Backtracking Problems

Agenda

1. N- Queens
2. Sudoku - Solver
3. Word Break Problem

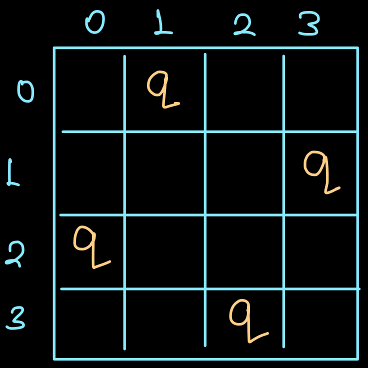
Hello Everyone
very special Good Evening
to All of you
We will start
from 9:06 PM

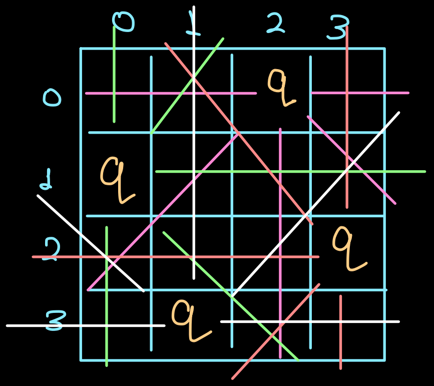
N- Queen:

Given N*N chess board, Place N-queens in these chess board and print the possibility:

N=4,

4*4 →

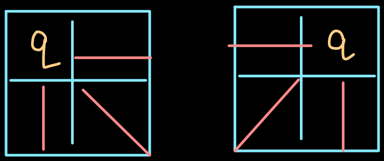




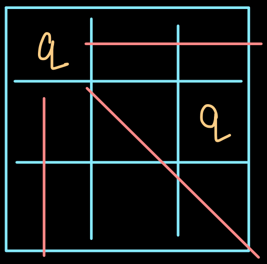
No two queens are killing to each other

N=1

N=2

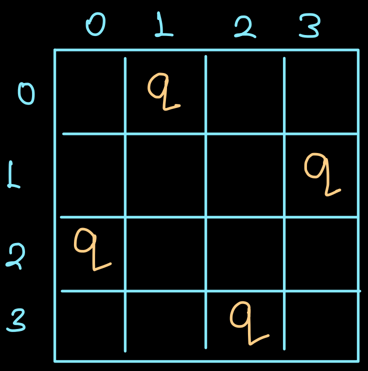


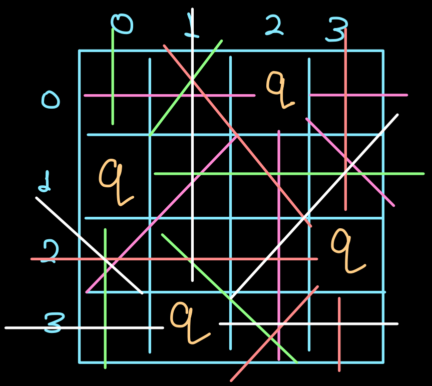
N=3



Imp: For N-queen problem value of $N \geq 4$

Observation!

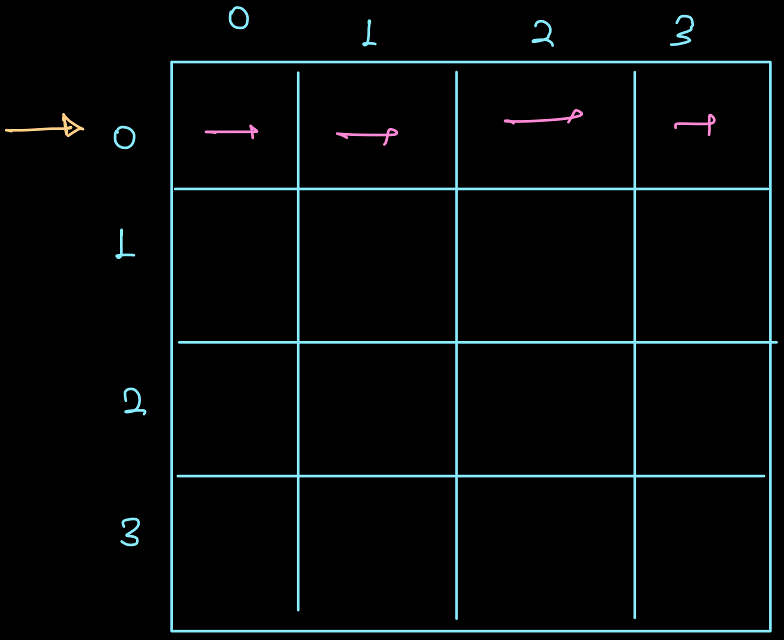


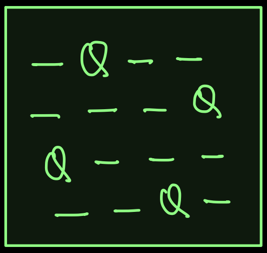


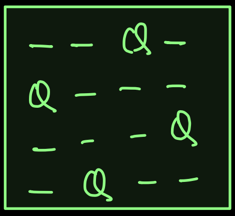
#1. Each Row contains one queen.

#2. One queen have N- cell option

N=4





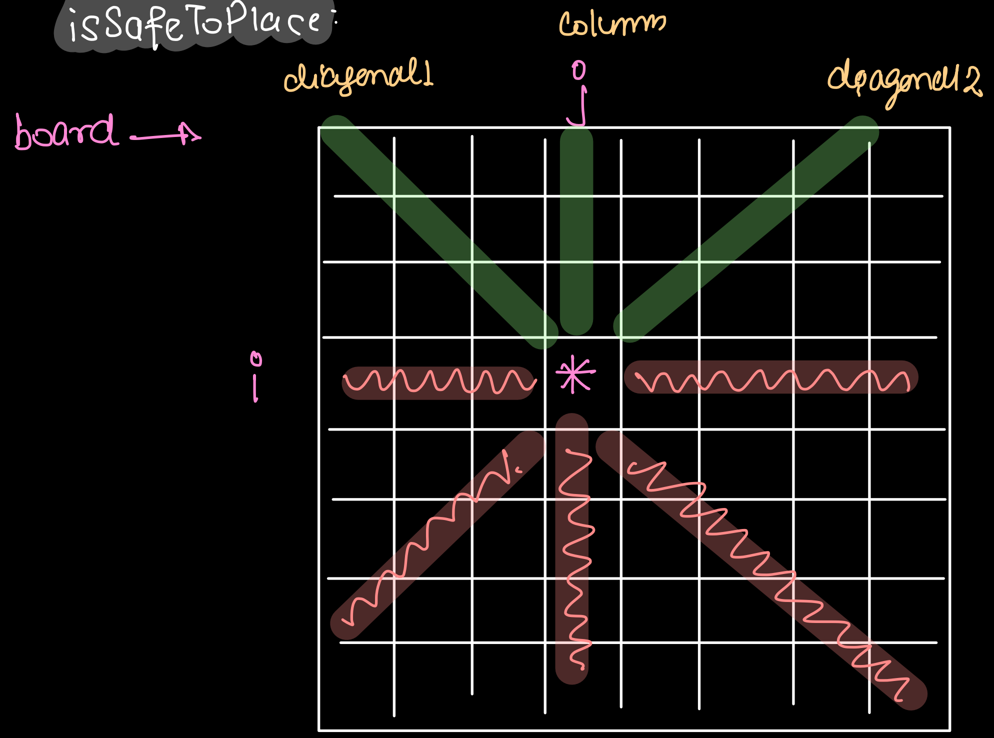


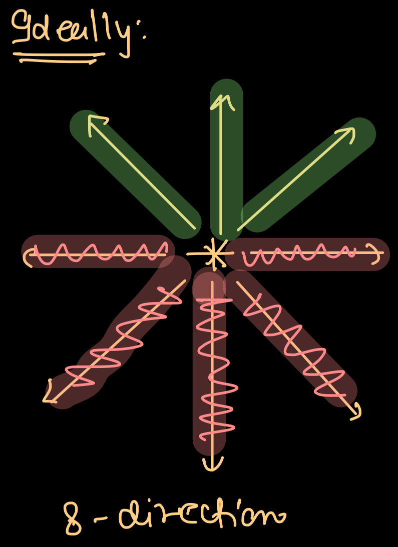
Recursive Syntax : board → char[][]

Row No → int ;

```
void   N-queens(char[][] board, int i){
    if( i== board.size()){
        print(board);  ⟶ 2D array printing
        return;
    }

    for( int j=0;  j< board.size(); j++){
        if( isSafeToPlace (board, i,j)) {
            board[i][j]= 'Q';
            N-Queens(board, i+1);

            board[i][j] = '-';
        }
    }
}
```

isSafeToPlace:





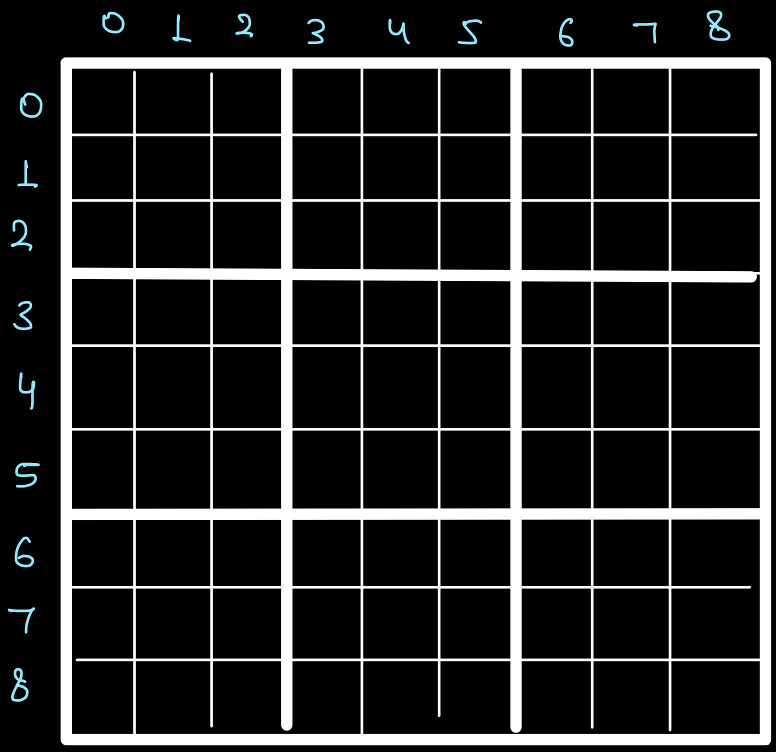
```
booleam isSafeToPlace ( char[][] board, int i, int j) {
    int n= board.length;
    // current colum checking
    for(int r=0; r< i; r++) {
        if( board[r][j] == 'Q') {
            return false;
        }
    }
    // diagonal 1
    for(int r=i-1, c=j-1; r>=0 && c>=0; r--, c--) {
        if( board[r][c] == 'Q') {
            return false;
        }
    }
    // digonal 2
    for(int r=i-1, c=j+1; r>=0 && c<n ; r--, c++) {
        if( board[r][c] == 'Q') {
            return false;
        }
    }
    return true;
}
```

T.C: $O(N!)$

S.C: $O(N)$

Sudoku Solver:

You are given an 9x9 board, in which the cells contains numbers from 1 - 9.
You need to check the below conditions,
1. Each row must contain the numbers from 1 to 9 w/o repititions.
2. Each col must contain the numbers from 1 to 9 w/o repititions.
3. Each block of size 3 * 3 should contain all numbers from 1 to 9 w/o repitition.



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

board[i][j] = = 0
$\rightarrow$ blank cell

i = 0, j = 2

$\vdash$ option of number for cell

ideally $\rightarrow$ 1 to 9

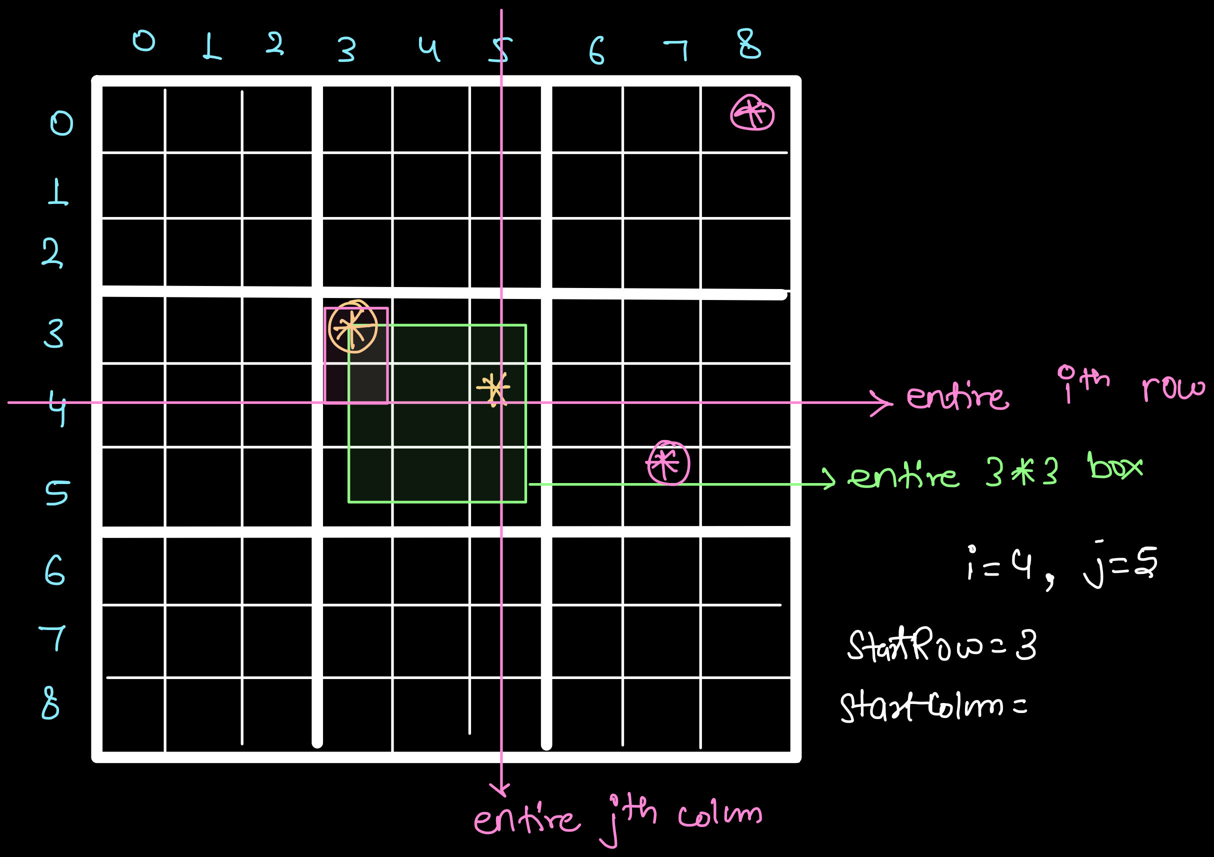
move Recursively on every cell and try to solve all possiblility for blank cell

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

Make an array of pairs for empty cell

(0,2)	0,3	0,5	0,6	0,7	0,8	1,1

isSafeToPlace:



$i=4, j=5$

StartRow = 3

Start Colm =

⇒ Row No — Remainder after division 3

⇒ $4 - 4\%3$

⇒ $4 - 1 = \boxed{3}$

Col·no — Remainder after division 3

⇒ $5 - 5\%3$

⇒ $5 - 2 = \boxed{3}$

Start Row = $i - i\%3$

Start Col = $j - j\%3$

$i=0, j=8$

$r = 0 - 0\%3 = 0$

$c = 8 - 8\%3 = 6$ ✓

$i=5, j=7$

$r = 5 - 5\%3 = 3$

$c = 7 - 7\%3 = 6$ ✓

$i=6, j=2$

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

$c = 2 - 2\%3 = 0$ ✓

$i=8, j=3$

$r = 8 - 8\%3 = 6$

$c = 3 - 3\%3 = 3$ ✓

$j=5, j=1$

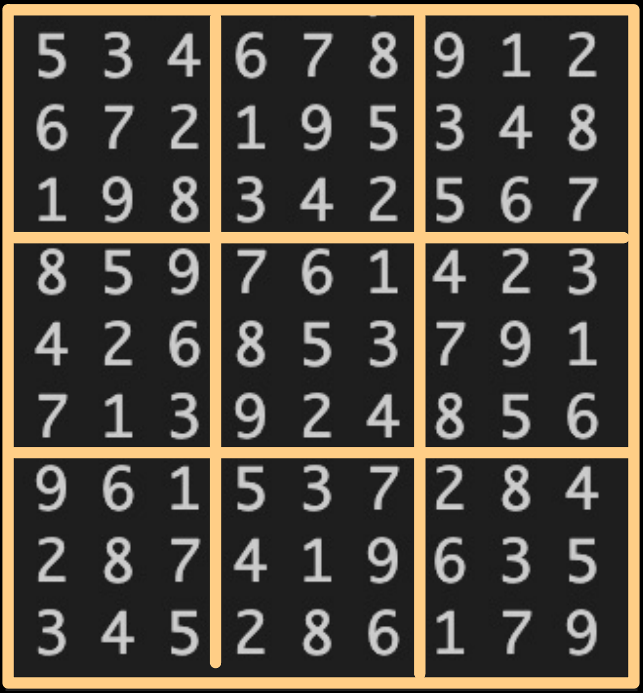
$r = 5 - 5\%3 = 3$

$c = 1 - 1\%3 = 0$ ✓

$i=8, j=8$

$r = 8 - 8\%3 = 6$

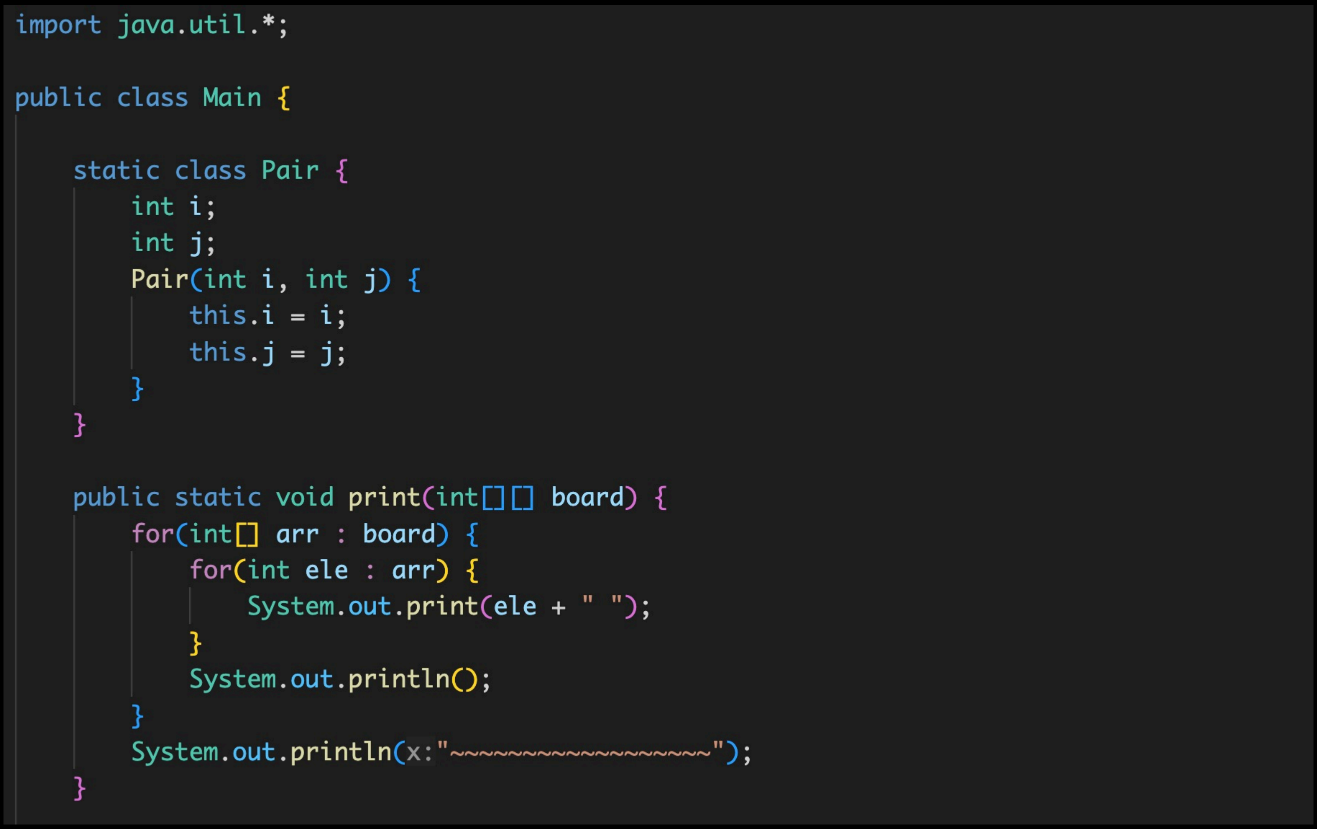
$c = 8 - 8\%3 = 6$ ✓



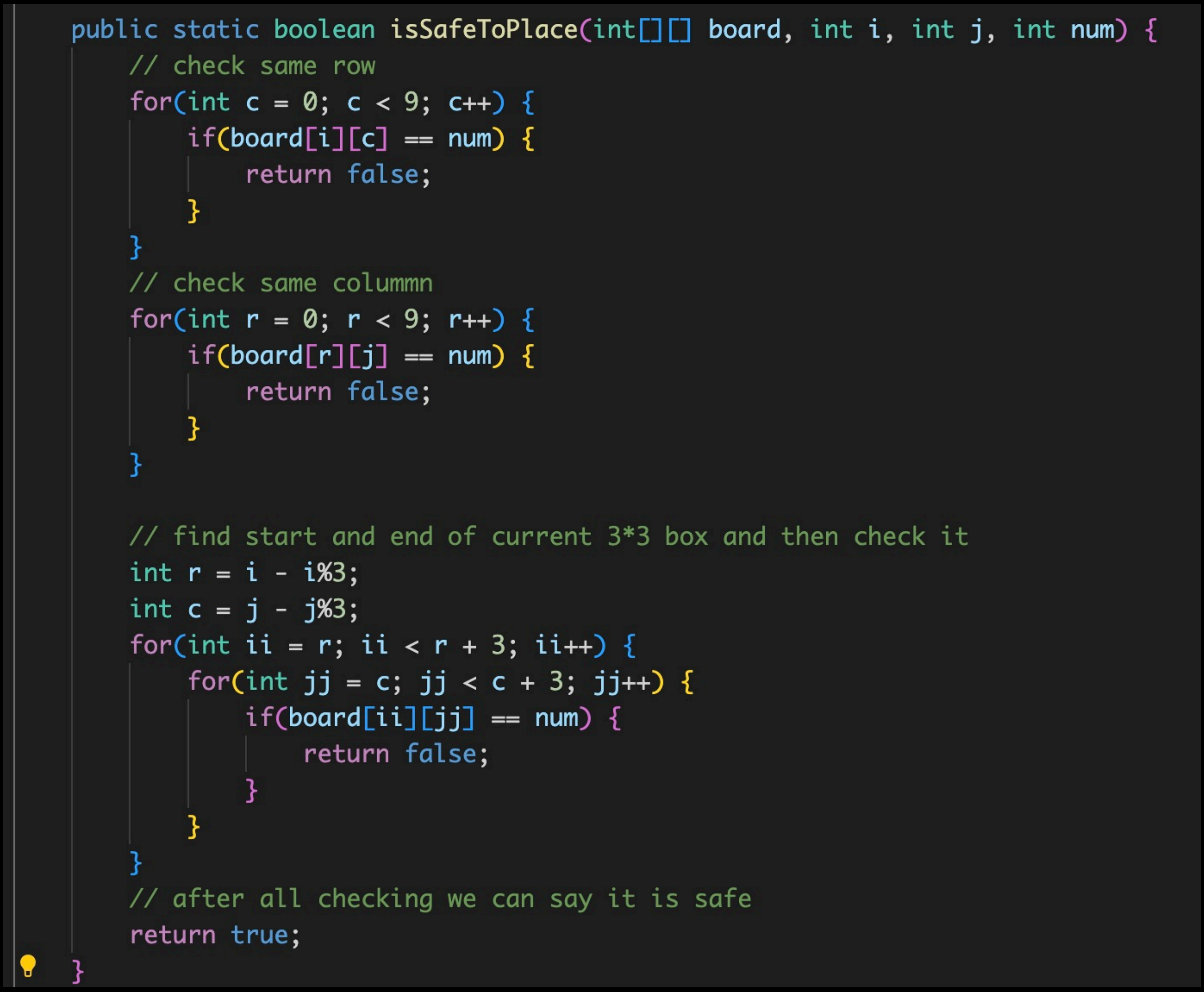
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

It is correct

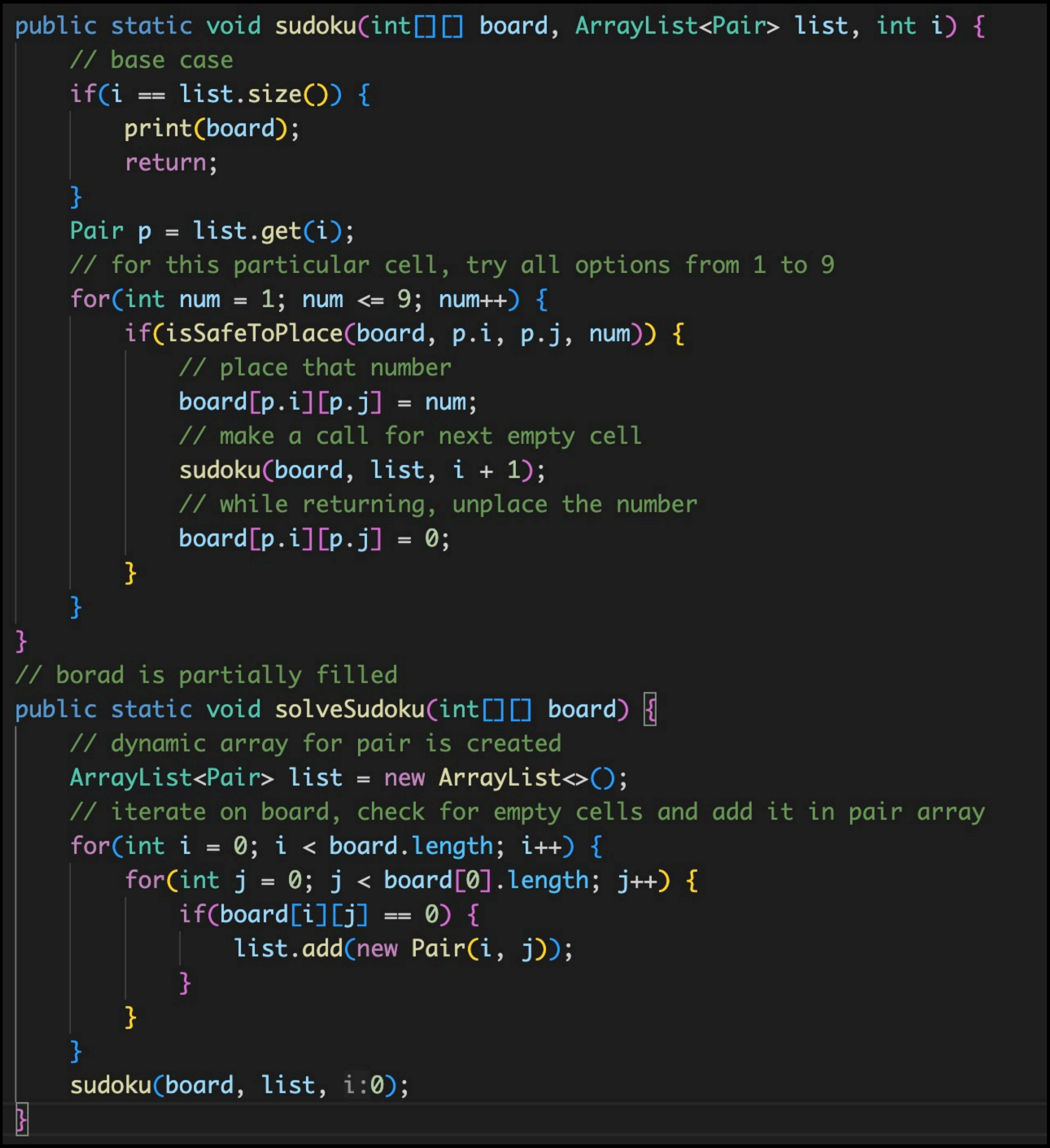
$\underbrace{10:51 - 11:00}_{\text{Break}}$ pm



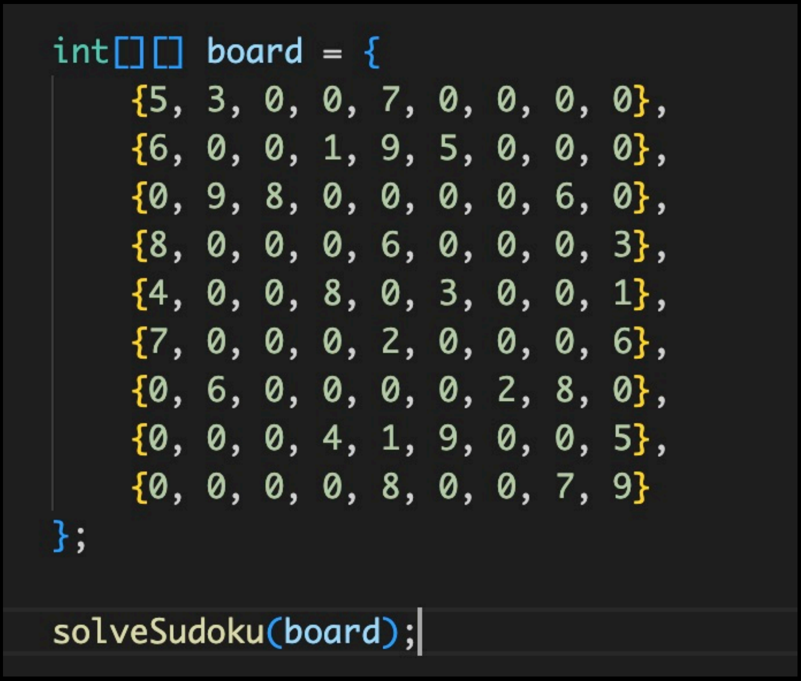
```java
import java.util.*;

public class Main {

    static class Pair {
        int i;
        int j;
        Pair(int i, int j) {
            this.i = i;
            this.j = j;
        }
    }

    public static void print(int[][] board) {
        for(int[] arr : board) {
            for(int ele : arr) {
                System.out.print(ele + " ");
            }
            System.out.println();
        }
        System.out.println(x:"~~~~~~~~~~~~~~~~~~~");
    }
```



```java
    public static boolean isSafeToPlace(int[][] board, int i, int j, int num) {
        // check same row
        for(int c = 0; c < 9; c++) {
            if(board[i][c] == num) {
                return false;
            }
        }
        // check same colummn
        for(int r = 0; r < 9; r++) {
            if(board[r][j] == num) {
                return false;
            }
        }

        // find start and end of current 3*3 box and then check it
        int r = i - i%3;
        int c = j - j%3;
        for(int ii = r; ii < r + 3; ii++) {
            for(int jj = c; jj < c + 3; jj++) {
                if(board[ii][jj] == num) {
                    return false;
                }
            }
        }
        // after all checking we can say it is safe
        return true;
    }
```



```java
public static void sudoku(int[][] board, ArrayList<Pair> list, int i) {
    // base case
    if(i == list.size()) {
        print(board);
        return;
    }
    Pair p = list.get(i);
    // for this particular cell, try all options from 1 to 9
    for(int num = 1; num <= 9; num++) {
        if(isSafeToPlace(board, p.i, p.j, num)) {
            // place that number
            board[p.i][p.j] = num;
            // make a call for next empty cell
            sudoku(board, list, i + 1);
            // while returning, unplace the number
            board[p.i][p.j] = 0;
        }
    }
}
// borad is partially filled
public static void solveSudoku(int[][] board) {
    // dynamic array for pair is created
    ArrayList<Pair> list = new ArrayList<>();
    // iterate on board, check for empty cells and add it in pair array
    for(int i = 0; i < board.length; i++) {
        for(int j = 0; j < board[0].length; j++) {
            if(board[i][j] == 0) {
                list.add(new Pair(i, j));
            }
        }
    }
    sudoku(board, list, i:0);
}
```



```java
int[][] board = {
    {5, 3, 0, 0, 7, 0, 0, 0, 0},
    {6, 0, 0, 1, 9, 5, 0, 0, 0},
    {0, 9, 8, 0, 0, 0, 0, 6, 0},
    {8, 0, 0, 0, 6, 0, 0, 0, 3},
    {4, 0, 0, 8, 0, 3, 0, 0, 1},
    {7, 0, 0, 0, 2, 0, 0, 0, 6},
    {0, 6, 0, 0, 0, 0, 2, 8, 0},
    {0, 0, 0, 4, 1, 9, 0, 0, 5},
    {0, 0, 0, 0, 8, 0, 0, 7, 9}
};

solveSudoku(board);
```

Word Break:

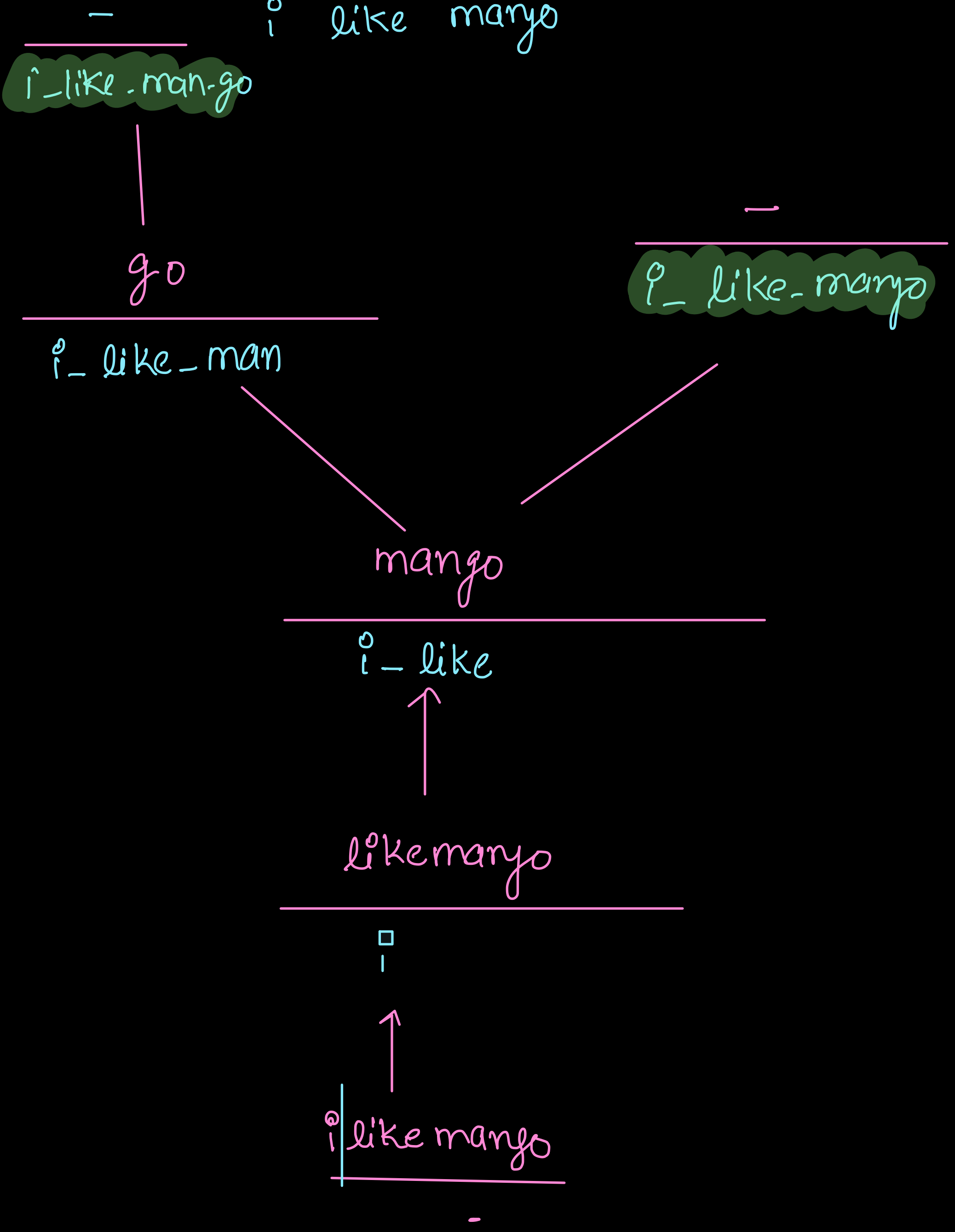
Dictionary of some word is given and one string is also given. Make the sentences from given string by using word of Dictionary.

Dictionary: ["i", "like", "man", "go", "mango"]

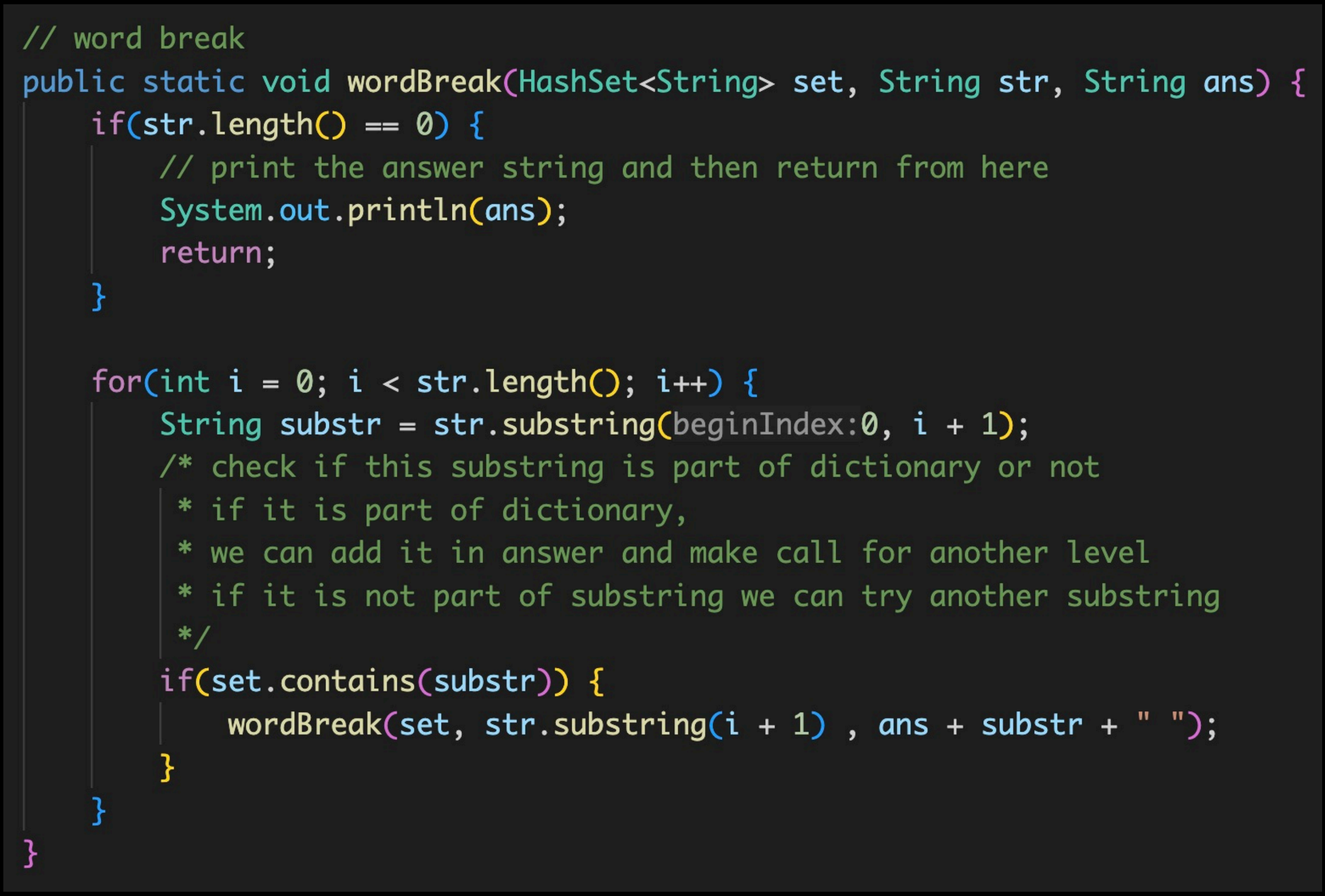
String str= "ilikemango"

i like man go

i like mango



(Hashset)
Dictionary
["i"
"like"
"mango"
"man"
"go"
]



```java
// word break
public static void wordBreak(HashSet<String> set, String str, String ans) {
    if(str.length() == 0) {
        // print the answer string and then return from here
        System.out.println(ans);
        return;
    }

    for(int i = 0; i < str.length(); i++) {
        String substr = str.substring(beginIndex:0, i + 1);
        /* check if this substring is part of dictionary or not
         * if it is part of dictionary,
         * we can add it in answer and make call for another level
         * if it is not part of substring we can try another substring
         */
        if(set.contains(substr)) {
            wordBreak(set, str.substring(i + 1) , ans + substr + " ");
        }
    }
}
```

Ques : T.C : O(n!)

S.C : O(n)