Lecture 12- Backtracking Algorithm for solving Sudoku Problem

Sudoku

Sudoku is a number-placement puzzle where the objective is to fill a square grid of size 'n' with numbers between 1 to 'n'. The numbers must be placed so that each column, each row, and each of the sub-grids (if any) contains all of the numbers from 1 to 'n'.

The most common Sudoku puzzles use a 9x9 grid. The grids are partially filled (with hints) to ensure a solution can be reached.

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

And here's the solution. Notice how each row, each column and each sub-grid have all numbers from 1 to 9. Some puzzles may even have multiple solutions.

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

Sudoku & Backtracking

We will now create a Sudoku solver using backtracking by encoding our problem, goal and constraints in a step-by-step algorithm.

Problem

Given a, possibly, partially filled grid of size ‘n’, completely fill the grid with number between 1 and ‘n’.

Goal

Goal is defined for verifying the solution. Once the goal is reached, searching terminates. A fully filled grid is a solution if:

1. Each row has all numbers form 1 to ‘n’.
2. Each column has all numbers form 1 to ‘n’.
3. Each sub-grid (if any) has all numbers form 1 to ‘n’.

Constraints

Constraints are defined for verifying each candidate. A candidate is valid if:

1. Each row has unique numbers form 1 to ’n’ or empty spaces.
2. Each column has unique numbers form 1 to ‘n’ or empty spaces.
3. Each sub-grid (if any) has unique numbers form 1 to ‘n’ or empty spaces.

Termination conditions

Typically, backtracking algorithms have termination conditions other than reaching goal. These help with failures in solving the problem and special cases of the problem itself.

1. There are no empty spots left to fill and the candidate still doesn’t qualify as a the solution.

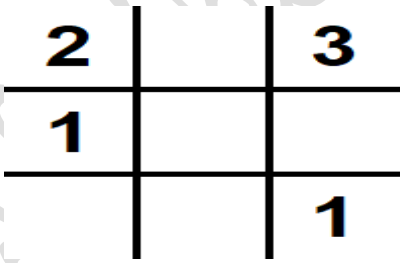
2. There are no empty spots to begin with, i.e., the grid is already fully filled.

Step-by-step algorithm

Here's how our code will "guess" at each step, all the way to the final solution:

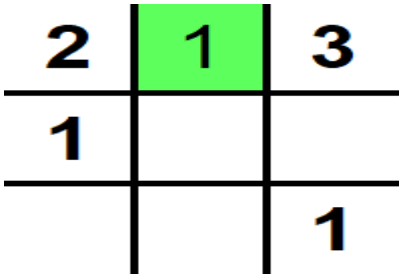
1. Make a list of all the empty spots.
2. Select a spot and place a number, between 1 and 'n', in it and validate the candidate grid.
3. If any of the constraints fails, *abandon candidate and repeat step 2 with the next number*. Otherwise, check if the goal is reached.
4. If a solution is found, stop searching. Otherwise, repeat steps 2 to 4.

Now let's try all this in practice with a simple 3x3 grid.

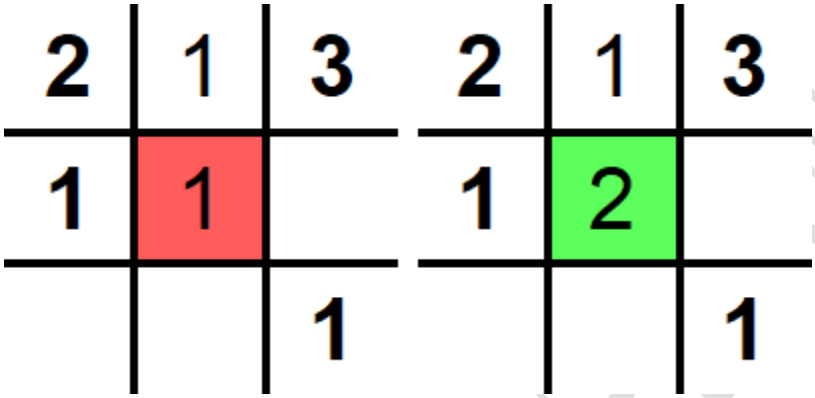


We start off by listing all the empty spots. If we label each cell in the grid with a pair of numbers (x,y) and mark the first cell (1,1), then our empty spots will be at locations:

(1,2) (2,2) (2,3) (3,1) (3,2)
We now select the first spot (1,2) to work with. Since this is a 3x3 grid, we have numbers 1 to 3 at our disposal and no sub-grids to worry about (sub-grids are only a bother for grids with squared sides, like 4, 9, 16 etc.).
Let's place number 1 in this spot and see if it fits.

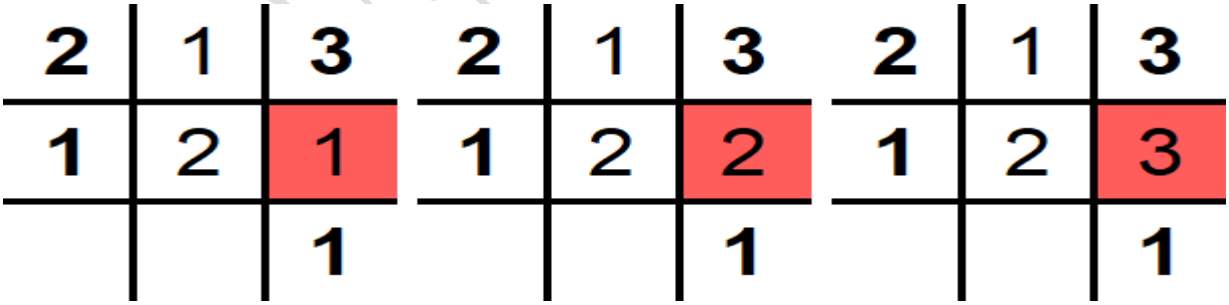


We can now select the next spot on the list (2,2) and do the same thing again. This time however, it fails. We already have a 1 in this row. This means that we must abandon candidate and repeat step 2 with the next number—which is 2.



One more spot is filled. Also, it might not look like it, but we did just perform backtracking on a single spot. We abandoned a candidate solution (1 at spot (2,2)), visited a previous stage (empty spot (2,2)) and explored a new candidate solution (number 2 at spot (2,2)).

When we move on to spot (2,3), we have another problem. As you can see, we are all out of options. None of the possible numbers fit in. This means that we must now abandon candidate and repeat step 2 with the next number. Only this time, we must visit spot (2,2) first to fix spot (2,3).



We need to fill number 3 in spot (2,2) and that will resolve the issue.

2	1	3
1	3	2
		1

We now repeat this process until with either reach the goal or we hit one of the termination conditions.

Since this was a demo problem, it should be obvious that we'd arrive at the solution without any further complications.

2	1	3
1	3	2
3	2	1

However, consider the same grid with one small change. Replacing the 1 in cell (3,3) with a 2 renders the grid unsolvable. Similarly, removing hints from cells (2,1) and (3,3) allows for multiple solutions. But since this algorithm has a single goal, it stops after the first solution is reached.

2	1	3
1	3	?
		2

2	1	3
1/3	3/2	2/1
3/1	2/3	1/2