BACKTRACKING

1. Backtracking on Arrays

```java
import java.util.Scanner;
public class BacktrackingOnArrays {
    public static void changeArr (int arr[], int i, int val) {
        // BASE CASE
        if (i == arr.length) {
            printArr(arr);
            return;
        }
        // RECURSION
        arr[i] = val;
        changeArr(arr, i+1, val+1);  // function call
        arr[i] = arr[i] - 2;  // backtracking
    }
    public static void printArr (int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int arr[] = new int [8];
        changeArr(arr, 0, 1);
        printArr(arr);
    }
}
```

2. Grid Ways

```java
public class GridWays {
    public static int gridWays (int i, int j, int n, int m) {
        // i = row, j = column, n = no. of rows, m = no. of columns
        // Base Case
        if (i == n-1 && j == m-1) {     // Condition for last cell
            return 1;
        } else if (i == n || j == n) {  // Boundary cross condition
            return 0;
        }
        int w1 = gridWays(i+1, j, n, m);
        int w2 = gridWays(i, j+1, n, m);
        return w1 + w2;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter no. of Rows: ");
        int n = sc.nextInt();
        System.out.print("Enter no. of Columns: ");
        int m = sc.nextInt();
        System.out.print("Total no. of Grid ways to acquire Target: ");
        System.out.println(gridWays(0, 0, n, m));                          // (0, 0) to (n, m)
    }
}
```

3. Keypad Combinations

```java
public class KeypadCombinations {
    public static void bfs (int pos, int len, StringBuilder sb, String D) {
        if (pos == len) {
            System.out.print(sb.toString()+ ", ");
        } else {
            char[] letters = L [Character.getNumericValue(D.charAt(pos))];
            for (int i = 0; i < letters.length; i++) {
                bfs(pos+1, len, new StringBuilder(sb).append(letters[i]), D);
            }
        }
    }
    final static char[][] L = { {}, {}, {'a', 'b', 'c'}, {'d', 'e', 'f'},
                                {'g', 'h', 'i'}, {'j', 'k', 'l'}, {'m', 'n', 'o'},
                                {'p', 'q', 'r', 's'}, {'t', 'u', 'v'}, {'w', 'x', 'y', 'z'} };
    public static void letterCombinations (String D) {
        int len = D.length();
        if (len == 0) {
            System.out.print(" ");
            return;
        }
        bfs(0, len, new StringBuilder(), D);
    }
    public static void main(String[] args) {
        // Scanner sc = new Scanner (System.in);
        // System.out.print("Enter Digits: ");
        // String digit = sc.next();
        // System.out.print("Letters Combinations are: ");
        letterCombinations("23");
    }
}
```

4. Permutations

```java
public class Permutations {
    public static void findPermutation (String str, String ans) {
        // BASE CASE
        if (str.length() == 0) {
            System.out.println(ans);
            return;                                    // Time Complexity - O(n*n!)
        }           // RECURSION
        for (int i = 0; i < str.length(); i++) {
            char curr = str.charAt(i);
            String NewStr = str.substring(0, i) + str.substring(i+1);
            findPermutation(NewStr, ans + curr);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter String: ");
        String str = sc.next();
        System.out.println("Permutations of String are: ");
        findPermutation(str, " ");
    }
}
```

5. Knights Tour

```java
public class KnightsTour {
    static int N = 8;
    public static boolean isSafe (int x, int y, int sol[][]) {
        return (x >= 0 && x < N && y >= 0 && y < N && sol[x][y] == -1);
    }
    public static void printSolution (int sol[][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                System.out.print(sol[x][y] + " ");
            }
            System.out.println();
        }
    }
    public static boolean solveKTUtil (int x,int y,int moveI,int sol[][],int xMove[],int yMove[]){
        int k, next_x, next_y;
        if (moveI == N*N) {
            return true;
        }
        for (k = 0; k < 8; k++) {
            next_x = x + xMove[k];
            next_y = y + yMove[k];
            if (isSafe(next_x, next_y, sol)) {
                sol[next_x][next_y] = moveI;
                if (solveKTUtil(next_x, next_y, moveI + 1, sol, xMove, yMove)) {
                    return true;
                } else {
                    sol[next_x][next_y] = -1; // Backtracking
                }
            }
        }
        return false;
    }
    public static boolean solveKT() {
        int sol[][] = new int [8][8];
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                sol[x][y] = -1;
            }
        }
        int xMove[] = {2, 1, -1, -2, -2, -1, 1, 2};
        int yMove[] = {1, 2, 2, 1, -1, -2, -2, -1};
        // As the knight starts from cell(0, 0)
        sol[0][0] = 0;
        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
            System.out.println("Solution doesn't Exist.");
            return false;
        } else {
            printSolution(sol);
        }
        return true;
    }
    public static void main(String[] args) {
        solveKT();
    }
}
```

6. N Queens

```java
public class Nqueens {
    public static boolean isSafe (char board[][], int row, int col) {
        for (int i=row-1; i>=0; i--) {      // Vertical Up
            if (board[i][col] == 'Q') {
                return false;
            }
        }          // Diagonal Left Up
        for (int i=row-1, j=col-1; i>=0 && j>=0; i--, j--) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }          // Diagonal Right Up
        for(int i=row-1, j=col+1; i>=0 && j<board.length; i--, j++) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        return true;
    }
    public static void printBoard (char board[][]) {
        System.out.println("------Chess Board------");
        for(int i=0; i<board.length; i++) {
            for(int j=0; j<board.length; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static void nQueens (char board[][], int row) {
        if (row == board.length) {    // Base Case
            printBoard(board);
            return;
        }
        for(int j=0; j<board.length; j++) {             // Column Loop
            if (isSafe(board, row, j)) {
                board[row][j] = 'Q';
                nQueens(board, row+1);                  // Function Calling
                board[row][j] = 'x';                    // Backtracking
            }
        }
    }
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter value of N (N*N Board): ");
        int n = sc.nextInt();
        System.out.println("N Queens All Possible Solutions are: ");
        char board[][] = new char[n][n];
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                board[i][j] = 'x';
            }
        }
        nQueens(board, 0);
    }
}
```

7. N Queens Count

```java
public class NQueensCount {
    public static boolean isSafe (char board[][], int row, int col) {
        for (int i=row-1; i>=0; i--) {      // Vertical Up
            if (board[i][col] == 'Q') {
                return false;
            }
        }
        for (int i=row-1, j=col-1; i>=0 && j>=0; i--, j--) {  // Diagonal Left Up
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        for(int i=row-1, j=col+1; i>=0 && j<board.length; i--, j++) {   // Diagonal Right Up
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        return true;
    }
    public static void printBoard (char board[][]) {
        System.out.println("------Chess Board------");
        for(int i=0; i<board.length; i++) {
            for(int j=0; j<board.length; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static void nQueens (char board[][], int row) {
        if (row == board.length)      // Base Case
            count++;
            return;
        for(int j=0; j<board.length; j++) {             // Column Loop
            if (isSafe(board, row, j)) {
                board[row][j] = 'Q';
                nQueens(board, row+1);                  // Function Calling
                board[row][j] = 'x';                    // Backtracking
            }
        }
    }
    static int count = 0;
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter value of N (N*N Board): ");
        int n = sc.nextInt();
        char board[][] = new char[n][n];
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                board[i][j] = 'x';
            }
        }
        nQueens(board, 0);
        System.out.println("Total ways to solve N Queens: " + count);
    }
}
```

8. N Queens One Solution

```java
import java.util.Scanner;

public class NQueensOneSolution {
    public static boolean isSafe (char board[][], int row, int col) {
        // Vertical Up
        for (int i=row-1; i>=0; i--) {
            if (board[i][col] == 'Q') {
                return false;
            }
        }
        // Diagonal Left Up
        for (int i=row-1, j=col-1; i>=0 && j>=0; i--, j--) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        // Diagonal Right Up
        for(int i=row-1, j=col+1; i>=0 && j<board.length; i--, j++) {
            if (board[i][j] == 'Q') {
                return false;
            }
        }
        return true;
    }
    public static boolean nQueens (char board[][], int row) {
        if (row == board.length) {
            count++;
            return true;
        }
        for (int j=0; j<board.length; j++) {
            if (isSafe(board, row, j)) {
                board[row][j] = 'Q';
                if (nQueens(board, row+1)) {
                    return true;
                }
                board[row][j] = 'x';
            }
        }
        return false;
    }
    public static void printBoard (char board[][]) {
        System.out.println("------Chess Board------");
        for(int i=0; i<board.length; i++) {
            for(int j=0; j<board.length; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }
    static int count = 0;
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter value of N (N*N Board): ");
        int n = sc.nextInt();
        char board[][] = new char[n][n];
```

```
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                board[i][j] = 'x';
            }
        }
        if (nQueens(board, 0)) {
            System.out.println("Solution is Possible");
            printBoard(board);
        } else {
            System.out.println("Solution is not Possible");
        }
    }
}
```

9. Subsets

```
import java.util.Scanner;

public class Subsets {
    public static void findSubsets (String str, String ans,int i) {
        if (i == str.length()) {
            if (ans.length() == 0) {
                System.out.print("null");
            } else {
                System.out.print(ans);                  // Time Complexity - O(n*2^n)
            }                                           // Space Complexity - O(n)
            return;
        }
        findSubsets(str, ans + str.charAt(i), i+1); // YES
        findSubsets(str, ans, i+1); // NO
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter String: ");
        String str = sc.next();
        System.out.println("Subsets of String are: ");
        findSubsets(str, " ", 0);
    }
}
```

10.Rat in Maze

```java
public class RatInMaze {
    public static void printSolution(int sol[][]) {
        for (int i = 0; i < sol.length; i++) {
            for (int j = 0; j < sol.length; j++) {
                System.out.print(" " + sol[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static boolean solveMazeUtil(int maze[][], int x, int y, int sol[][]) {
        if (x == maze.length - 1 && y == maze.length - 1 && maze[x][y] == 1) {
            sol[x][y] = 1;
            return true;
        }
        // check if maze[x][y] is valid
        if (isSafe(maze, x, y) == true) {
            if (sol[x][y] == 1) {
                return false;
            }
            sol[x][y] = 1;
            if (solveMazeUtil(maze, x + 1, y, sol)) {
                return true;
            }
            if (solveMazeUtil(maze, x, y + 1, sol)) {
                return true;
            }
            sol[x][y] = 0;
            return false;
        }
        return false;
    }

    public static boolean isSafe(int maze[][], int x, int y) {
        // if (x, y) outside maze then return false
        return (x >= 0 && x < maze.length && y >= 0 && y < maze.length && maze[x][y] == 1);
    }
    public static boolean solveMaze(int maze[][]) {
        int N = maze.length;
        int sol[][] = new int[N][N];
        if (solveMazeUtil(maze, 0, 0, sol) == false) {
            System.out.println("Solution doesn't Exist");
            return false;
        }
        printSolution(sol);
        return true;
    }
    public static void main(String[] args) {
        int maze[][] = { {1, 0, 0, 0},
                         {1, 1, 0, 1},
                         {0, 1, 0, 0},
                         {1, 1, 1, 1} };
        solveMaze(maze);
    }
}
```

11.Sudoku

```java
public class Sudoku {
    public static boolean isSafe (int sudoku[][], int row, int col, int digit) {
        // Column
        for (int i = 0; i <= 8; i++) {
            if (sudoku[i][col] == digit) {
                return false;
            }
        }
        // Row
        for (int j = 0; j <= 8; j++) {
            if (sudoku[row][j] == digit) {
                return false;
            }
        }
        // Grid
        int sr = (row/3) * 3;
        int sc = (col/3) * 3;
        // 3*3 Grid
        for (int i = sr; i < sr+3; i++) {
            for (int j = sc; j < sc+3; j++) {
                if (sudoku[i][j] == digit) {
                    return false;
                }
            }
        }
        return true;
    }
    public static boolean SudokuSolver (int sudoku[][], int row, int col) {
        // Base Case
        if (row == 9) {
            return true;
        }
        // Recursion
        int nextRow = row, nextCol = col + 1;
        if (col + 1 == 9) {
            nextRow = row + 1;
            nextCol = 0;
        }
        if (sudoku[row][col] != 0) {
            return SudokuSolver(sudoku, nextRow, nextCol);
        }
        for (int digit = 1; digit <= 9; digit++) {
            if (isSafe(sudoku, row, col, digit)) {
                sudoku[row][col] = digit;
                if (SudokuSolver(sudoku, nextRow, nextCol)) {
                    return true;
                }
                sudoku[row][col] = 0;
            }
        }
        return false;
    }
```

```
    public static void printSudoku (int Sudoku[][]) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                System.out.print(Sudoku[i][j]+ " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int sudoku[][] = {{ 0, 0, 8, 0, 0, 0, 0, 0, 0},
        {4, 9, 0, 1, 5, 7, 0, 0, 2},
        {0, 0, 3, 0, 0, 4, 1, 9, 0},
        {1, 8, 5, 0, 6, 0, 0, 2, 0},
        {0, 0, 0, 0, 2, 0, 0, 6, 0},
        {9, 6, 0, 4, 0, 5, 3, 0, 0},
        {0, 3, 0, 0, 7, 2, 0, 0, 4},
        {0, 4, 9, 0, 3, 0, 0, 5, 7},
        {8, 2, 7, 0, 0, 9, 0, 1, 3} };
        if (SudokuSolver(sudoku, 0, 0)) {
            System.out.println("-----Solution Exists-----");
            printSudoku(sudoku);
        } else {
            System.out.println("-----Solution Doesn't Exists-----");
        }
    }
}
```