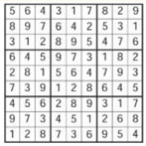


5	6	4	3	1	7	8	2	9
8	9	7	6	4	2	5	3	1
3	1	2	8	9	5	4	7	6
6	4	5	9	7	3	1	8	2
2	8	1	5	6	4	7	9	3
7	3	9	1	2	8	6	4	5
4	5	6	2	8	9	3	1	7
9	7	3	4	5	1	2	6	8
1	2	8	7	3	6	9	5	4

Goal

- In 1984 the Japanese newspaper Nokilist Monthly published in a section called hobbies *Sūji wa dokushin ni kagiru* (" the numbers must be alone"*). It was Kaji Maki, president of Nikoli, who gave it the name. The name was shortened to Sūdoku (sū = number, doku = alone)*

- The standard Sudoku puzzle consists of a 9x9 grid, broken into nine 3x3 boxes

- Each of the eighty-one squares must be filled in with a number between 1 and 9

- Some cells already contain numbers, known as "givens" or "tracks"

- The aim is to fill the empty cells with a number in each, so that each **column, row** and **region** contains the numbers 1-9 only once

- Each number in the solution appears only once in each of the three "directions“
    - Hence "the numbers must be alone" evoking the name of the game

Data for a specific problem

- A board of 9x9 composed of regions of 3x3

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

→

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

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

?

Strategy (I)

- Rule 1: Complete a row or column or box when only one cell is open and eight digits have been filled in already

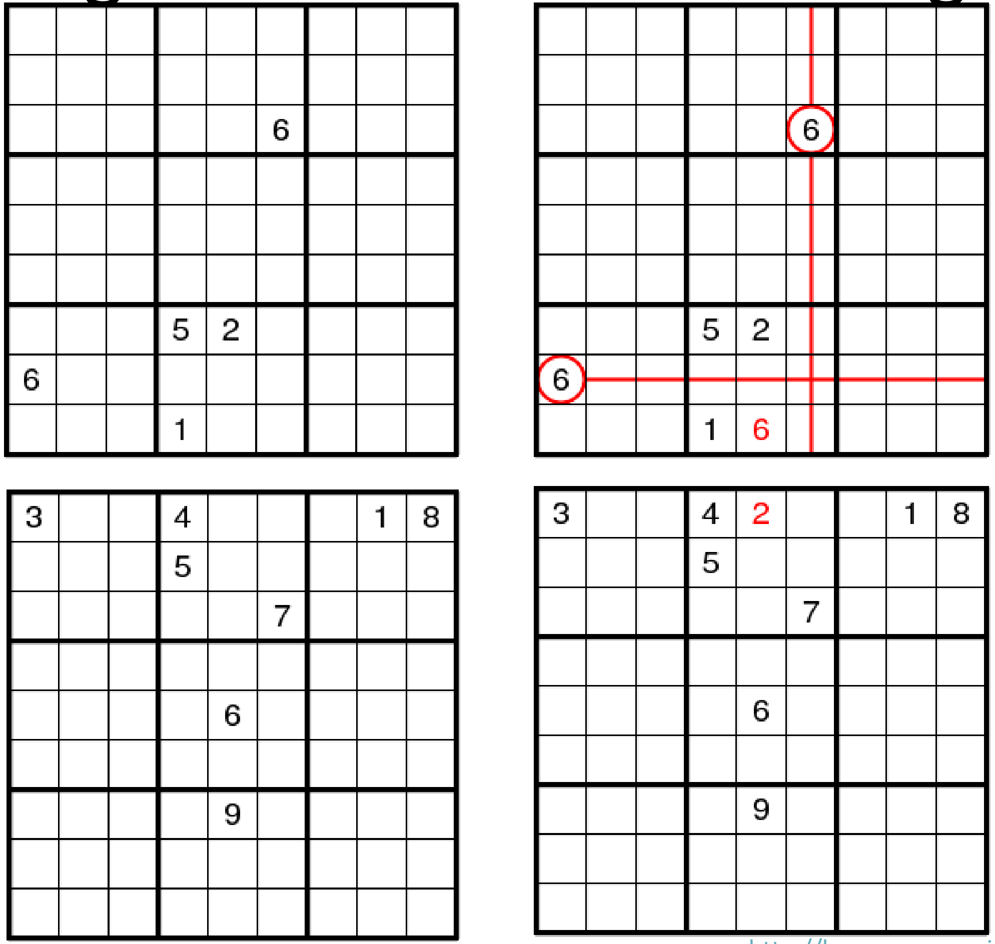
						5	9	4
4	2		9	5	1	6	3	8
							2	7
								6
								3
								5
								1
								9

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

http://homepages.cwi.nl/~aeb/games/sudoku/

Strategy (II)

- Rule 2: Write a digit in a box where it can go in only one place



					6			
			5	2				
6								
			1					

					6			
			5	2				
6								
			1	6				

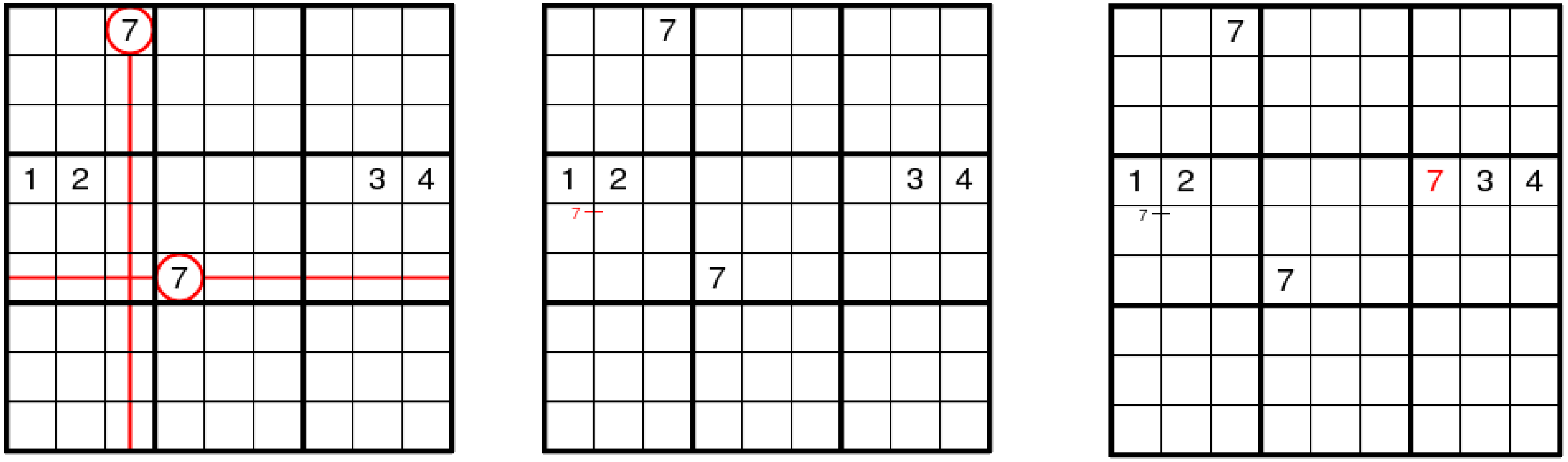
3			4				1	8
			5					
					7			
				6				
				9				

3			4	2			1	8
			5					
					7			
				6				
				9				

http://homepages.cwi.nl/~aeb/games/sudoku/

Strategy (III)

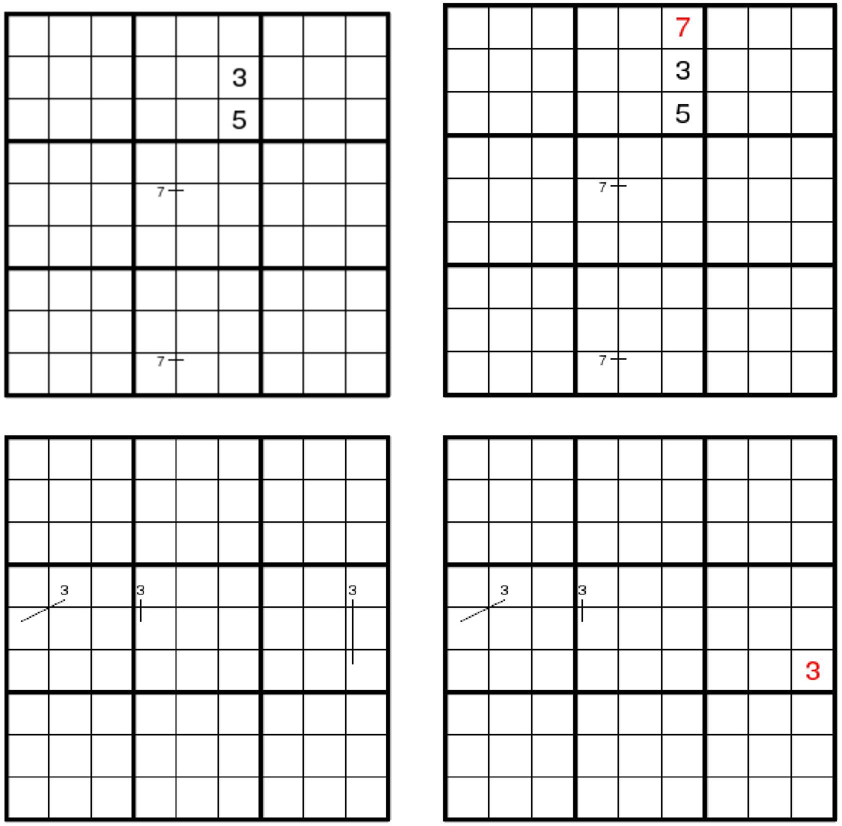
- If a Sudoku can be solved with the previous techniques, this is an **easy Sudoku**
  - At each step there is only one possibility
- Rule 3: For more complex Sudokus it is necessary to annotate
  - Pair



http://homepages.cwi.nl/~aeb/games/sudoku/

Strategy (IV)

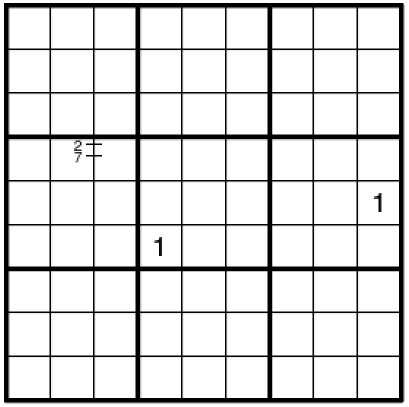
- For more complex Sudokus it is necessary to annotate
  - Two pairs

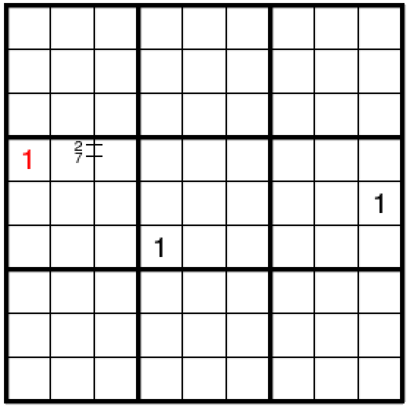


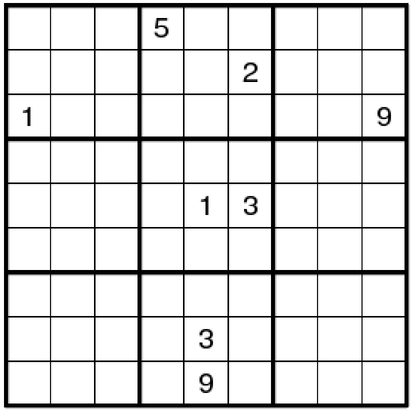
http://homepages.cwi.nl/~aeb/games/sudoku/

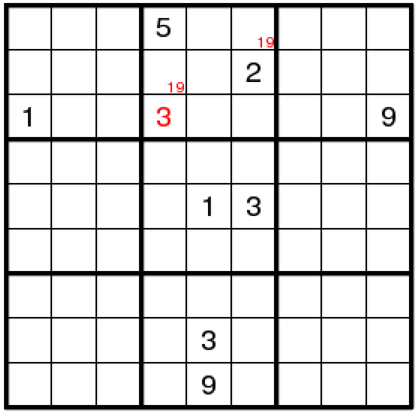
Strategy (V)

- For more complex Sudokus it is necessary to annotate
  - Double pairs









http://homepages.cwi.nl/~aeb/games/sudoku/

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

Strategy (VI)

- Despite these simple rules there are 6,670,903,752,021,072,936,960 valid Sudokus
- There are different strategies that are effective to solve this problem
- The brute force algorithm is the simplest (and a general algorithm)
  - Try all combinations until you find one that works
  - It works because computers are fast
  - Although it is not optimized…

Strategy (VII)

- In the example, we have reached a dead
- When the search reaches a not applicable end, it returns to the previous cell that was trying to fill in and tries the next digit
- We would back up to the cell with a 2 and that turns out to be a dead end as well so we back up again
  - That way, the algorithm needs to remember what number to try next
- Now in the cell with the 2. We try 3 and move forward again. Since 3 does not work either, we try 4 and move forward again

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

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

Strategy (VIII)

- Brute force algorithms are pretty slow but easy to implement
    - Backtracking is an example of a brute force algorithm
- There are not very clever
    - For example we know that in the middle-top box there can not be a one in either the first or the third column but the algorithm still tries it
- After trying placing a digit in a cell we want to solve the new Sudoku board
    - **It is a smaller (or a simpler version) of the same problem we started with**
- We will use recursive backtracking
    - Recursive because later versions of the problem are just slightly simpler versions of the original
    - Backtracking because we may have to try different alternatives

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

Strategy (IX)

- `board` → array that represents the board

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

- `int [][] board= new int [n][n]`
  - `board[x][y]=0` cell (x y) has not been calculated
  - `board[x][y]=i` cell (x, y) has been calculated
- `i` → value of the digit (from 1 to 9)
- `x, y` → coordinates of the last value in the Sudoku board
- Initial `board[0][0]`

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

Strategy (X)

- Valid values on the board
  - Check row
    - `checkIfRowHasValue(value)`
  - Check column
    - `checkIfColumnHasValue(value)`
  - Check subtable (region)
    - `checkIfRegionHasValue(value)`

-
```
legal(value) {
      if ((checkIfRowHasValue(value)) ||
            (checkIfColumnHasValue(value)) ||
            (checkIfRegionHasValue(value))) return false;
      else return true;
  }
```

Only for helping to understand. It does not follow the general scheme given in the Java code

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

Pseudocode

```
public boolean findSolution(int i, int j){
        boolean hasSolution= false;
        if (j == 9) { j= 0; i++; }//when a column ends, go to the next row

        if (i == 9) return true; //check if it is solution (you got the end)

        if (cells[i, j] == 0) //skip cells already occupied
                 for (int val= 1; val<=9 && !hasSolution; val++){
                         if (legal(i, j, val, cells)){//check if it is a
                                                        valid state
                                 cells[i][j] = val;

                                 findSolution (i,j+1,cells);

                                 cells[i,j] = 0;

                         }//if
                 }//end (for)
        } //end (if)
        return hasSolution;
   }
```

Solution

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

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