**Backtracking in Recursion**

**Java**

**Print all Permutations**

Time complexity - O(n\*n!)

public class Recursion3 {

public static void printPermutation(String str, int idx, String perm) {

if(str.length() == 0) {

System.out.println(perm);

return;

}

for(int i=0; i<str.length(); i++) {

char currChar = str.charAt(i);

String newStr = str.substring(0, i) + str.substring(i+1);

printPermutation(newStr, idx+1, perm+currChar);

}

}

public static void main(String args[]) {

String str = "abc";

printPermutation(str, 0, "");

}

}

**N-Queens**

Time complexity - O(n^n)

class Solution {

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

//horizontal

for(int j=0; j<board.length; j++) {

if(board[row][j] == 'Q') {

return false;

}

}

//vertical

for(int i=0; i<board.length; i++) {

if(board[i][col] == 'Q') {

return false;

}

}

//upper left

int r = row;

for(int c=col; c>=0 && r>=0; c--, r--) {

if(board[r][c] == 'Q') {

return false;

}

}

//upper right

r = row;

for(int c=col; c<board.length && r>=0; r--, c++) {

if(board[r][c] == 'Q') {

return false;

}

}

//lower left

r = row;

for(int c=col; c>=0 && r<board.length; r++, c--) {

if(board[r][c] == 'Q') {

return false;

}

}

//lower right

for(int c=col; c<board.length && r<board.length; c++, r++) {

if(board[r][c] == 'Q') {

return false;

}

}

return true;

}

public void saveBoard(char[][] board, List<List<String>> allBoards) {

String row = "";

List<String> newBoard = new ArrayList<>();

for(int i=0; i<board.length; i++) {

row = "";

for(int j=0; j<board[0].length; j++) {

if(board[i][j] == 'Q')

row += 'Q';

else

row += '.';

}

newBoard.add(row);

}

allBoards.add(newBoard);

}

public void helper(char[][] board, List<List<String>> allBoards, int col) {

if(col == board.length) {

saveBoard(board, allBoards);

return;

}

for(int row=0; row<board.length; row++) {

if(isSafe(row, col, board)) {

board[row][col] = 'Q';

helper(board, allBoards, col+1);

board[row][col] = '.';

}

}

}

public List<List<String>> solveNQueens(int n) {

List<List<String>> allBoards = new ArrayList<>();

char[][] board = new char[n][n];

helper(board, allBoards, 0);

return allBoards;

}

}

**Homework Problems**

1. <https://leetcode.com/problems/permutations/> (Similar to print Permutations)
2. <https://www.hackerrank.com/challenges/knightl-on-chessboard/problem> (Similar to N-Queens)
3. <https://leetcode.com/problems/sudoku-solver/> (Will be discussed in next class)