```
package Love_Babbar;

public class BackTracking {
    class BackTracking {
        // Rat in a maze Problem
        class Solution {
            public static ArrayList<String> findPath(int[][] m, int n) {
                // Your code here
                ArrayList<String> res = new ArrayList<String>();
                int dx[] = new int[] { 0, 1, 0, -1 };
                int dy[] = new int[] { -1, 0, 1, 0 };
                char ds[] = new char[] { 'L', 'D', 'R', 'U' };
                boolean vis[][] = new boolean[n][n];
                if (m[0][0] == 0) {
                    return res;
                }
                dfs(0, 0, vis, m, n, res, "", dx, dy, ds);
                return res;

            }

            public static void dfs(int row, int col, boolean vis[][], int matrix[][], int n,
ArrayList<String> res,
                    String temp, int dx[], int dy[], char ds[]) {
                if (row == n - 1 && col == n - 1) {
                    res.add(temp);
                    return;
                }
                // System.out.println("Path Taken "+temp+" Position:"+"["+row+","+col+"]");
                vis[row][col] = true;
                for (int i = 0; i < 4; i++) {
                    int nr = row + dx[i];
                    int nc = col + dy[i];
                    if (nr >= 0 && nr < n && nc >= 0 && nc < n && matrix[nr][nc] == 1 &&
vis[nr][nc] == false) {
                        dfs(nr, nc, vis, matrix, n, res, temp + ds[i], dx, dy, ds);
                    }
                }
                vis[row][col] = false;
            }
        }
        // Printing all solutions in N-Queen
        class Solution {
            public static List<List<String>> solveNQueens(int n) {
                char[][] board = new char[n][n];
                for (int i = 0; i < n; i++)
                    for (int j = 0; j < n; j++)
                        board[i][j] = '.';
                List<List<String>> res = new ArrayList<List<String>>();
```

```
    dfs(0, board, res);
    return res;
}

static boolean validate(char[][] board, int row, int col) {
    int duprow = row;
    int dupcol = col;
    while (row >= 0 && col >= 0) {
        if (board[row][col] == 'Q')
            return false;
        row--;
        col--;
    }
    row = duprow;
    col = dupcol;
    while (col >= 0) {
        if (board[row][col] == 'Q')
            return false;
        col--;
    }
    row = duprow;
    col = dupcol;
    while (col >= 0 && row < board.length) {
        if (board[row][col] == 'Q')
            return false;
        col--;
        row++;
    }
    return true;
}

static void dfs(int col, char[][] board, List<List<String>> res) {
    if (col == board.length) {
        res.add(construct(board));
        return;
    }

    for (int row = 0; row < board.length; row++) {
        if (validate(board, row, col)) {
            board[row][col] = 'Q';
            dfs(col + 1, board, res);
            board[row][col] = '.';
        }
    }
}

static List<String> construct(char[][] board) {
    List<String> res = new LinkedList<String>();
    for (int i = 0; i < board.length; i++) {
```

```
                String s = new String(board[i]);
                res.add(s);
            }
            return res;
        }
    }
    // Word Break Problem using Backtracking
    // Remove Invalid Parentheses
    // Sudoku Solver
    // M Coloring Problem
    // Print all palindromic partitions of a string
    // Subset Sum Problem
    // The Knight's tour problem
    // Tug of War
    // Find shortest safe route in a path with landmines
    // Combinational Sum
    // Find Maximum number possible by doing at-most K swaps
    // Print all permutations of a string
    // Find if there is a path of more than k length from a source
    // Longest Possible Route in a Matrix with Hurdles
    class Solution {

        public static int longestPath(int[][] mat, int n, int m, int xs, int ys, int xd, int yd) {
            // code here
            int dx[] = new int[] { -1, 0, 1, 0 };
            int dy[] = new int[] { 0, 1, 0, -1 };
            boolean vis[][] = new boolean[n][m];
            int pathval[] = new int[1];
            if (mat[xs][ys] == 0 || mat[xd][yd] == 0) {
                return -1;
            }
            pathval[0] = -1;
            f(xs, ys, mat, vis, n, m, dx, dy, xd, yd, pathval, 0);
            return pathval[0];
        }

        public static void f(int xs, int ys, int[][] mat, boolean[][] vis, int n, int m, int[] dx,
int[] dy, int xd,
                int yd, int[] p, int cp) {
            if (xs == xd && ys == yd) {
                if (p[0] < cp) {
                    p[0] = cp;
                }
                return;
            }

            vis[xs][ys] = true;
            for (int i = 0; i < 4; i++) {
                int nx = xs + dx[i];
```

```
                int ny = ys + dy[i];
                if (nx >= 0 && nx < n && ny >= 0 && ny < m && vis[nx][ny] == false &&
mat[nx][ny] == 1) {
                    f(nx, ny, mat, vis, n, m, dx, dy, xd, yd, p, (cp + 1));
                }
            }
            vis[xs][ys] = false;
            return;
        }

        public static int longestPath(int[][] mat, int n, int m, int xs, int ys, int xd, int yd) {
            // code here
            int dx[] = new int[] { 0, 1, 0, -1 };
            int dy[] = new int[] { -1, 0, 1, 0 };
            if (mat[xs][ys] == 0 || mat[xd][yd] == 0) {
                return -1;
            }
            boolean vis[][] = new boolean[n][m];
            int max = f(xs, ys, mat, vis, n, m, dx, dy, xd, yd, 0);
            return max;
        }

        public static int f(int xs, int ys, int[][] mat, boolean[][] vis, int n, int m, int[] dx, int[]
dy, int xd,
                int yd, int p) {
            if (xs == xd && ys == yd) {
                // System.out.println("DESX: "+xs+" DESY: "+ys+" Path: "+ p);
                return 0;
            }
            // System.out.println("X: "+xs+" Y: "+ys+" Path: "+ p);
            vis[xs][ys] = true;
            int path = Integer.MIN_VALUE;
            for (int i = 0; i < 4; i++) {
                int nx = xs + dx[i];
                int ny = ys + dy[i];
                if (nx >= 0 && nx < n && ny >= 0 && ny < m && vis[nx][ny] == false &&
mat[nx][ny] == 1) {
                    int c_path = 1 + f(nx, ny, mat, vis, n, m, dx, dy, xd, yd, p + 1);
                    path = Math.max(c_path, path);
                    // System.out.println("NEWX: "+nx+" NEWY: "+ny+" PathVal: "+ path);
                }
            }
            vis[xs][ys] = false;
            return path;
        }
    }
    // Print all possible paths from top left to bottom right of a mXn matrix
    // Partition of a set into K subsets with equal sum
    // Find the K-th Permutation Sequence of first N natural numbers
```

```
    }
}
```