Sudoku Puzzle Solver using Backtracking Algorithm

Min H. Kao Department of Electrical Engineering and Computer Science

Python Assignment from CS-140 Lecture notes

University of Tennessee, (Knoxville) US

University of Karachi (Mathematics Department)

Course Software Engineering (663) MSc Mathematics

Engineer Syed Umaid Ahmed

BE (EE), ME (Mechatronics)

NED University of Engineering & Technology

What is Sudoku ?

- Sudoku requires no calculation or arithmetic skills.
- It is essentially a game of placing numbers in squares.
- It uses very simple rules of logic and deduction.
- It can be played by children and adults and the rules are simple to learn.
- Objective of game is to fill all the blank squares in a game with the correct numbers.

	a	b	c	d	e	f	g	h	i
A			8			6		4	7
B	4					1		3	8
C					7			6	5
D	6		3			7	5		
E	7		1				6		3
F							7		4
G	8				9		3		
H	5	3		2			8		9
I				7			4		

Simple Rules of Sudoku Solving

There are three very simple constraints to follow:

- ✓ Every row of 9 numbers must include all digits 1 through 9 in any order
- ✓ Every column of 9 numbers must include all digits 1 through 9 in any order
- ✓ Every 3 by 3 subsection of the 9 by 9 square must include all digits 1 through 9

Online Material of Sudoku Puzzle

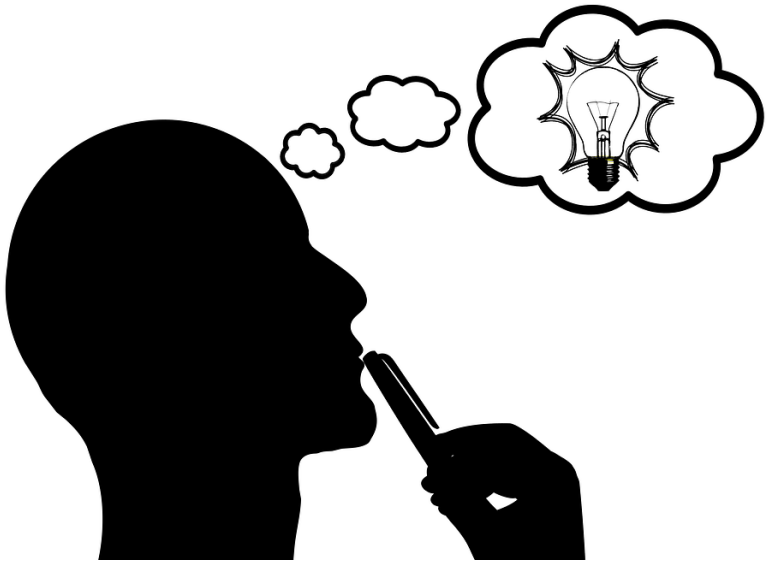
Here are a few Sudoku Websites that you can practice some more:

- https://www.thesudoku.com/
- www.sudokufun.com
- www.dailymail.co.uk/coffeebreak/puzzles/sudoku.html

Manual Solution of Sudoku Puzzle

First of all, we will solve sudoku puzzle manually using backtracking algorithm.

Let’s start with the worksheet provided to individual students



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

Python Solution of Sudoku Puzzle (Step # 1)

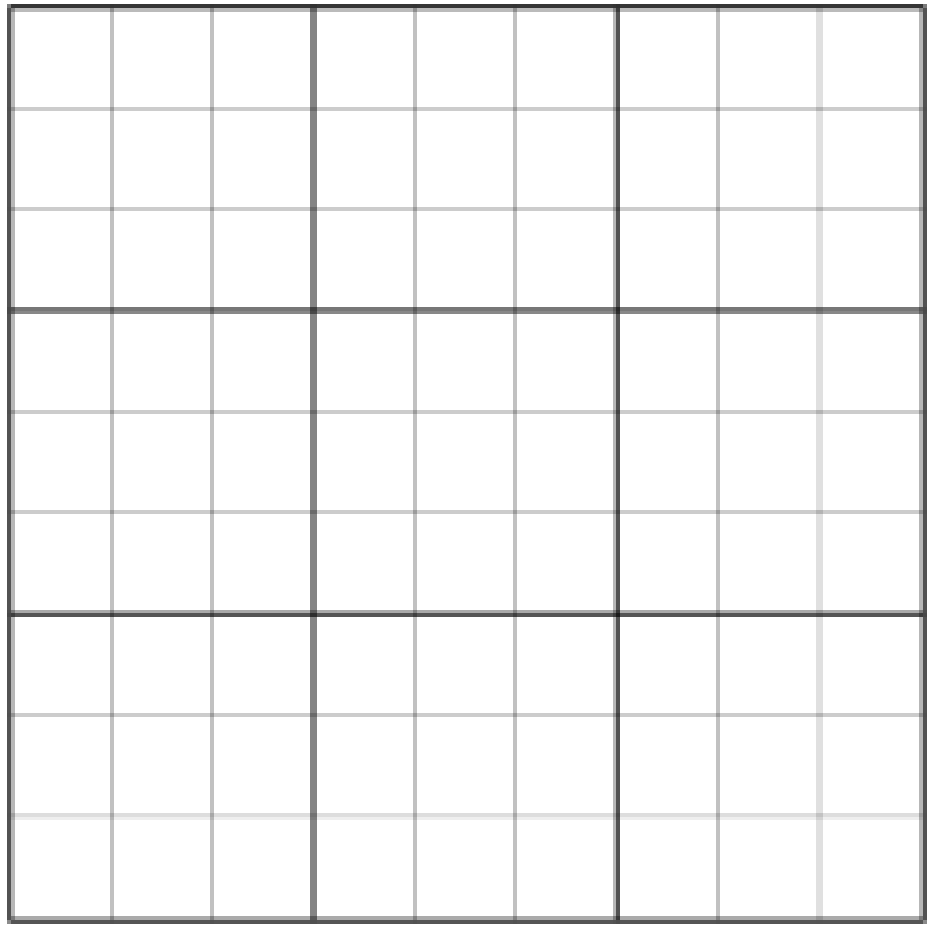
<u>The first step is to put the manual sudoku in Python lists, For empty spaces write zero</u>

```
# This is the Sudoku Puzzle It consists of 9 lists inside
# Always Remember the
#"ONE COMPLETE LIST [] IS ONE ELEMENT"
# So there are 9 elements in total, IF you consider only one list
# THERE are also 9 elements

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

Python Solution of Sudoku Puzzle (Step # 2a)

<u>The second step is to make function for printing board “like this below” with lines seperating each BOX, ROW and COLUMNS</u>



```
7 8 0 | 4 0 0 | 1 2 0
6 0 0 | 0 7 5 | 0 0 9
0 0 0 | 6 0 1 | 0 7 8
- - - - - - - - - - -
0 0 7 | 0 4 0 | 2 6 0
0 0 1 | 0 5 0 | 9 3 0
9 0 4 | 0 6 0 | 0 0 5
- - - - - - - - - - -
0 7 0 | 3 0 0 | 0 1 2
1 2 0 | 0 0 7 | 4 0 0
0 4 9 | 2 0 6 | 0 0 7
```

Python Solution of Sudoku Puzzle (Step # 2b)

```
def print_board(bo):  #FUNCTION

# Each list (one square bracket) count the row number
# That's why used "len(bo)"

    for i in range(len(bo)):   ##Because see the ROW number

        if i%3==0 and i!=0:
            print("- - - - - - - - - - - -")

# But in J we used len(bo[0])
# means that we are using column , See column marks inside the list 1st Element

        for j in range(len(bo[0])):
            if j%3==0 and j!=0:
                print("| ", end="")

# Change line after 8 elements

            if j==8:
                print(bo[i][j])

#Print space after each element, if not on last entry
#Don't spare the line

            else:
                print(str(bo[i][j])+" ",end="")
```

Python Solution of Sudoku Puzzle (Step # 3)

<u>The third step is to make function for finding positions of empty values in ROW and COLUMNS</u>

```
def find_empty(bo):
    for i in range(len(bo)):  # Same we write for ROW number
        for j in range(len(bo[0])):   # Same we write for column number
            if bo[i][j]==0:
                return(i,j)
    return None
```

```
7 8 0 | 4 0 0 | 1 2 0
6 0 0 | 0 7 5 | 0 0 9
0 0 0 | 6 0 1 | 0 7 8
- - - - - - - - - - -
0 0 7 | 0 4 0 | 2 6 0
0 0 1 | 0 5 0 | 9 3 0
9 0 4 | 0 6 0 | 0 0 5
- - - - - - - - - - -
0 7 0 | 3 0 0 | 0 1 2
1 2 0 | 0 0 7 | 4 0 0
0 4 9 | 2 0 6 | 0 0 7
```

(3,0)
Tuple

(5,6)
Tuple

(7,2)
Tuple

These all will be later passed to a function
Having a parameter names **"pos"**

FIND_EMPTY FUNCTION

***All Locations of Empty Boxes**

(0 , 2) (0 , 4) (0 , 5) (0 , 8)

(1 , 1) (1 , 2) (1 , 3) (1 , 6) (1 , 7)

(2 , 0) (2 , 1) (2 , 2) (2 , 4) (2 , 6)

(3 , 0) (3 , 1) (3 , 3) (3 , 5) (3 , 8)

(4 , 0) (4 , 1) (4 , 3) (4 , 5) (4 , 8)

(5 , 1) (5 , 3) (5 , 5) (5 , 6) (5 , 7)

(6 , 0) (6 , 2) (6 , 4) (6 , 5) (6 , 6)

(7 , 2) (7 , 3) (7 , 4) (7 , 7) (7 , 8)

(8 , 0) (8 , 4) (8 , 6) (8 , 7)

Python Solution of Sudoku Puzzle (Step # 4a)

The fourth (a) step is to make function "valid" with 3 arguments for checking valid entries

Before starting the work, first learn what are these "THREE" arguments

```
def valid(bo,num,pos):
```

This is nothing but only the "Print Board Function" passed

This is simple numbers list "Passed from 1 to 9" in next step

This is tuple returned from "find_empty" function

Python Solution of Sudoku Puzzle (Step # 4b)

<u>The fourth (b) step is to checking entries by row and column (One by One)</u>
<u>Also, we are making sure that it is not the same checked before !</u>

```
def valid(bo,num,pos):

# CHECK ROW AND COLUMN IN 2 FOR LOOPS

    for i in range(len(bo)):
        if bo[pos[0]][i]==num:
            return False

    for i in range(len(bo[0])):
        if bo[i][pos[1]]==num:
            return False

    # CHECK BOX
    box_x = pos[0]//3
    box_y = pos[1]//3

    for i in range(box_x*3, box_x*3 + 3):
        for j in range(box_y*3, box_y*3 + 3):
            if bo[i][j]==num:
                return False

    return True
```

Python Solution of Sudoku Puzzle (Step # 4c)

<u>The fourth (c) step is to move technically find the "box number and box term" Using the "FLOOR DIVISION (//) and MULTIPLICATION TECHNIQUE"</u>

```python
def valid(bo,num,pos):

# CHECK ROW AND COLUMN IN 2 FOR LOOPS

    for i in range(len(bo)):
        if bo[pos[0]][i]==num:
            return False

    for i in range(len(bo[0])):
        if bo[i][pos[1]]==num:
            return False

    # CHECK BOX
    box_x = pos[0]//3
    box_y = pos[1]//3

    for i in range(box_x*3, box_x*3 + 3):
        for j in range(box_y*3, box_y*3 + 3):
            if bo[i][j]==num:
                return False

    return True
```

Box No (0,0)

Box No(0,2)

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

Box No (2,1)

box_x = pos[0] // 3	box_y = pos[1] // 3	
0 // 3 = 0	2 // 3 = 0	→ Box No (0,0)
1 // 3 = 0	4 // 3 = 1	
2 // 3 = 0	5 // 3 = 1	
3 // 3 = 1	8 // 3 = 2	→ Box No (1,2)
4 // 3 = 1	1 // 3 = 0	
5 // 3 = 1	3 // 3 = 1	
6 // 3 = 2	6 // 3 = 2	
7 // 3 = 2	7 // 3 = 2	
8 // 3 = 2	0 // 3 = 0	→ Box No (2,0)

(0 , 2) (0 , 4) (0 , 5) (0 , 8)

(1 , 1) (1 , 2) (1 , 3) (1 , 6) (1 , 7)

(2 , 0) (2 , 1) (2 , 2) (2 , 4) (2 , 6)

(3 , 0) (3 , 1) (3 , 3) (3 , 5) (3 , 8)

(4 , 0) (4 , 1) (4 , 3) (4 , 5) (4 , 8)

(5 , 1) (5 , 3) (5 , 5) (5 , 6) (5 , 7)

(6 , 0) (6 , 2) (6 , 4) (6 , 5) (6 , 6)

(7 , 2) (7 , 3) (7 , 4) (7 , 7) (7 , 8)

(8 , 0) (8 , 4) (8 , 6) (8 , 7)

Pos[0] is the “x” element

Pos[1] is the “y” element

Python Solution of Sudoku Puzzle (Step # 5)

<u>The final step is to just use the solve function managing all the backtracking Tasks with the recursive function</u>

```
def solve(bo):

    find = find_empty(bo)

    if not find:
        return True
    else:
        row,col = find


    for i in range(1,10):
        if valid(bo,i,(row,col)):
            bo[row][col]=i

            if solve(bo):
                return True

            bo[row][col] = 0

    return False
```