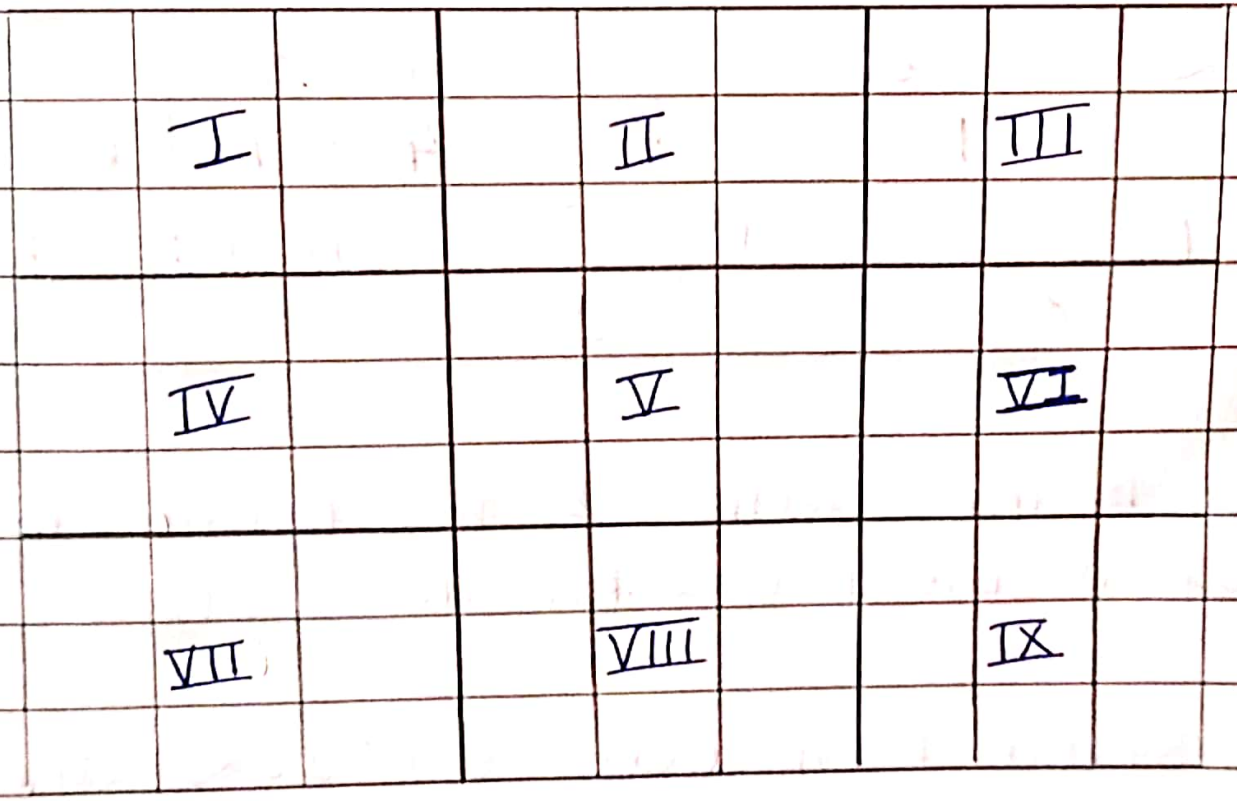
26/03/2023

Sudoku Solver

There will be a 9x9 matrix.



Scanned with Cam

* In each column there should be numbers from 1 to 9 & should not be repeating.
* In each row there should be numbers from 1 to 9 and should not be repeating
* There will be 9, 3×3 boxes & each box should have numbers from 1 to 9 & there should be no repetition. 9 boxes each of 3×3 have been marked.

All the above 3 conditions are mandatory conditions.

Note→ If there are many clues, then less chances of multiple solutions. If there are less clues, then high chance of multiple solution. The exact number of clues is 17 for which unique solution will exist. ↳ or above

	0	1	2	3	4	5	6	7	8
0	4	5							
1			2		7		6	3	
2								2	8
3				9	5				
4		8	6				2		
5		2		6			7	5	
6							4	7	6
7		7			4	5			
8			8			9			

The above is the question & the numbers which are written in black are known as clues.

We will start from cell s [0][2] and apply

Scanned with CamScanner

isSafe function to place numbers from 1 to 9 starting from 1. In the isSafe function we need to check the 3 conditions which we have already discussed. If a particular number isSafe, simply insert it & move ahead. Suppose we are not able to insert any number & this means that there is fault in previous placement just like n-queen problem. Here backtracking plays an important role. This is the only logic we need to apply until we fill all the cells.

Code

```
bool isSafe (int value, int board[][9],
             int curr_row, int curr_col) {
 // Row check → Same row, changing column
 for (int col = 0; col < 9; col++) {
      if (board[curr_row][col] == value)
          return false;
 }
 // Column check → Same column, changing row
 for (int row = 0; row < 9; row++) {
      if (board[row][curr_col] == value)
          return false;
 }
 //3 X 3 box checking
 for (int i = 0; i < 9; i++) {
   if (board[3*(curr_row/3) + i/3][3*(curr_col/
       3) + i%3] == value) {
          return false;
      }
 }
 return true; // Safe value
}
```

Scanned with CamScanner

```
// Tweek done → return type bool to avoid base case
bool solve (int board[][9], int n) {
  // Traversing the sudoku board
  for (int i=0; i< n ; i++) {
     for (int j=0; j<n; j++) {
          // Insert only in empty cell
          if (board[i][j] == 0) {
1 to 9 value ← for (int value = 1; value <=9; value++){
Check for safety ← if (isSafe (value, board, i, j)) {
If safe, insert ← board[i][j] = value;
                  // Recursive call
                  bool aage Solution = solve (board, n);
                  // Found solution or not ?
                  // Due to below line, we don't need base case
                  if (aage Solution) {
Don't explore further ← return true; // Found soln
soln as in question }
it says unique          // Backtracking
soln is there.          board[i][j] = 0;  ↗ empty cell
                  }
               }
               return false; // If no value from 1 to 9
     }                        can be inserted.
  }
}
  return true; // All cells are filled & hence return
}                 true

In the main function simply print the
sudoku board as the values will be
filled.
```

Scanned with CamScanner

Understanding 3×3 box check condition

* i = 0 to 8 → sent in isSafe ←

board [3 * (curr_row/3) + i/3] [3 * (curr_col/3) + i % 3]

	0	1	2
0	4	5	
1			2
2			

→ (curr_row, curr_col) → (0, 2)

* i = 0

board [3 * (0/3) + 0/3] [3 * (2/3) + 0 % 3]

board [0 + 0] [3 * (0) + 0] ⇒ board [0] [0]

* i = 1

board [3 * (0/3) + 1/3] [3 * (2/3) + 1 % 3]

board [0] [0 + 1] ⇒ board [0] [1]

* i = 2

board [3 * (0/3) + 2/3] [3 * (2/3) + 2 % 3]

board [0 + 0] [0 + 2] ⇒ board [0] [2]

* i = 3

board [3 * (0/3) + 3/3] [3 * (2/3) + 3 % 3]

board [0 + 1] [0 + 0] ⇒ board [1] [0]

* i = 4

board [3 * (0/3) + 4/3] [3 * (2/3) + 4 % 3]

board [0 + 1] [0 + 1] ⇒ board [1] [1]

* i = 5

board [3 * (0/3) + 5/3] [3 * (2/3) + 5 % 3]

board [0 + 1] [0 + 2] ⇒ board [1] [2]

* i = 6

board [3 * (0/3) + 6/3] [3 * (2/3) + 6 % 3]

board [0 + 2] [0 + 0] ⇒ board [2] [0]

* i = 7

board [3 * (0/3) + 7/3] [3 * (2/3) + 7 % 3]

board [0 + 2] [0 + 1] ⇒ board [2] [1]

Scanned with CamScanner

* $i = 8$

board [3 * (0/3) + 8/3] [3 * (2/3) + 8%3]

board [0+2] [0 +2] ⇒ board[2][2]

Note → 3 * (curr_row/3) → Starting row of each box

i/3 → movement in down direxn

3 * (curr_col/3) → starting col of each box

i%3 → movement in right direxn

Scanned with Cam