Fundamentals of Sudoku Backtracking

An algorithm that provides the basis of solving Sudoku

Priyanshu Mehta
Student
Class XII(CBSE),
Jain International School Aurangabad, Aurangabad, India

Abstract : In this discussion , we attempt to provide a generalized solution for solving the popular game of Sudoku .We will lay out various different parameters that should be kept in mind for solving any size of Sudoku. For simplicity , we will take use the 3*3 sudoku or the 9*9 sudoku for the explanation .We will describe various methods of breaking down the game in smaller parts and then solving them .The language used in these explanations is Python ,however, it is not limited to only this language .We will primarily focus on the mathematical side of the solution and not the cosmetic part .

IndexTerms **– Sudoku, Backtracking, Algorithm**

I. INTRODUCTION

[4]For Sudoku is a logical and mathematical puzzle with numbers. In classic Sudoku, the goal is to fill a 9 × 9 grid with numbers so that each column, row, and nine 3 × 3 sub-grids (also called boxes) that make up the grid contain values from 1 to 9. Puzzle Creator provides a partially complete grid, of which only one solution can be used for well-placed puzzles. Sudoku is entirely logic based, with no arithmetic operations, and the level of difficulty is determined by the number and position of the original numbers.T here are larger puzzles with 16x16 or 25x25 grids, puzzles consisting of interlocking Sudoku grids, and 3D variants in the form of 3x3x3 cubes .A Sudoku grid is a special type of Latin square. Latin squares, after which Cell by the mathematician Leonhard Euler from the 18th examples are shown. The finished standard Sudoku Grid (also known as a solution grid) is a 9 × 9 Latin square that has the additional restriction that each of its nine subgrids contains the digits 1 through 9.

1.1Basic Math Behind Sudoku

[5]The standard version of Sudoku consists of a 9 × 9 grid with 81 cells. The grid is divided into 9 3 × 3 blocks, some of the 81 cells are filled with numbers in the set {1,2,3,4,5,6,7,8,9}. These filled cells are called cubes. . The goal is to fill the entire grid with all nine digits so that every row, every column, and every block contains every digit exactly once. This restriction on rows, columns, and blocks is a standard rule. The above puzzle is called Rank 3 Sudoku. Rank n Sudoku is an n2×n2 square, and each block is divided into n2 n×n blocks. The numbers that fill the grid are 1, 2, 3,..., n2, the only rule still applies.

1.2 Constraints of Sudoku Grid

1. Any column cannot contain the same number more than once.
2. Any row cannot contain the same number more than once
3. Any box or submatrix cannot have the same number more than once.

II. TERMS IN USE

1. Size :Size refers to the length and breadth of the sudoku and they vary depending upon the matrix in question .The Length and Breadth of the matrix of Sudoku is equal to the square of its degree ,i.e., a 3*3 matrix will have a length of 9 and the breadth of 9 .Also the number of individual elements ,i.e., the places where number from 1 to 9 can be input, is the product of the length and breadth or simply square of either or the fourth power of the matrix name.

2. Boxes :Boxes refer to submatrices .The size and number of the submatrix can vary depending upon the type of sudoku .The number of submatrices depend upon the size of a Sudoku matrix . The number of submatrices is equal to the square of degree of matrix .A 3*3 Sudoku has 9 boxes while a 4*4 has 16 and so on .

3. Coordinate system :As we are only focused on 2D Sudoku for this discussion, we will use a xy-axes coordinate system for this representation. X or the Horizontal axis will be referred to as row and Vertical Axis or Y-Axis will be referred to as column

1.1 Conventions

1. As the matrix will be a multidimensional array , the coordinate system will provide a way of accessing the individual elements .
2. The first element of a row will be designated the coordinate of (0,column) and the last element as the ((length-1),column).
3. The first element of a column will be designated the coordinate of (row,0) and the last element as the (0,(breadth-1))
4. All empty boxes will be designated the number 0.
5. The array of sudoku will be named sudoku_matrix

III. METHODOLOGY

We consider that the matrix is already input in the form of a multidimensional array which will have 9 subarrays each with 9 elements with the first having the index of 0 and the last one with the index of 8.

Note: We are using the example of a 3*3 sudoku matrix ,however, this method will work for any degree of matrix.

1.**Row Checker**: It is a function that checks whether a particular number is present in the row or not .

```
for x in range(0, size):
   if sudoku_matrix[row][x] == number:
       counter = counter + 1
      counter = counter + 1
```

2.**Column Checker** : It is a function that checks whether a particular number is present in the column or not .

```
for x in range(0, size):
   if sudoku_matrix[x][column] == number:
       counter = counter + 1
      counter = counter + 1
```

3.**Box Checker** :It is a function that checks whether a specific number is in the particular box or not.It uses math to classify different boxes and can be changed to do any number of them .The box checker uses to make boxes starting from the right and making the number of boxes depending upon the degree of the Sudoku .When a number is input it uses floor divisions to determine which box it is in and then uses a nested loop which iterates through each and every element in the box to check whether the element is in it or not. If the number is found , it updates the counter variable

```
for i in range(row_box, row_box + 3):
   for j in range(column_box, column_box + 3):
       if sudoku_matrix[i][j] == number:
           counter = counter + 1
```

4.**Empty Box Checker** :It is a function that checks whether all the boxes are filled with number or not .If no box is empty ,i.e.,has the value of element that is not zero **,**it will relay this to the main function .If no box is empty that means the sudoku has been solved and main function will give a signal to display the sudoku however if any element is empty the function will return its coordinates to the main function and the main function will try to go through all the 3 conditions mentioned above to assign it a value.

```
def empty_box_checker(rows, columns):
   empty_location = 0
   for x in range(0, 9):
       for y in range(0, 9):
           if sudoku_matrix[x][y] == 0:
               rows = x
               columns = y
               empty_location = 100
               array = [rows, columns, empty_location]
               return array
   array = ['X-Coordinate', 'Y-Coordinate', empty_location]
   return array
```

5.**Printer** :This function is used to print sudoku in a segregated manner .It divides the matrix into boxes and prints it for a better understanding of the solution.

```
def print_sudoku():
   for x in range(0, 9):
       if x % 3 == 0 and x != 0:
           print("- - - -|- - - - |- - - - |")
       for y in range(0, 9):
           print(sudoku_matrix[x][y], end=' ')
           if (y + 1) % 3 == 0:
               print(' | ', end='')
       print('')
```

6.**Assigner** :This is a sub-function to the main function and allots values to an element when all other conditions are satisfied .However when there is a wrong input , it simply allots the value of 0 to the element as in these cases the sudoku cannot be solved.

```
for i in range(1, 10):
   if validity(i, row, column):
       sudoku_matrix[row][column] = i
       if solver():
           return True
       sudoku_matrix[row][column] = 0
```

IV. INTEGRATION

1.**validity_checker()** :the function combines all the conditions into one and checks if the conditions are satisfied or not .If a condition is not satisfied , it will update the counter .The counter is initially assigned the value of 0 and updates by one if a condition is not satisfied .At the end of the function if all conditions are satisfied , it return a true and even of one of them is not , it returns a false.

```
def validity(number, row, column):
   counter = 0
   for x in range(0, 9):
       if sudoku_matrix[row][x] == number:
           counter = counter + 1
   for y in range(0, 9):
       if sudoku_matrix[y][column] == number:
           counter = counter + 1
   row_box = (row // 3) * 3
   column_box = (column // 3) * 3
   for i in range(row_box, row_box + 3):
       for j in range(column_box, column_box + 3):
           if sudoku_matrix[i][j] == number:
               counter = counter + 1
   if counter == 0:
       return True
   else:
       return False
```

2.**empty_box_checker()** : [3]This function is invoked by the main function and checks whether all boxes are filled or not. The data is relayed through an array where if the third variable of the array with 3 variables .If a box is not filled , it relays its coordinates to the main function where a loop is used to assign values between 1 to 9 to the particular coordinate and uses the validity_checker to check which value is appropriate value .If all the boxes are full , it relays a value of 0 to the main function which understands that all the elements are full and thus invokes the function of print the matrix.

```
def empty_box_checker(rows, columns):
   empty_location = 0
   for x in range(0, 9):
       for y in range(0, 9):
           if sudoku_matrix[x][y] == 0:
               rows = x
               columns = y
               empty_location = 100
               array = [rows, columns, empty_location]
               return array
   array = ['X-Coordinate', 'Y-Coordinate', empty_location]
   return array
```

3.**print_sudoku()** :It is used to print the sudoku

```
def print_sudoku():
   for x in range(0, 9):
       if x % 3 == 0 and x != 0:
           print("- - - -|- - - - |- - - - |")
       for y in range(0, 9):
           print(sudoku_matrix[x][y], end=' ')
           if (y + 1) % 3 == 0:
               print(' | ', end='')
       print('')
```

4.solver():This is the main function of the program .The function starts with invoking the empty_box_checker to check whether the sudoku is already solved or not .If the empty_box_checker function return an array .Next it uses 2 variables called row and column with each assigned the value 0 at start and these act as the variables for the coordinate system .The assigner is used to figure out which value is apt for the element .The assigner inturn employs the validity_checker to check the whether a particular value is at that particular element is fine or not .If the loop exhausts and no value could be assigned ,it then allocates the value 0 to the element.If the function is exhausted and still all places could not assigned values , the function return false signifying that the matrix entered is not correct .

```
def solver():
   row = 0
   column = 0
   array = empty_box_checker(row, column)
   if array[2] == 0:
       return True
   row = array[0]
   column = array[1]
   for i in range(1, 10):
       if validity(i, row, column):
           sudoku_matrix[row][column] = i
           if solver():
               return True
           sudoku_matrix[row][column] = 0
   return False
```

5.I**nput Validity Checker and Final Output:** This statement invokes solver() and as mentioned before , if it return false , the matrix is perceived to be flawed by the computer and a message is output .On the other hand if answer from solver is true,the solved sudoku is displayed using print_sudoku function.

```
if solver():
print_sudoku()
else:
print('No Solution')
```

V. SOURCE CODE

Note: The program also has a test matrix in it.

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


def print_sudoku():
for x in range(0, 9):
if x % 3 == 0 and x != 0:
print("- - - -|- - - - |- - - - |")
for y in range(0, 9):
print(sudoku_matrix[x][y], end=' ')
if (y + 1) % 3 == 0:
print(' | ', end='')
print('')


def empty_box_checker(rows, columns):
empty_location = 0
for x in range(0, 9):
for y in range(0, 9):
if sudoku_matrix[x][y] == 0:
rows = x
columns = y
empty_location = 100
array = [rows, columns, empty_location]
return array
array = ['X-Coordinate', 'Y-Coordinate', empty_location]
return array


def validity(number, row, column):
counter = 0
for x in range(0, 9):
if sudoku_matrix[row][x] == number:
counter = counter + 1
for y in range(0, 9):
if sudoku_matrix[y][column] == number:
counter = counter + 1
row_box = (row // 3) * 3
column_box = (column // 3) * 3
for i in range(row_box, row_box + 3):
for j in range(column_box, column_box + 3):
if sudoku_matrix[i][j] == number:
counter = counter + 1
if counter == 0:
return True
else:
```

```
return False


def solver():
row = 0
column = 0
array = empty_box_checker(row, column)
if array[2] == 0:
return True
row = array[0]
column = array[1]
for i in range(1, 10):
if validity(i, row, column):
sudoku_matrix[row][column] = i
if solver():
return True
sudoku_matrix[row][column] = 0
return False


if solver():
print_sudoku()
else:
print('No Solution')
```

VI. ACKNOWLEDGEMENTS

[1] https://www.geeksforgeeks.org/sudoku-backtracking-7/
[2] https://leetcode.com/problems/sudoku-solver/
[3] https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
[4] https://www.codesdope.com/blog/article/solving-sudoku-with-backtracking-c-java-and-python/
[5] https://en.wikipedia.org/wiki/Sudoku
[6] http://pi.math.cornell.edu/~mec/Summer2009/Mahmood/Intro.html