Backtracking Assignment 2 done by N S K K K Naga Jayanth

https://leetcode.com/problems/n-queens/

```
class Solution {
    public List<List<String>> solveNQueens(int n) {

        char [][] board = new char[n][n];
        for(int i = 0; i < board.length; i++) {
            for(int j = 0; j < board.length; j++) {
                board[i][j] = '.';
            }
        }

        List<List<String>> ans = new ArrayList<>();
        queen(board, 0, ans);
        return ans;
    }

    static void queen(char[][] board, int row, List<List<String>> list) {

        if(row == board.length) {

            list.add(construct(board));
            return;
        }

        for (int col = 0; col < board.length; col++) {
            if(isSafe(board, row, col)) {
                board[row][col] = 'Q';
                queen(board, row + 1, list);
                board[row][col] = '.';
            }
        }
    }

    static List<String> construct(char[][] board) {

        List<String> internal = new ArrayList<>();
        for (int i = 0; i < board.length; i++) {
            String row = new String(board[i]);
            internal.add(row);
        }
        return internal;
    }

    static boolean isSafe(char[][] board, int row, int col) {

        //for checking vertical row
        for (int i = 0; i < row; i++) {
            if(board[i][col] == 'Q') {
                return false;
            }
        }

        //for checking left diagonal

        int maxLeft = Math.min(row, col);

        for (int i = 1; i <= maxLeft; i++) {
            if(board[row - i][col - i] == 'Q') {
                return false;
            }
        }
        //for checking right diagonal

        int maxRight = Math.min(row, board.length - 1 - col);

        for (int i = 1; i <= maxRight; i++) {
            if(board[row - i][col + i] == 'Q') {
                return false;
            }
        }

        return true;
    }
}
```

https://leetcode.com/problems/sudoku-solver/

```
class Solution {

    public boolean isValid(char[][] board, int row, int col, char c) {
        for (int i = 0; i < 9; i++) {
            if (board[i][col] == c) return false;
            if (board[row][i] == c) return false;
            if (board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c) return false;
        }
        return true;
    }

    public boolean solve(char[][] board) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                if (board[i][j] == '.') {
                    for (char c = '1'; c <= '9'; c++) {
                        if (isValid(board, i, j, c)) {
                            board[i][j] = c;
                            if (solve(board)) return true;
                            else board[i][j] = '.';
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }

    public void solveSudoku(char[][] board) {
        solve(board);
    }
}
```

https://www.codechef.com/problems/BPHC03

```
import java.util.*;

class Codechef {
    static int[] dx = {0, 0, -1, 1};
    static int[] dy = {-1, 1, 0, 0};
    static int count = 0;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[][] maze = new int[n][n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                maze[i][j] = scanner.nextInt();
            }
        }

        boolean[][] visited = new boolean[n][n];
        dfs(maze, 0, 0, n, visited);

        System.out.println(count);
    }

    public static void dfs(int[][] maze, int x, int y, int n, boolean[][] visited) {
        if (x == n - 1 && y == n - 1) {
            count++;
            return;
        }

        visited[x][y] = true;

        for (int i = 0; i < 4; i++) {
            int newX = x + dx[i];
            int newY = y + dy[i];

            if (newX >= 0 && newX < n && newY >= 0 && newY < n && maze[newX][newY] == 0 &&
!visited[newX][newY]) {
                dfs(maze, newX, newY, n, visited);
            }
        }

        visited[x][y] = false;
    }
```

```
}
```

https://leetcode.com/problems/combination-sum/

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<>();
        List<Integer> currentSubset = new ArrayList<>();

        findCombination(0, target, candidates, currentSubset, result);
        return result;
    }

    public static void findCombination(int index, int target, int[] candidates, List<Integer>
currentSubset, List<List<Integer>> result) {
        // checks if we have explored all the elements of array
        if(index == candidates.length) {
            if(target == 0) {
                result.add(new ArrayList<>(currentSubset));
            }
            return;
        }

        if(candidates[index] <= target) {
            currentSubset.add(candidates[index]);

            // After adding the element of curr index, iterate the left path until the base condition is met
            findCombination(index, target - candidates[index], candidates, currentSubset, result);

            // this is required because when the above recursion call
            // is executed then the Data structure still has curr index element so we need to remove it
            currentSubset.remove(currentSubset.size() - 1);
        }
        // check for the next element of array
        findCombination(index + 1, target, candidates, currentSubset, result);
    }
}
```

https://leetcode.com/problems/combination-sum-ii/

```
class Solution {
    List<List<Integer>> result=new ArrayList<>();
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        Arrays.sort(candidates);
        helper(candidates,target,new ArrayList<Integer>(),0);
        return result;
    }
    public void helper(int[] arr,int target,List<Integer> templist,int start){
        if(target==0){
            result.add(new ArrayList<>(templist));
        }else if(target>0){
            for(int i=start;i<arr.length;i++){
                if (i > start && arr[i] == arr[i - 1]) {
                    continue;
                }
                templist.add(arr[i]);
                helper(arr,target-arr[i],templist,i+1);
                templist.remove(templist.size()-1);
            }
        }
    }
}
```