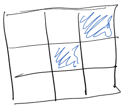
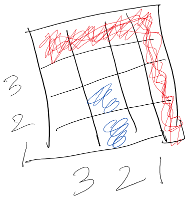
# Recursion

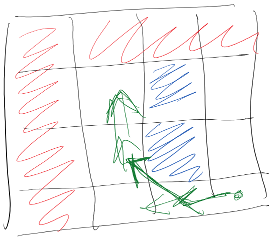
① Maze with obstacles :



obstacle → false boolean value in a matrix

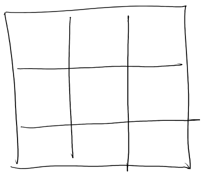
solution: just check if (arr [row][col]) and only then make the recursion call.

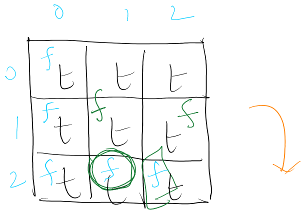




RRDD
RDiaD

## ② Alldirections with backtracking

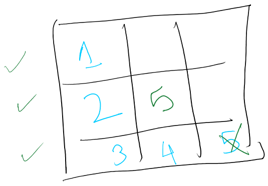




2 1

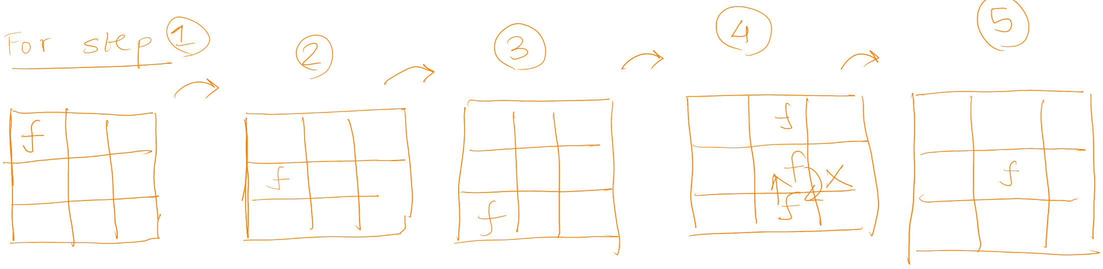
Path [ 1 ]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| row | 0 | 1 | 2 | 2 | 2 | 1 | ~~2~~ | 1 | 2 |
| coln | 0 | 0 | 0 | 1 | 2 | 1 | 1 | 2 | 2 |

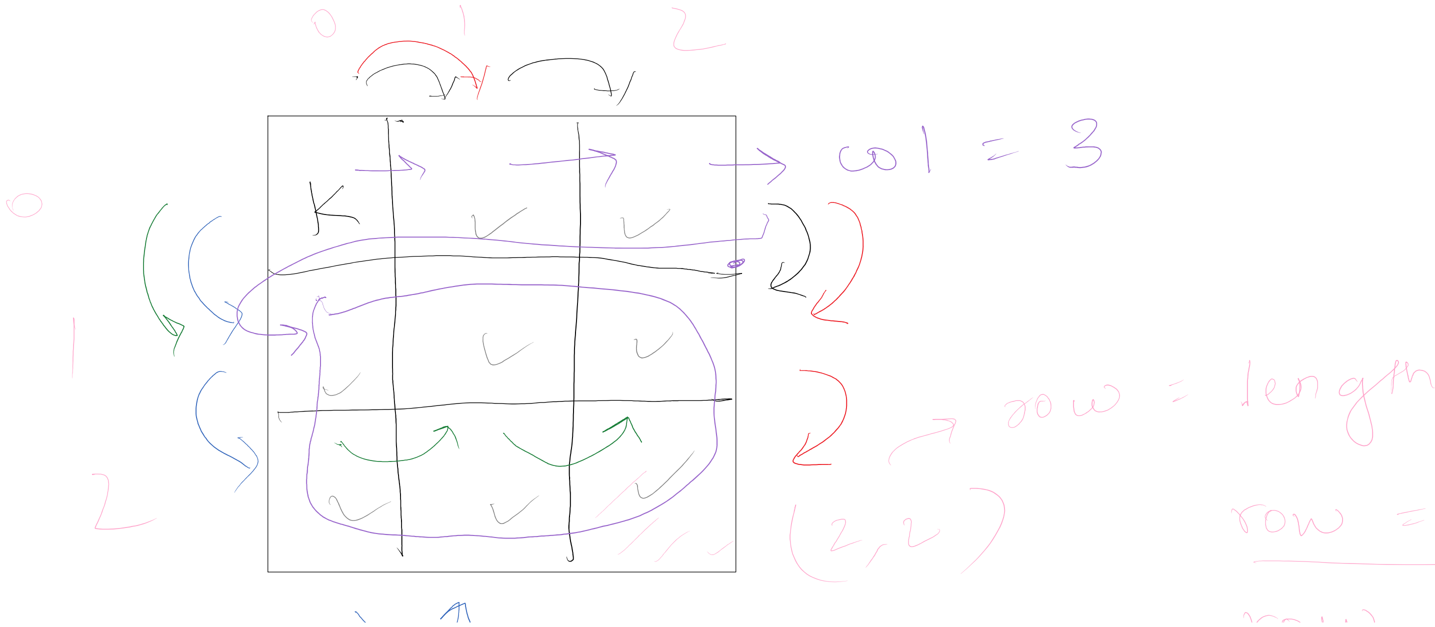


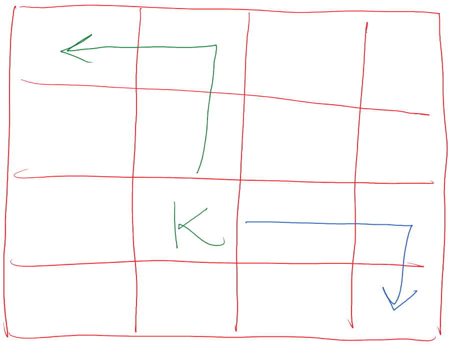
DDRR

For step ①



## ③ N-Knights





3 5

(board) - 1

2

(col - 2, row - 1)

(col + 2, row + 1)

row

$$\begin{bmatrix} row+2, & col+1 \\ row-2, & col+1 \\ row+1, & col+2 \\ row-1, & col+2 \end{bmatrix}$$

→ ↓ ↓

→ ↑ ↑

→ → ↓

→ → ↑

# Sudoku solver

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

(char[][] bo

3 → solve(),

① → cast char

② solve() ?

(6,6)

$(\overset{r}{7},\overset{c}{7})$

$r = 7 - 7\%3$
$= 6$

$c = 7 - 7\%3$
$= 6$

③ isSafe() ?
Try to ins

$< 0$ (col + 2, row + 1)

oad)

is safe( ), display( )

acter array to numerical array

a) Checking empty box (row & col)

b) checking distance i.e. subgrid 3 x 3

$r = r - r \% \text{sqrt}$

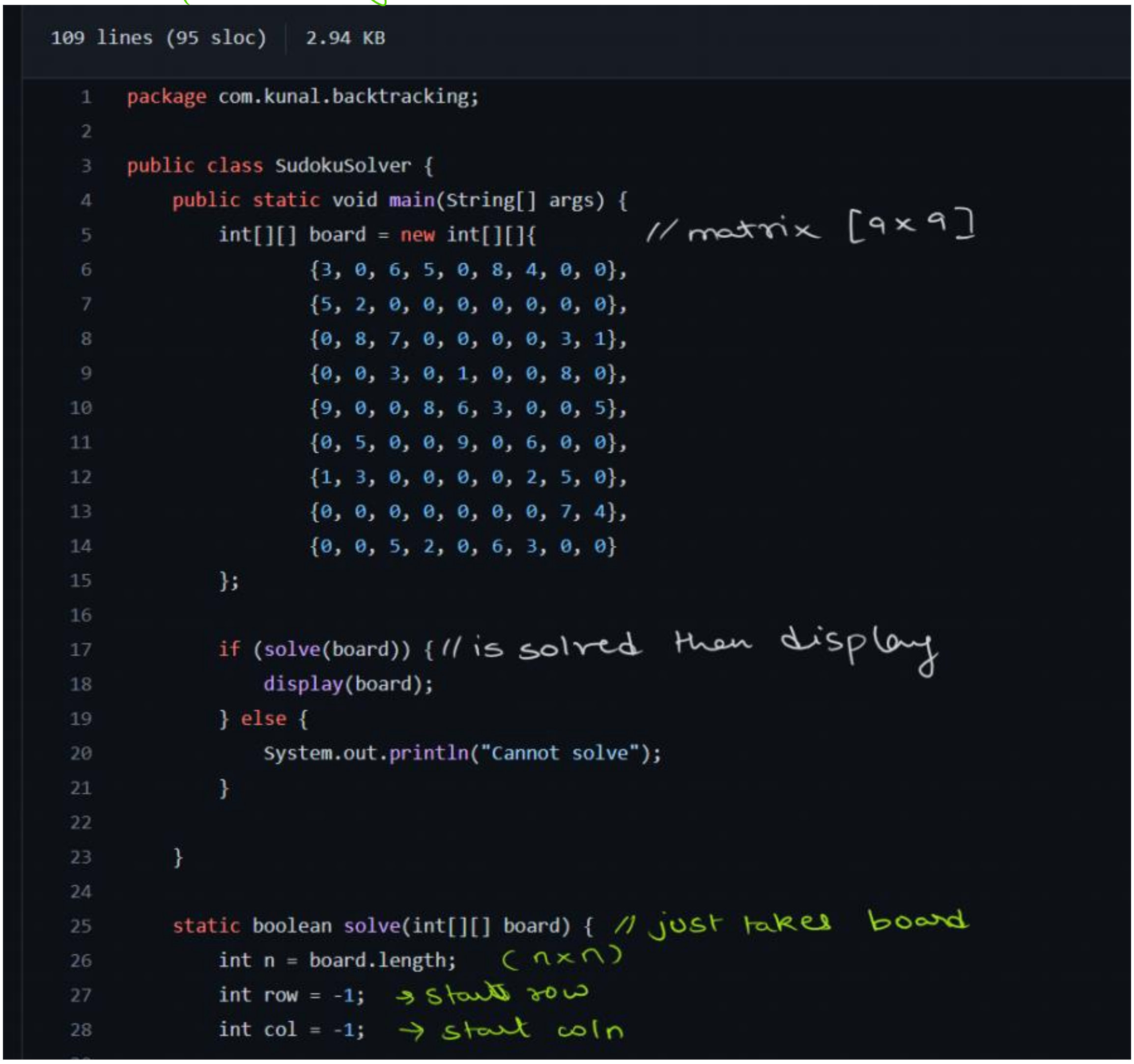
$c = c - c \% \text{sqrt}$

c) check is safe()

sert no.'s [1 - 9]

multiples of 3 + remainder

→ keep c
→ keep



109 lines (95 sloc) | 2.94 KB

```
package com.kunal.backtracking;

public class SudokuSolver {
    public static void main(String[] args) {
        int[][] board = new int[][]{          // matrix [9x9]
                {3, 0, 6, 5, 0, 8, 4, 0, 0},
                {5, 2, 0, 0, 0, 0, 0, 0, 0},
                {0, 8, 7, 0, 0, 0, 0, 3, 1},
                {0, 0, 3, 0, 1, 0, 0, 8, 0},
                {9, 0, 0, 8, 6, 3, 0, 0, 5},
                {0, 5, 0, 0, 9, 0, 6, 0, 0},
                {1, 3, 0, 0, 0, 0, 2, 5, 0},
                {0, 0, 0, 0, 0, 0, 0, 7, 4},
                {0, 0, 5, 2, 0, 6, 3, 0, 0}
        };

        if (solve(board)) { // is solved then display
            display(board);
        } else {
            System.out.println("Cannot solve");
        }

    }

    static boolean solve(int[][] board) { // just takes board
        int n = board.length;   ( n x n)
        int row = -1;   → start row
        int col = -1;   → start coln
```

Steps

① chec

② chec

③ che
ma

Steps

① chec

② set c
store

③ once
out

heck on row

check on column

for isSafe()

ck row elements matching

ck coln elements matching

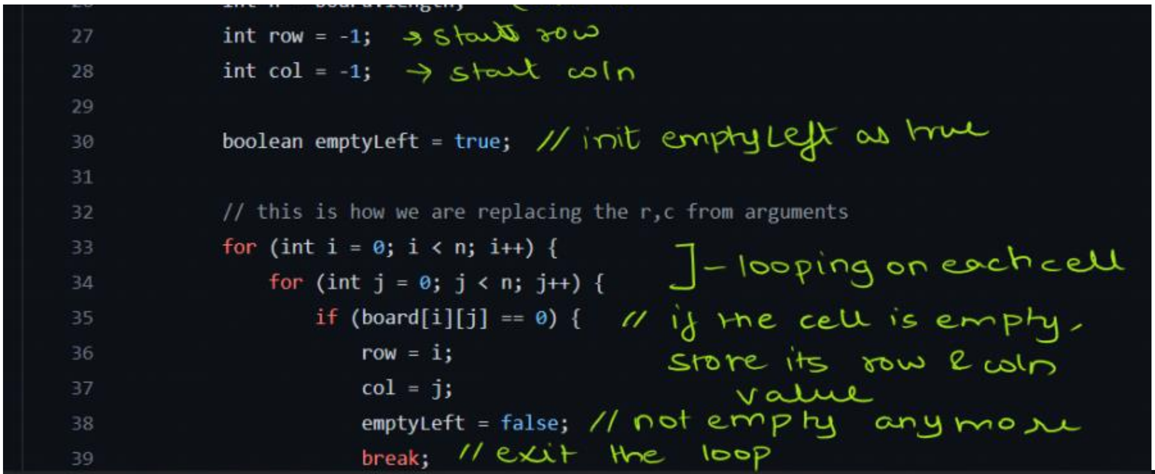
ck subgrid elements
atching [use formula]

for solve()

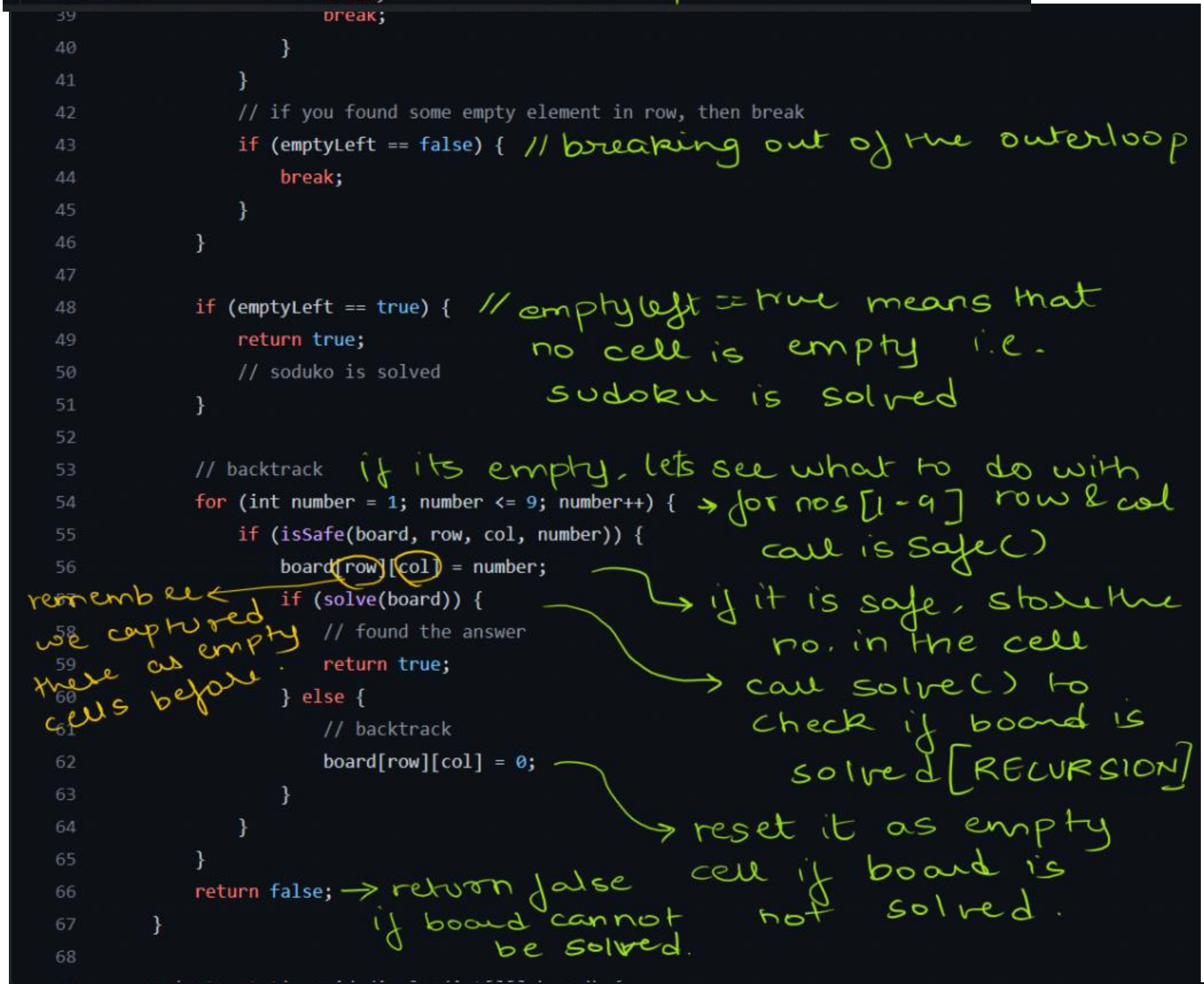
k which cell is empty

emptyleft == false and
row & col of empty cell

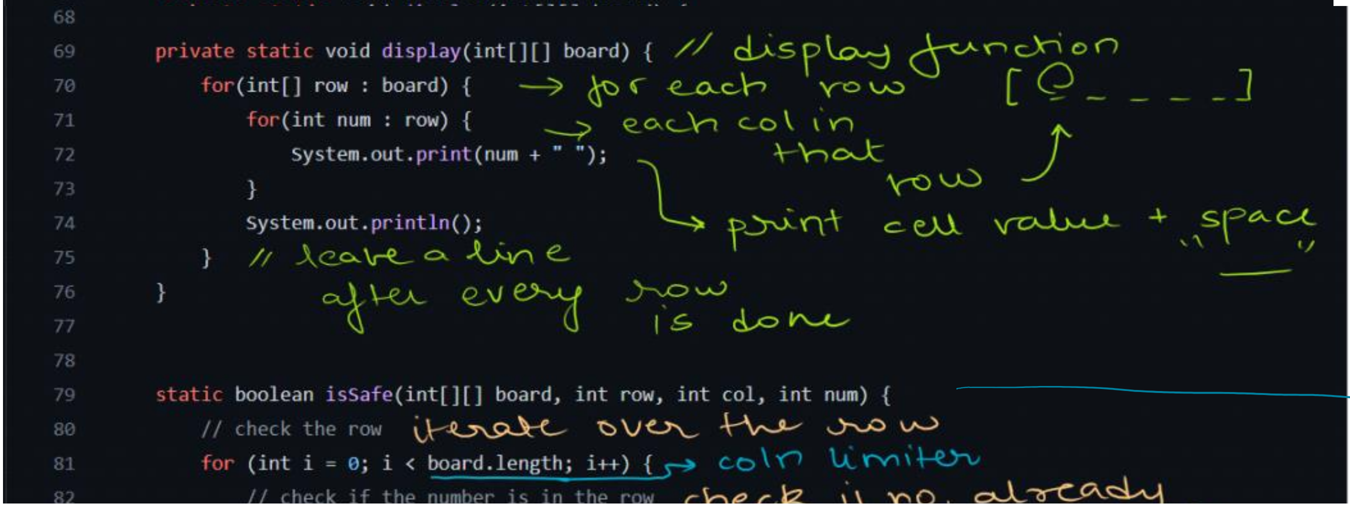
empty is found, break
ed loops



```
        int row = -1;   → start row
        int col = -1;   → start coln

        boolean emptyLeft = true;  // init emptyLeft as true

        // this is how we are replacing the r,c from arguments
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {      ]- looping on each cell
                if (board[i][j] == 0) {    // if the cell is empty,
                    row = i;                  store its row & coln
                    col = j;                      value
                    emptyLeft = false;  // not empty any more
                    break;  // exit the loop
```



```
                }
            }
            // if you found some empty element in row, then break
            if (emptyLeft == false) {  // breaking out of the outerloop
                break;
            }
        }

        if (emptyLeft == true) {  // emptyLeft == true means that
            return true;              no cell is empty i.e.
            // soduko is solved        sudoku is solved
        }

        // backtrack   if its empty, lets see what to do with
        for (int number = 1; number <= 9; number++) {  → for nos [1-9] row & col
            if (isSafe(board, row, col, number)) {          call isSafe()
                board[row][col] = number;        → if it is safe, store the
remember ←      if (solve(board)) {                  no. in the cell
we captured         // found the answer          → call solve() to
these as empty      return true;                    check if board is
cells before    } else {                            solved [RECURSION]
                    // backtrack
                    board[row][col] = 0;      → reset it as empty
                }                                 cell if board is
            }                                     not solved.
        }
        return false; → return false
    }                   if board cannot
                           be solved.
```



```
    private static void display(int[][] board) { // display function
        for(int[] row : board) {     → for each row        [@ _ _ _ _]
            for(int num : row) {     → each col in              ↑
                System.out.print(num + " ");   that
            }                                      row
                                        → print cell value + " space "
            System.out.println();
        }  // leave a line
    }        after every row
                        is done

    static boolean isSafe(int[][] board, int row, int col, int num) {
        // check the row  iterate over the row
        for (int i = 0; i < board.length; i++) { → coln limiter
            // check if the number is in the row  check if no. already
```

(3) once
out

(4) If so
if m
that

(5)

e

(6

(7

New Section 1 Page 7

empty is found, break
of loops

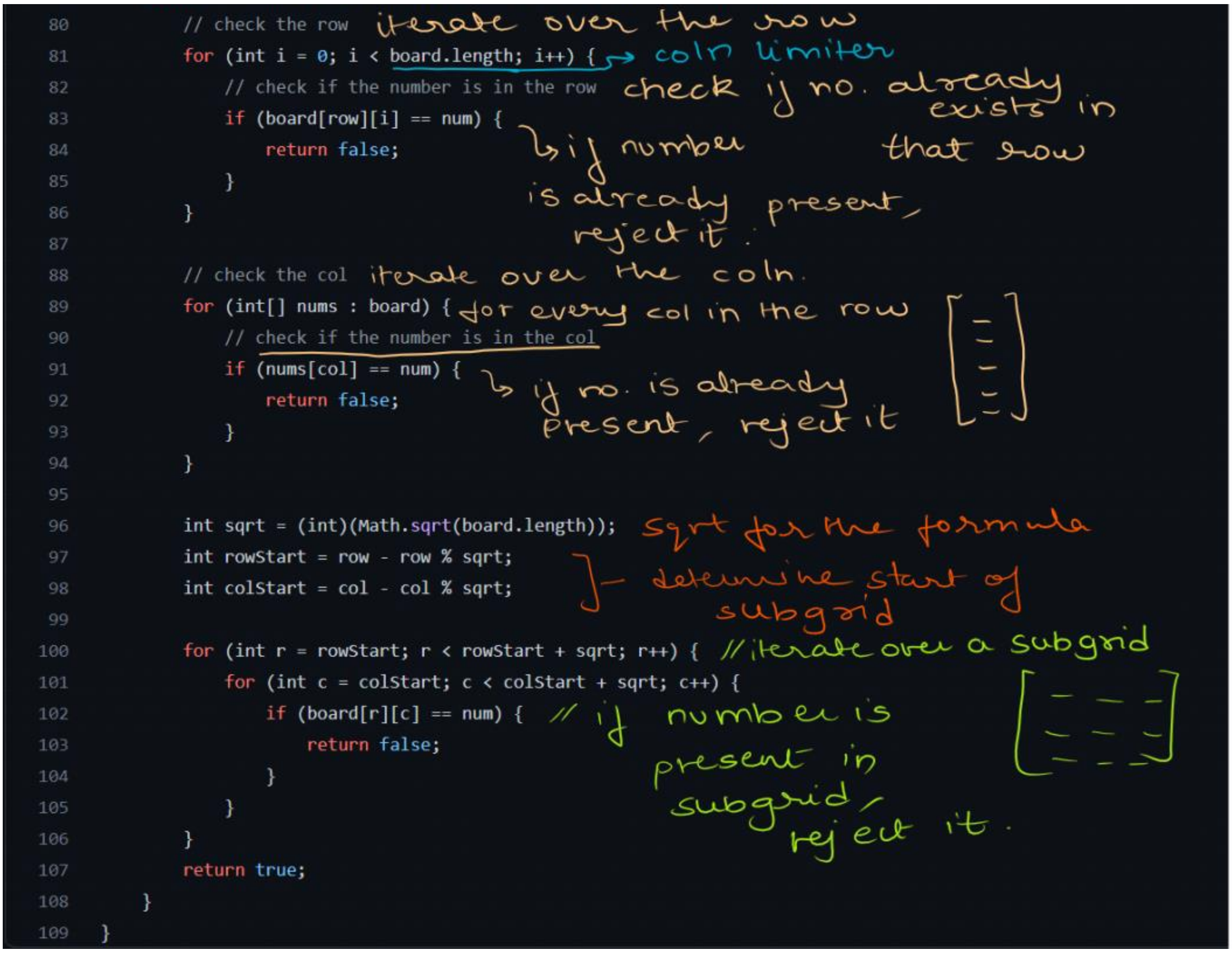
omehow emptyleft is true,
eans no empty cell &
sudoku is complete

For nos. [1-9] for
mpty cell (row, col)
all isSafe() function.

) If it is safe, store that
no. in the empty cell and
make recursion call
to solve() again.

) If it is not safe,
replace (row, col) with
empty cell i.e.
backtrack.

→ Lets assume
we called
loop (row, col)



```
        // check the row
        for (int i = 0; i < board.length; i++) {
            // check if the number is in the row
            if (board[row][i] == num) {
                return false;
            }
        }

        // check the col
        for (int[] nums : board) {
            // check if the number is in the col
            if (nums[col] == num) {
                return false;
            }
        }

        int sqrt = (int)(Math.sqrt(board.length));
        int rowStart = row - row % sqrt;
        int colStart = col - col % sqrt;

        for (int r = rowStart; r < rowStart + sqrt; r++) {
            for (int c = colStart; c < colStart + sqrt; c++) {
                if (board[r][c] == num) {
                    return false;
                }
            }
        }
        return true;
    }
}
```

iterate over the row

→ coln limiter

check if no. already exists in that row

↳ if number is already present, reject it.

iterate over the coln.

for every col in the row

↳ if no. is already present, reject it

sqrt for the formula

]– determine start of subgrid

//iterate over a subgrid

// if number is present in subgrid, reject it.

as the

of the su

check the w

for the presen

number.

we called for (row, col) = ( 5, 6 )

① It will check entire 5th row for presence of that no.

② It will check entire 6th coln for presence of that no.

③ It will use

r = 5 - 5%3
= 3

c = 6 - 6%3
= 6

i.e. (3, 6)

e starting point
bgrid & then
hole subgrid
nce of that