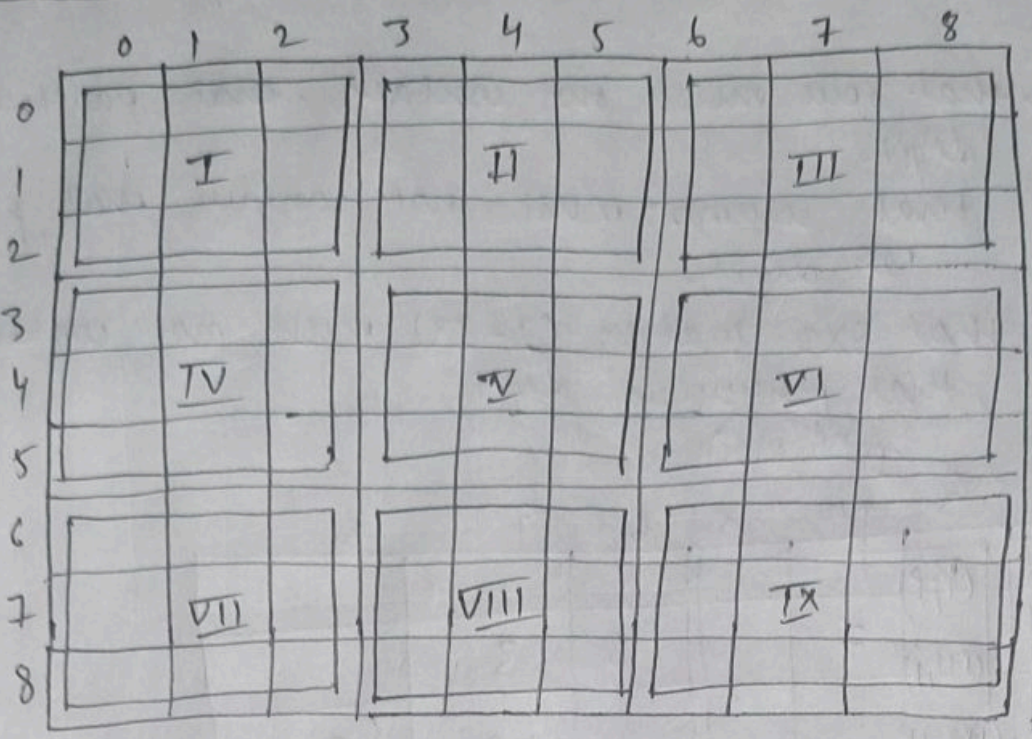
* Sudoku Solver [Imp ques]



→ We have given 9×9 matrix

→ We need to place digits (1-9) in each cell i.e such that

① row ② column ③ 3×3 Box matrix (i.e I to IX) such that no number is repeated.

Inputs: We are given by 9×9 matrix along with some inputs.

Board	0	1	2 (1-9)	3 (1-9)	4	5	6	7	8
0	4	5	1						
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			

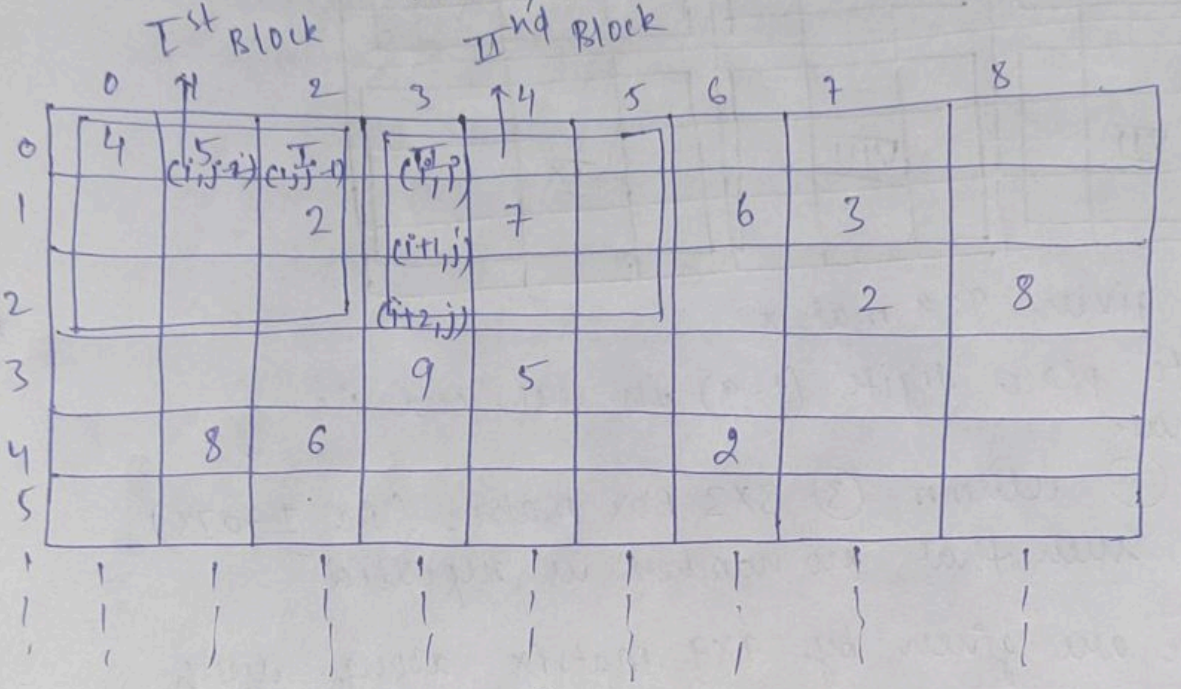
→ 1 is not present at that row, column & 3×3 matrix

isSafe()
- → row
- → column
- → 3×3 matrix

These are the inputs given to us

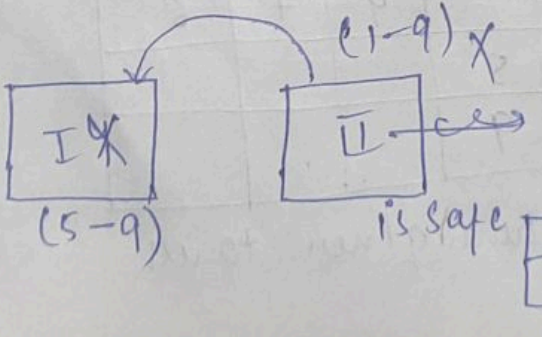
We will try to fill each cell with digits 1-9 by checking is placing the particular digit is safe or not.

is Safe → that row must not contain that particular digit.
→ that column must not contain that particular digit.
→ that 3×3 matrix (I-IX) must not contain that particular digit.



Goes on

We will try to place (1-9) digits in the cell I. Now we will check whether placing that 1 is safe or not. Now we can place 1 as 1 is not present at 0 row, 2 column and I st Block so we can place 1 in it.
Next we will try to place (1-9) digits in the cell II so we can't place 1 as 1 is not present at 0 row, so we go with 2 can be place as it is not present at 0 row & 3 column and II nd Block. similar goes on till last column...



when we are unable to place any digits from @ 1-9 in cell II then we can say that the digit place in cell I was wrong.

code

```
bool issafe(vector<vector<int>> &board, int value, int row,
            int col){
    for(int i=0; i<board.size(); i++){
        if(board[row][i] == value){
            return false;
        }
        if(board[i][col] == value){
            return false;
        }
        if(board[3*(row/3)+(i/3)][3*(col/3)+(i%3)] == value)
        {
            return false;
        }
    }
    return true;
}
bool sudokuSolver(vector<vector<int>> &board){
    for(int i=0; i<board.size(); i++){
        for(int j=0; j<board[i].size(); j++){
            if(board[i][j] == 0){
                for(int value = 1; value <= 9; value++)
                {
                    board[i][j] = value;
                    bool remainingSol = sudokuSolve(board);
                    if(remainingSol){
                        return true;
                    }
                    else{
                        board[i][j] = 0;
                    }
                }
            }
        }
    }
```

```
            }
            return false;
        }
    }
    return true;
}

int main() {
    vector<vector<int>> board = {
        {5,3,0,0,7,0,0,0,0},
        {6,0,0,1,9,5,3,4,0},
        {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},
        {0,0,0,0,8,0,0,7,9}
    };
    if (sudokuSolver(board)) {
        for (int i=0; i<board.size(); i++) {
            for (int j=0; j<board[i].size(); j++) {
                cout << board[i][j] << " ";
            }
            cout << endl;
        }
    }
}
```

output

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