, 5, 1, 3, 9, 6, 7, 2, 4},
             {2, 6, 5, 9, 7, 3, 8, 4, 1},
             {4, 8, 9, 5, 6, 1, 2, 7, 3},
             {3, 1, 7, 4, 8, 2, 9, 6, 5},
             {1, 3, 6, 7, 4, 8, 5, 9, 2},
             {9, 7, 4, 2, 1, 5, 6, 3, 8},
             {5, 2, 8, 6, 3, 9, 4, 1, 7}}
```

Output: *Valid*

Input:

```
board[][] = {{5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5},
             {5, 5, 5, 5, 5, 5, 5, 5, 5}}
```

Output: *Not Valid*

Approach: The problem can be solved by checking the following conditions:

- Check if each row of the **board[][]** array stores only unique values from the range [**1**, **9**] or not.
- Check if each column of the **board[][]** array stores unique values from the range [**1**, **9**] or not.
- Check if all possible **3 × 3 submatrices** of the **board[][]** array stores unique values from the range [**1**, **9**] or not.

Follow the steps below to solve the problem:

- Traverse the given matrix **board[][]**.
- Check if the above conditions are satisfied or not.
- If any of the above conditions is not satisfied, then print "**Not valid**".
- Otherwise, print "**Valid**".

Below is the implementation of the above approach:

C++

```
// C++ program to implement
// the above approach

#include <bits/stdc++.h>
using namespace std;
#define N 9
```

```
// Function to check if all elements
// of the board[][] array store
// value in the range[1, 9]
bool isinRange(int board[][N])
{

    // Traverse board[][] array
    for (int i = 0; i < N;
         i++) {
        for (int j = 0; j < N;
             j++) {

            // Check if board[i][j]
            // lies in the range
            if (board[i][j] <= 0 || board[i][j] > 9) {
                return false;
            }
        }
    }
    return true;
}

// Function to check if the solution
// of sudoku puzzle is valid or not
bool isValidSudoku(int board[][N])
{
    // Check if all elements of board[][]
    // stores value in the range[1, 9]
    if (isinRange(board)
        == false) {
        return false;
    }

    // Stores unique value
    // from 1 to N
    bool unique[N + 1];

    // Traverse each row of
    // the given array
    for (int i = 0; i < N; i++) {

        // Initialize unique[]
        // array to false
        memset(unique, false,
               sizeof(unique));

        // Traverse each column
        // of current row
        for (int j = 0; j < N;
             j++) {

            // Stores the value
            // of board[i][j]
            int Z = board[i][j];

            // Check if current row
            // stores duplicate value
            if (unique[Z]) {
                return false;
            }
            unique[Z] = true;
        }
    }

    // Traverse each column of
    // the given array
    for (int i = 0; i < N; i++) {

        // Initialize unique[]
        // array to false
        memset(unique, false,
               sizeof(unique));

        // Traverse each row
        // of current column
        for (int j = 0; j < N;
             j++) {

            // Stores the value
            // of board[j][i]
            int Z = board[j][i];

            // Check if current column
            // stores duplicate value
            if (unique[Z]) {
                return false;
            }
            unique[Z] = true;
        }
    }

    // Traverse each block of
    // size 3 * 3 in board[][] array
    for (int i = 0; i < N - 2;
         i += 3) {

        // j stores first column of
        // each 3 * 3 block
        for (int j = 0; j < N - 2;
             j += 3) {

            // Initialize unique[]
            // array to false
            memset(unique, false,
                   sizeof(unique));
            // Traverse current block
            for (int k = 0; k < 3;
                 k++) {

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

                    // Stores row number
                    // of current block
                    int X = i + k;

                    // Stores column number
                    // of current block
                    int Y = j + l;

                    // Stores the value
                    // of board[X][Y]
                    int Z = board[X][Y];

                    // Check if current block
                    // stores duplicate value
                    if (unique[Z]) {
                        return false;
                    }
                    unique[Z] = true;
                }
            }
        }
    }

    // If all conditions satisfied
    return true;
}

// Driver Code
int main()
{
    int board[N][N]
        = { { 7, 9, 2, 1, 5, 4, 3, 8, 6 },
            { 6, 4, 3, 8, 2, 7, 1, 5, 9 },
            { 8, 5, 1, 3, 9, 6, 7, 2, 4 },
            { 2, 6, 5, 9, 7, 3, 8, 4, 1 },
            { 4, 8, 9, 5, 6, 1, 2, 7, 3 },
            { 3, 1, 7, 4, 8, 2, 9, 6, 5 },
            { 1, 3, 6, 7, 4, 8, 5, 9, 2 },
            { 9, 7, 4, 2, 1, 5, 6, 3, 8 },
            { 5, 2, 8, 6, 3, 9, 4, 1, 7 } };

    if (isValidSudoku(board)) {
        cout << "Valid";
    }
    else {
        cout << "Not Valid";
    }
}
```

Java

```
// Java program to implement
// the above approach
import java.io.*;
import java.util.*;

class GFG{

static int N = 9;

// Function to check if all elements
// of the board[][] array store
// value in the range[1, 9]
static boolean isinRange(int[][] board)
{

    // Traverse board[][] array
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
        {

            // Check if board[i][j]
            // lies in the range
            if (board[i][j] <= 0 ||
                board[i][j] > 9)
            {
                return false;
            }
        }
    }
    return true;
}

// Function to check if the solution
// of sudoku puzzle is valid or not
static boolean isValidSudoku(int board[][])
{

    // Check if all elements of board[][]
    // stores value in the range[1, 9]
    if (isinRange(board) == false)
    {
        return false;
    }

    // Stores unique value
```

```
        // from 1 to N
        boolean[] unique = new boolean[N + 1];

        // Traverse each row of
        // the given array
        for(int i = 0; i < N; i++)
        {

            // Initialize unique[]
            // array to false
            Arrays.fill(unique, false);

            // Traverse each column
            // of current row
            for(int j = 0; j < N; j++)
            {

                // Stores the value
                // of board[i][j]
                int Z = board[i][j];

                // Check if current row
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each column of
        // the given array
        for(int i = 0; i < N; i++)
        {

            // Initialize unique[]
            // array to false
            Arrays.fill(unique, false);

            // Traverse each row
            // of current column
            for(int j = 0; j < N; j++)
            {

                // Stores the value
                // of board[j][i]
                int Z = board[j][i];

                // Check if current column
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each block of
        // size 3 * 3 in board[][] array
        for(int i = 0; i < N - 2; i += 3)
        {

            // j stores first column of
            // each 3 * 3 block
            for(int j = 0; j < N - 2; j += 3)
            {

                // Initialize unique[]
                // array to false
                Arrays.fill(unique, false);

                // Traverse current block
                for(int k = 0; k < 3; k++)
                {
                    for(int l = 0; l < 3; l++)
                    {

                        // Stores row number
                        // of current block
                        int X = i + k;

                        // Stores column number
                        // of current block
                        int Y = j + l;

                        // Stores the value
                        // of board[X][Y]
                        int Z = board[X][Y];

                        // Check if current block
                        // stores duplicate value
                        if (unique[Z])
                        {
                            return false;
                        }
                        unique[Z] = true;
                    }
                }
            }
        }

        // If all conditions satisfied
        return true;
    }
```

```
// Driver Code
public static void main(String[] args)
{
    int[][] board = { { 7, 9, 2, 1, 5, 4, 3, 8, 6 },
                      { 6, 4, 3, 8, 2, 7, 1, 5, 9 },
                      { 8, 5, 1, 3, 9, 6, 7, 2, 4 },
                      { 2, 6, 5, 9, 7, 3, 8, 4, 1 },
                      { 4, 8, 9, 5, 6, 1, 2, 7, 3 },
                      { 3, 1, 7, 4, 8, 2, 9, 6, 5 },
                      { 1, 3, 6, 7, 4, 8, 5, 9, 2 },
                      { 9, 7, 4, 2, 1, 5, 6, 3, 8 },
                      { 5, 2, 8, 6, 3, 9, 4, 1, 7 } };

    if (isValidSudoku(board))
    {
        System.out.println("Valid");
    }
    else
    {
        System.out.println("Not Valid");
    }
}
}

// This code is contributed by akhilsaini
```

Python3

```
# Python3 program to implement
# the above approach

# Function to check if all elements
# of the board[][] array store
# value in the range[1, 9]
def isinRange(board):

  N = 9

  # Traverse board[][] array
  for i in range(0, N):
    for j in range(0, N):

      # Check if board[i][j]
      # lies in the range
      if ((board[i][j] <= 0) or
          (board[i][j] > 9)):
        return False

  return True
```

```
# Function to check if the solution
# of sudoku puzzle is valid or not
def isValidSudoku(board):

  N = 9

  # Check if all elements of board[][]
  # stores value in the range[1, 9]
  if (isinRange(board) == False):
    return False

  # Stores unique value
  # from 1 to N
  unique = [False] * (N + 1)

  # Traverse each row of
  # the given array
  for i in range(0, N):

    # Initialize unique[]
    # array to false
    for m in range(0, N + 1):
      unique[m] = False

    # Traverse each column
    # of current row
    for j in range(0, N):

      # Stores the value
      # of board[i][j]
      Z = board[i][j]

      # Check if current row
      # stores duplicate value
      if (unique[Z] == True):
        return False

      unique[Z] = True

  # Traverse each column of
  # the given array
  for i in range(0, N):

    # Initialize unique[]
    # array to false
    for m in range(0, N + 1):
      unique[m] = False

    # Traverse each row
    # of current column
    for j in range(0, N):

      # Stores the value
      # of board[j][i]
      Z = board[j][i]

      # Check if current column
      # stores duplicate value
      if (unique[Z] == True):
        return False

      unique[Z] = True

  # Traverse each block of
  # size 3 * 3 in board[][] array
  for i in range(0, N - 2, 3):

    # j stores first column of
    # each 3 * 3 block
    for j in range(0, N - 2, 3):

      # Initialize unique[]
      # array to false
      for m in range(0, N + 1):
        unique[m] = False

      # Traverse current block
      for k in range(0, 3):
        for l in range(0, 3):

          # Stores row number
          # of current block
          X = i + k

          # Stores column number
          # of current block
          Y = j + l

          # Stores the value
          # of board[X][Y]
          Z = board[X][Y]

          # Check if current block
          # stores duplicate value
          if (unique[Z] == True):
            return False

          unique[Z] = True

  # If all conditions satisfied
  return True

# Driver Code
if __name__ == '__main__':

  board = [ [ 7, 9, 2, 1, 5, 4, 3, 8, 6 ],
            [ 6, 4, 3, 8, 2, 7, 1, 5, 9 ],
            [ 8, 5, 1, 3, 9, 6, 7, 2, 4 ],
            [ 2, 6, 5, 9, 7, 3, 8, 4, 1 ],
            [ 4, 8, 9, 5, 6, 1, 2, 7, 3 ],
            [ 3, 1, 7, 4, 8, 2, 9, 6, 5 ],
            [ 1, 3, 6, 7, 4, 8, 5, 9, 2 ],
            [ 9, 7, 4, 2, 1, 5, 6, 3, 8 ],
            [ 5, 2, 8, 6, 3, 9, 4, 1, 7 ] ]

  if (isValidSudoku(board)):
    print("Valid")
  else:
    print("Not Valid")

# This code is contributed by akhilsaini
```

C#

```
// C# program to implement
// the above approach
using System;
using System.Collections.Generic;

class GFG{

static int N = 9;

// Function to check if all elements
// of the board[][] array store
// value in the range[1, 9]
static bool isinRange(int[, ] board)
{

    // Traverse board[][] array
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
        {

            // Check if board[i][j]
            // lies in the range
            if (board[i, j] <= 0 ||
                board[i, j] > 9)
            {
                return false;
            }
        }
    }
    return true;
}
```

```
    // Function to check if the solution
    // of sudoku puzzle is valid or not
    static bool isValidSudoku(int[, ] board)
    {

        // Check if all elements of board[][]
        // stores value in the range[1, 9]
        if (isinRange(board) == false)
        {
            return false;
        }

        // Stores unique value
        // from 1 to N
        bool[] unique = new bool[N + 1];

        // Traverse each row of
        // the given array
        for(int i = 0; i < N; i++)
        {

            // Initialize unique[]
            // array to false
            Array.Fill(unique, false);

            // Traverse each column
            // of current row
            for(int j = 0; j < N; j++)
            {

                // Stores the value
                // of board[i][j]
                int Z = board[i, j];

                // Check if current row
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each column of
        // the given array
        for(int i = 0; i < N; i++)
        {

            // Initialize unique[]
            // array to false
            Array.Fill(unique, false);
            // Traverse each row
            // of current column
            for(int j = 0; j < N; j++)
            {

                // Stores the value
                // of board[j][i]
                int Z = board[j, i];

                // Check if current column
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each block of
        // size 3 * 3 in board[][] array
        for(int i = 0; i < N - 2; i += 3)
        {

            // j stores first column of
            // each 3 * 3 block
            for(int j = 0; j < N - 2; j += 3)
            {

                // Initialize unique[]
                // array to false
                Array.Fill(unique, false);

                // Traverse current block
                for(int k = 0; k < 3; k++)
                {
                    for(int l = 0; l < 3; l++)
                    {

                        // Stores row number
                        // of current block
                        int X = i + k;

                        // Stores column number
                        // of current block
                        int Y = j + l;

                        // Stores the value
                        // of board[X][Y]
                        int Z = board[X, Y];

                        // Check if current block
                        // stores duplicate value
                        if (unique[Z])
                        {
                            return false;
                        }
                        unique[Z] = true;
                    }
                }
            }
        }

        // If all conditions satisfied
        return true;
    }

    // Driver Code
    public static void Main()
    {
        int[,] board = { { 7, 9, 2, 1, 5, 4, 3, 8, 6 },
                          { 6, 4, 3, 8, 2, 7, 1, 5, 9 },
                          { 8, 5, 1, 3, 9, 6, 7, 2, 4 },
                          { 2, 6, 5, 9, 7, 3, 8, 4, 1 },
                          { 4, 8, 9, 5, 6, 1, 2, 7, 3 },
                          { 3, 1, 7, 4, 8, 2, 9, 6, 5 },
                          { 1, 3, 6, 7, 4, 8, 5, 9, 2 },
                          { 9, 7, 4, 2, 1, 5, 6, 3, 8 },
                          { 5, 2, 8, 6, 3, 9, 4, 1, 7 } };

        if (isValidSudoku(board))
        {
            Console.WriteLine("Valid");
        }
        else
        {
            Console.WriteLine("Not Valid");
        }
    }
    }

    // This code is contributed by akhilsaini
```

Javascript

```
// JavaScript program to implement
// the above approach

var N = 9;

// Function to check if all elements
// of the board[][] array store
// value in the range[1, 9]
```

```
function isinRange(board)
{

    // Traverse board[][] array
    for(var i = 0; i < N; i++)
    {
        for(var j = 0; j < N; j++)
        {

            // Check if board[i][j]
            // lies in the range
            if (board[i][j] <= 0 ||
                board[i][j] > 9)
            {
                return false;
            }
        }
    }
    return true;
}

// Function to check if the solution
// of sudoku puzzle is valid or not
function isValidSudoku(board)
{

    // Check if all elements of board[][]
    // stores value in the range[1, 9]
    if (isinRange(board) == false)
    {
        return false;
    }

    // Stores unique value
    // from 1 to N
    var unique = Array(N+1).fill(false);

    // Traverse each row of
    // the given array
    for(var i = 0; i < N; i++)
    {
        unique = Array(N+1).fill(false);

        // Traverse each column
        // of current row
        for(var j = 0; j < N; j++)
        {

            // Stores the value
            // of board[i][j]
            var Z = board[i][j];

            // Check if current row
```

```
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each column of
        // the given array
        for(var i = 0; i < N; i++)
        {

            // Initialize unique[]
            // array to false
            unique = Array(N+1).fill(false);

            // Traverse each row
            // of current column
            for(var j = 0; j < N; j++)
            {

                // Stores the value
                // of board[j][i]
                var Z = board[j][i];

                // Check if current column
                // stores duplicate value
                if (unique[Z])
                {
                    return false;
                }
                unique[Z] = true;
            }
        }

        // Traverse each block of
        // size 3 * 3 in board[][] array
        for(var i = 0; i < N - 2; i += 3)
        {

            // j stores first column of
            // each 3 * 3 block
            for(var j = 0; j < N - 2; j += 3)
            {

                // Initialize unique[]
                // array to false
                unique = Array(N+1).fill(false);

                // Traverse current block
                for(var k = 0; k < 3; k++)
                {
                    for(var l = 0; l < 3; l++)
                    {

                        // Stores row number
                        // of current block
                        var X = i + k;

                        // Stores column number
                        // of current block
                        var Y = j + l;

                        // Stores the value
                        // of board[X][Y]
                        var Z = board[X][Y];

                        // Check if current block
                        // stores duplicate value
                        if (unique[Z])
                        {
                            return false;
                        }
                        unique[Z] = true;
                    }
                }
            }
        }

        // If all conditions satisfied
        return true;
    }

    // Driver Code
    var board = [ [ 7, 9, 2, 1, 5, 4, 3, 8, 6 ],
                      [ 6, 4, 3, 8, 2, 7, 1, 5, 9 ],
                      [ 8, 5, 1, 3, 9, 6, 7, 2, 4 ],
                      [ 2, 6, 5, 9, 7, 3, 8, 4, 1 ],
                      [ 4, 8, 9, 5, 6, 1, 2, 7, 3 ],
                      [ 3, 1, 7, 4, 8, 2, 9, 6, 5 ],
                      [ 1, 3, 6, 7, 4, 8, 5, 9, 2 ],
                      [ 9, 7, 4, 2, 1, 5, 6, 3, 8 ],
                      [ 5, 2, 8, 6, 3, 9, 4, 1, 7 ] ];
    if (isValidSudoku(board))
    {
        document.write("Valid");
    }
    else
    {
        document.write("Not Valid");
    }
```

Output

```
Valid
```

Time Complexity: *$O(N^2)$*

Auxiliary Space: *O(N)*

Approach 2: Using Set:

- In the above approach, a set is used to check if a number is repeated in each row, column, and subgrid of the Sudoku board. A set is a container that stores unique elements in no particular order.
- In this approach, we initialize three sets, one for each row, one for each column, and one for each subgrid. We iterate through each cell in the board and check if its value is not equal to 0. If the value is not 0, we insert it into the corresponding set for its row, column, and subgrid. If the value is already in the set, it means that the value is repeated in that particular row, column, or subgrid, and the Sudoku board is invalid.
- If we iterate through all the cells in the board without finding any duplicates, the Sudoku board is valid. This approach is efficient because it avoids nested loops and only iterates through the board once. It also allows us to quickly check if a number is repeated in each row, column, and subgrid using sets, which have an average constant-time complexity for insertions and lookups.

Here is the Code of above Approach:

C++

```cpp
#include <bits/stdc++.h>
using namespace std;
#define N 9

bool isValidSudoku(int board[][N]) {
    unordered_set<int> rows[N], cols[N], subgrids[N];

    // Traverse board[][] array
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            int num = board[i][j];
            if (num == 0) continue; // Skip empty cells
            int subgrid_index = (i / 3) * 3 + j / 3; // Get sub-grid index

            // Check if num is already present in current row, column or sub-gr
            if (rows[i].count(num) || cols[j].count(num) || subgrids[subgrid_ir
                return false;

            // Add num to the corresponding sets
            rows[i].insert(num);
            cols[j].insert(num);
            subgrids[subgrid_index].insert(num);
        }
    }
    return true;
}

// Driver code
int main() {
    int board[N][N] = {
        {5, 3, 0, 0, 7, 0, 0, 0, 0},
        {6, 0, 0, 1, 9, 5, 0, 0, 0},
        {0, 9, 8, 0, 0, 0, 0, 6, 0},
        {8, 0, 0, 0, 6, 0, 0, 0, 3},
        {4, 0, 0, 8, 0, 3, 0, 0, 1},
        {7, 0, 0, 0, 2, 0, 0, 0, 6},
        {0, 6, 0, 0, 0, 0, 2, 8, 0},
        {0, 0, 0, 4, 1, 9, 0, 0, 5},
        {0, 0, 0, 0, 8, 0, 0, 7, 9}
    };
    if (isValidSudoku(board))
        cout << "Valid" << endl;
    else
        cout << "Invalid" << endl;
    return 0;
}
```

Java

```
import java.util.HashSet;

public class GFG {
    public static boolean isValidSudoku(int[][] board) {
        HashSet<Integer>[] rows = new HashSet[9];
        HashSet<Integer>[] cols = new HashSet[9];
        HashSet<Integer>[] subgrids = new HashSet[9];

        for (int i = 0; i < 9; i++) {
            rows[i] = new HashSet<>();
            cols[i] = new HashSet<>();
            subgrids[i] = new HashSet<>();
        }
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                int num = board[i][j];
                if (num == 0) continue;

                int subgrid_index = (i / 3) * 3 + j / 3;

                if (rows[i].contains(num) || cols[j].contains(num) || subgrids[
                    return false;

                rows[i].add(num);
                cols[j].add(num);
                subgrids[subgrid_index].add(num);
            }
        }
        return true;
    }

    public static void main(String[] args) {
        int[][] board = {
                {5, 3, 0, 0, 7, 0, 0, 0, 0},
                {6, 0, 0, 1, 9, 5, 0, 0, 0},
                {0, 9, 8, 0, 0, 0, 0, 6, 0},
                {8, 0, 0, 0, 6, 0, 0, 0, 3},
                {4, 0, 0, 8, 0, 3, 0, 0, 1},
                {7, 0, 0, 0, 2, 0, 0, 0, 6},
                {0, 6, 0, 0, 0, 0, 2, 8, 0},
                {0, 0, 0, 4, 1, 9, 0, 0, 5},
                {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };

        if (isValidSudoku(board))
            System.out.println("Valid");
        else
            System.out.println("Invalid");
    }
}
```

Python3

```
def is_valid_sudoku(board):
    rows = [set() for _ in range(9)]
    cols = [set() for _ in range(9)]
    subgrids = [set() for _ in range(9)]

    for i in range(9):
        for j in range(9):
            num = board[i][j]
            if num == 0:
                continue

            subgrid_index = (i // 3) * 3 + j // 3

            if num in rows[i] or num in cols[j] or num in subgrids[subgrid_inc
                return False

            rows[i].add(num)
            cols[j].add(num)
            subgrids[subgrid_index].add(num)

    return True

board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]

if is_valid_sudoku(board):
    print("Valid")
else:
    print("Invalid")
```

C#

```
using System;
using System.Collections.Generic;

public class GFG {
    public static bool IsValidSudoku(int[][] board) {
        HashSet<int>[] rows = new HashSet<int>[9];
        HashSet<int>[] cols = new HashSet<int>[9];
        HashSet<int>[] subgrids = new HashSet<int>[9];

        for (int i = 0; i < 9; i++) {
            rows[i] = new HashSet<int>();
            cols[i] = new HashSet<int>();
            subgrids[i] = new HashSet<int>();
        }

        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                int num = board[i][j];
                if (num == 0) continue;

                int subgrid_index = (i / 3) * 3 + j / 3;

                if (rows[i].Contains(num) || cols[j].Contains(num) || subgrids[
                    return false;

                rows[i].Add(num);
                cols[j].Add(num);
                subgrids[subgrid_index].Add(num);
            }
        }
        return true;
    }

    public static void Main(string[] args) {
        int[][] board = new int[9][] {
            new int[] {5, 3, 0, 0, 7, 0, 0, 0, 0},
            new int[] {6, 0, 0, 1, 9, 5, 0, 0, 0},
            new int[] {0, 9, 8, 0, 0, 0, 0, 6, 0},
            new int[] {8, 0, 0, 0, 6, 0, 0, 0, 3},
            new int[] {4, 0, 0, 8, 0, 3, 0, 0, 1},
            new int[] {7, 0, 0, 0, 2, 0, 0, 0, 6},
            new int[] {0, 6, 0, 0, 0, 0, 2, 8, 0},
            new int[] {0, 0, 0, 4, 1, 9, 0, 0, 5},
            new int[] {0, 0, 0, 0, 8, 0, 0, 7, 9}
        };

        if (IsValidSudoku(board))
            Console.WriteLine("Valid");
        else
            Console.WriteLine("Invalid");
    }
}
```

Javascript

```
function isValidSudoku(board) {
    const rows = new Array(9).fill(null).map(() => new Set());
    const cols = new Array(9).fill(null).map(() => new Set());
    const subgrids = new Array(9).fill(null).map(() => new Set());

    for (let i = 0; i < 9; i++) {
        for (let j = 0; j < 9; j++) {
            const num = board[i][j];
            if (num === 0) continue;

            const subgridIndex = Math.floor(i / 3) * 3 + Math.floor(j / 3);

            if (rows[i].has(num) || cols[j].has(num) || subgrids[subgridIndex].
                return false;
            }

            rows[i].add(num);
            cols[j].add(num);
            subgrids[subgridIndex].add(num);
        }
    }
    return true;
}

const board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
];

if (isValidSudoku(board)) {
    console.log("Valid");
} else {
    console.log("Invalid");
}
```

Output

```
Valid
```

Time Complexity: O(N2)

Auxiliary Space: O(N)

- DSA in Python
- DSA in JavaScript

Last Updated : 11 Aug, 2023

5

Recommended Problem

Is Sudoku Valid

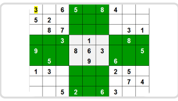
Matrix Data Structures Amazon Microsoft +1 more

Solve Problem

Submission count: 22.9K

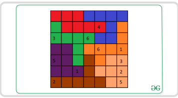
Similar Reads

Check if given Sudoku board configuration is valid or not

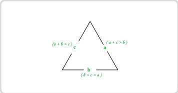

Algorithm to Solve Sudoku | Sudoku Solver

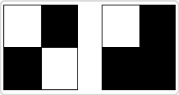
Find a valid parenthesis sequence of length K from a given valid parenthesis…

Find initial integral solution of Linear Diophantine equation if finite solution…


Solve Sudoku on the basis of the given irregular regions

Check whether a given Binary Tree is Complete or not | Set 1 (Iterative…

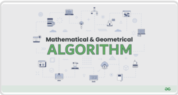

Check whether triangle is valid or not if sides are given


Check if the given chessboard is valid or not

Check if the given push and pop sequences of Stack is valid or not

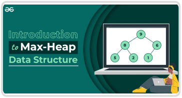
Check if the given Prufer sequence is valid or not

Related Tutorials

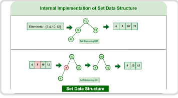


Mathematical and Geometric Algorithms - Data Structure and…

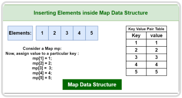
Learn Data Structures with Javascript | DSA Tutorial



Introduction to Max-Heap – Data Structure and Algorithm Tutorials



Introduction to Set – Data Structure and Algorithm Tutorials



Introduction to Map – Data Structure and Algorithm Tutorials

Previous Next

Create matrix whose sum of diagonals in each sub matrix is even

Nearest smaller number to N having multiplicative inverse under modulo N equal to that number

Article Contributed By :

pravallika26
pravallika26

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : akhilsaini, noob2000, gabaa406, karan_kumar19, yashdkadam

Article Tags : array-traversal-question , frequency-counting , submatrix , DSA , Greedy , Matrix , Pattern Searching , Searching

Practice Tags : Greedy, Matrix, Pattern Searching, Searching

Improve Article Report Issue

GeeksforGeeks
Sanchhaya Education Private Limited
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

GET IT ON Google Play Download on the App Store

Company

About Us

Legal

Careers

In Media

Contact Us

Advertise with us

GFG Corporate Solution

Placement Training Program

Apply for Mentor

Explore

Job-A-Thon Hiring Challenge

Hack-A-Thon

GfG Weekly Contest

Offline Classes (Delhi/NCR)

DSA in JAVA/C++

Master System Design

Master CP

GeeksforGeeks Videos

Languages

Python

Java

C++

PHP

GoLang

SQL

R Language

Android Tutorial

DSA Concepts

Data Structures

Arrays

Strings

Linked List

Algorithms

Searching

Sorting

Mathematical

Dynamic Programming

DSA Roadmaps

DSA for Beginners

Basic DSA Coding Problems

DSA Roadmap by Sandeep Jain

DSA with JavaScript

Top 100 DSA Interview Problems

All Cheat Sheets

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Express.js

Lodash

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top 15 Websites for CP

Interview Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

Commerce

Accountancy

Business Studies

Economics

Human Resource Management (HRM)

Management

Income Tax

Finance

Statistics for Economics

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

Top SD Interview Questions

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships