**Solve the Sudoku :-**

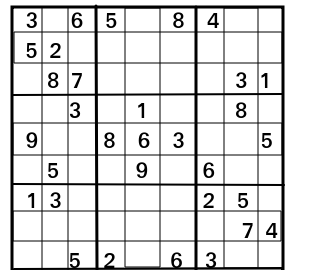
Hard Accuracy: 37.98% Submissions: 74K+ Points: 8

Given an incomplete [**Sudoku**](https://en.wikipedia.org/wiki/Sudoku)configuration in terms of a 9 x 9  2-D square matrix (grid[][]), the task is to find a solved Sudoku. For simplicity, you may assume that there will be only one unique solution.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits 1-9 must occur exactly once in each row.
2. Each of the digits 1-9 must occur exactly once in each column.
3. Each of the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

Zeros in the grid indicates blanks, which are to be filled with some number between 1-9. You can not replace the element in the cell which is not blank.

**Sample Sudoku for you to get the logic for its solution:**  
  


**Example 1:**

**Input:**

grid[][] =

[[3 0 6 5 0 8 4 0 0],

[5 2 0 0 0 0 0 0 0],

[0 8 7 0 0 0 0 3 1 ],

[0 0 3 0 1 0 0 8 0],

[9 0 0 8 6 3 0 0 5],

[0 5 0 0 9 0 6 0 0],

[1 3 0 0 0 0 2 5 0],

[0 0 0 0 0 0 0 7 4],

[0 0 5 2 0 6 3 0 0]]

**Output:**True

3 1 6 5 7 8 4 9 2

5 2 9 1 3 4 7 6 8

4 8 7 6 2 9 5 3 1

2 6 3 4 1 5 9 8 7

9 7 4 8 6 3 1 2 5

8 5 1 7 9 2 6 4 3

1 3 8 9 4 7 2 5 6

6 9 2 3 5 1 8 7 4

7 4 5 2 8 6 3 1 9  
**Explanation:**   
There exists a valid Sudoku for the input grid, which is shown in output.

**Example 2:**

**Input:**

grid[][] =

[[3 6 6 5 0 8 4 0 0],

[5 2 0 0 0 0 0 0 0],

[0 8 7 0 0 0 0 3 1 ],

[0 0 3 0 1 0 0 8 0],

[9 0 0 8 6 3 0 0 5],

[0 5 0 0 9 0 6 0 0],

[1 3 0 0 0 0 2 5 0],

[0 0 0 0 0 0 0 7 4],

[0 0 5 2 0 6 3 0 0]]

**Output:**False  
**Explanation:**   
There does not exists a valid Sudoku for the input grid, since there are two 6s in the first row. Which cannot replaced.

**Your Task:**  
You need to complete the two functions:  
  
**SolveSudoku()**: Takes a grid as its argument and returns true if a solution is possible and false if it is not, on returning **false**driver will print **"No solution exists"**.  
  
**printGrid()**: Takes a grid as its argument and prints the 81 numbers of the solved Sudoku in a single line with space separation. Do not give a new line character after printing the grid.

**Expected Time Complexity:** O(9N\*N).  
**Expected Auxiliary Space:** O(N\*N).

**Constraints:**  
0 ≤ grid[i][j] ≤ 9

**Code :-**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// UNASSIGNED is used for empty cells in sudoku grid

#define UNASSIGNED 0

// N is used for the size of Sudoku grid.

// Size will be NxN

#define N 9

// } Driver Code Ends

class Solution

{

public:

bool put(int grid[N][N], int row, int col, int choice){

for(auto i=0; i<N; i++){

//row check

if(grid[row][i] == choice)

return false;

//column check

if(grid[i][col] == choice)

return false;

//own square check

int x = 3\*(row/3) + i/3;

int y = 3\*(col/3) + i%3;

if(grid[x][y] == choice)

return false;

}

return true;

}

//Function to find a solved Sudoku.

bool SolveSudoku(int grid[N][N]){

for(auto row=0; row<N; row++){

for(auto col=0; col<N; col++){

//empty cell found

if(grid[row][col] == 0){

//choose 1 to 9 for that empty cell

for(auto choice=1; choice<=N; choice++){

//place the choice if it is valid

if(put(grid, row, col, choice) == true){

grid[row][col] = choice;

//recursive call to solve rest empty cell

if(SolveSudoku(grid) == true)

return true;

//backtrack

grid[row][col] = 0;

}

}

//already choosen 1 to 9 but no-one fits

return false;

}

}

}

return true;

}

//Function to print grids of the Sudoku.

void printGrid (int grid[N][N]){

for(auto i=0; i<N; i++){

for(auto j=0; j<N; j++)

cout << grid[i][j] << " ";

}

return;

}

};

//{ Driver Code Starts.

int main() {

int t;

cin>>t;

while(t--)

{ int grid[N][N];

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

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

cin>>grid[i][j];

Solution ob;

if (ob.SolveSudoku(grid) == true)

ob.printGrid(grid);

else

cout << "No solution exists";

cout<<endl;

}

return 0;

}

// } Driver Code Ends