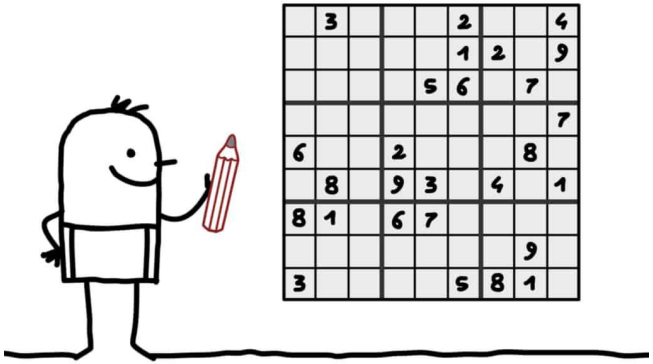
Workshop 7 - Sudoku



Sudoku

The objective of the original game is to fill a 9×9 square with digits, so that every column, every row, and every one of the nine 3×3 sub-squares within the square contains all of the digits from 1 to 9. In order to make the solution easier, the puzzle maker normally provides some of the digits, and the player has to complete the missing digits. Here is an example of a valid solution:

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

The Sudoku Class

In order to make the problem more general, we will consider Sudoku games in which the size of the square can be any number N that has an integer square root. That is, N = 1, 4, 9, 16, 25, etc. In this general case, the rules of the game are exactly the same, except that we have to fill the N-by-N square with the numbers 1,...,N (in the example that we saw above N=9).

We wish to write a Sudoku class that does one thing only: checks if a given N-by-N square contains a valid Sudoku solution. Once again, note that we don't worry here about playing the game. Instead, we wish to verify that a proposed solution is valid. That's all.

Approaching the task

- What will be our strategy?

- How can we split the problem into sub-problems?

- What functions would we need to implement in order to make the solution **simple** and **elegant**?

Implementation Strategy

```
// Given: an array of size N.
// Checks if the given array contains all the numbers 1,2,3...,N.
private static boolean containsAllNumbers(int[] arr)

// Given a 2-dimensional array and a column number,
// returns an array containing the values in this column.
// Assumes that col is a valid column number -- no need to check it.
private static int[] getColumn(int[][] arr, int col)

// Given a square 2-dimensional array, an index (row,col), and a size n,
// returns the n-by-n sub-array whose top-left coordinate is (row,col).
// Assumes that row, col, and n are all valid -- no need to check it.
private static int[] getSquare(int[][] arr, int row, int col, int n)

// Checks if the given 2D-array is a Sudoku square
private static boolean sudoku(int[][] arr)
```

Contains All Numbers

The following method checks that a given array of size N contains all the numbers 1,2,3...,N (assume that all the numbers in the array are between 1 and N, inclusive).

```
private static boolean containsAllNumbers(int[] arr)
```

For example, `containsAllNumbers({2, 1, 4, 3})` will return true, while `containsAllNumbers({1, 2, 3, 1})` will return `false`.

Get Column

The following method gets a 2-dimensional array and a column number, and returns an array that contains all the values in that column. The method assumes that the column number is valid, and there is no need to write code that checks it.

```
private static int[] getColumn(int[][] arr, int col)
```

For example, `getColumn(m,4)` where m is the following matrix, will return the array `[7, 9, 4, 6, 5, 2, 3, 1, 8]`

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

Get Square

The following method gets a 2-dimensional array and a sub-square within it. The method returns a 1-dimensional array which is the "flattening" of the nxn sub-square whose top-left coordinate is (row, col). The method assumes that row, col, and n are all valid, and there is no need to write code that checks it.

```
private static int[] getSquare(int[][] arr, int row, int col, int n)
```

For example, `getSquare(m,6,6,3)` where m is the following matrix will return the array `[2, 8, 4, 6, 3, 5, 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

→

2	8	4
6	3	5
1	7	9

Final Step

The following method checks if the values of the given 2-dimensional square array form a valid Sudoku solution. Note that if the size of the given square array is n-by-n, then according to the Sudoku rules, the size of every sub-square is $\sqrt{n}$ by $\sqrt{n}$. (the method assumes that n has an integer root $\sqrt{n}$, and there is no need to check it).

```
private static boolean sudoku(int[][] arr)
```

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

HW Questions?