



+20



Normal text Arial - 11 + B I U A G S E H X

[ONnb1wdx2Ma&index=51](#)

6. K-th permutation Sequence

https://www.youtube.com/watch?v=wT7gcXLYao&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=55

Day10: (Recursion and Backtracking)

1. Print all Permutations of a string/array

https://www.youtube.com/watch?v=f2ic2Rsc9pU&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=52

2. N queens Problem

https://www.youtube.com/watch?v=i05Ju7AftcM&list=PLgUwDviBIf0p4ozDR_kJJkONnb1wdx2Ma&index=57

3. **Sudoku Solver**

4. M coloring Problem (Graph prob)

5. Rat in a Maze

6. Word Break (print all ways)

Day11: (Divide and Conquer)

1. 1/N-th root of an integer (use binary search) (square root, cube root, ..)



TUF



0:04 / 26:09



37. Sudoku Solver

Hard 2779 106 Add to List Share

Write a program to solve a Sudoku puzzle by filling the empty cells.

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.

The '.' character indicates empty cells.

Example 1:

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	.	.	.

Input: board = [[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: [[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"]]

TUF

Example 1:

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		

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

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"]]`

Explanation: The input board is shown above and the only valid solution is shown below:

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

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

9 × 9.

9 (3 ×

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

$$\underline{9 \times 9}.$$

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

$$(2) \quad m \quad n \quad r \quad \text{in} \quad \xrightarrow{\text{avg}} \begin{matrix} 3 \\ 3 \end{matrix}$$

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

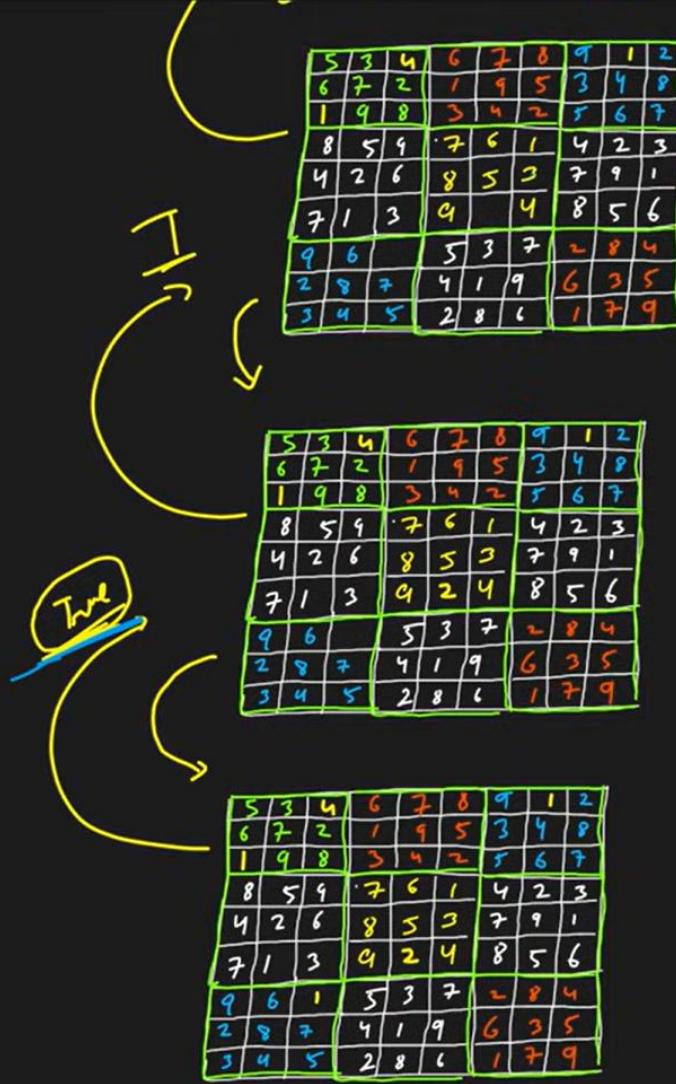
$\frac{1}{x} \rightarrow 0$

γ (possible)
 $f(l)$

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

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

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



```

C++    • Autocomplete

1+ class Solution {
2+ public:
3+     void solveSudoku(vector<vector<char>>& board) {
4+         solve(board);
5+     }
6+
7+     bool solve(vector<vector<char>>& board){
8+         for(int i = 0; i < board.size(); i++) {
9+             for(int j = 0; j < board[0].size(); j++) {
0+
1+                 if(board[i][j] == '.'){
2+
3+                     for(char c = '1'; c <= '9'; c++) {
4+                         if(isValid(board, i, j, c)){
5+                             board[i][j] = c;
6+
7+                             if(solve(board) == true)
8+                                 return true;
9+                             else
0+                                 board[i][j] = '.';
1+
2+                         }
3+
4+                     }
5+
6+                     return false;
7+                 }
8+
9+             }
0+
1+         }
2+
3+         return true;
4+     }
5+
6+     bool isValid(vector<vector<char>>& board, int row, int col, char c){
7+         for(int i = 0; i < 9; i++) {
8+             if(board[i][col] == c)
9+                 return false;
0+
1+             if(board[row][i] == c)
2+                 return false;
3+
4+             if(board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
5+                 return false;
6+
7+         }
8+
9+         return true;
0+
1+     }
2+
3+ };
4+

```

our previous code was restored from your local storage. [Reset to default](#)

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

(5,7)

for(i=0 → 8)

if board[[5] [i] == 1) return false

if board[i][7] == 1) return false

row = $\left\lceil 3 \times \left(\frac{5}{3} \right) \right\rceil = 3 + \frac{2}{3} = 3$

col = $\left\lceil 3 \times \left(\frac{7}{3} \right) \right\rceil = 6 + \frac{1}{3} = 6$

```
C++ └─ Autocomplete
1 v class Solution {
2 v   public:
3 v     void solveSudoku(vector<vector<char>>& board) {
4 v       solve(board);
5 v     }
6 v
7 v     bool solve(vector<vector<char>>& board){
8 v       for(int i = 0; i < board.size(); i++) {
9 v         for(int j = 0; j < board[0].size(); j++) {
10 v
11 v           if(board[i][j] == '.'){
12 v
13 v             for(char c = '1'; c <= '9'; c++){
14 v               if(isValid(board, i, j, c)){
15 v                 board[i][j] = c;
16 v
17 v                 if(solve(board) == true)
18 v                   return true;
19 v                 else
20 v                   board[i][j] = '.';
21 v               }
22 v             }
23 v           }
24 v         }
25 v       }
26 v       return false;
27 v     }
28 v
29 v     bool isValid(vector<vector<char>>& board, int row, int col, char c){
30 v       for(int i = 0; i < 9; i++) {
31 v         if(board[i][col] == c)
32 v           return false;
33 v
34 v         if(board[row][i] == c)
35 v           return false;
36 v
37 v         if(board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c)
38 v           return false;
39 v       }
40 v       return true;
41 v     }
42 v
43 v   };
44 v }
```

our previous code was restored from your local storage. [Reset to default](#)

Console - Contribute ↗