**Rat in a Maze – GFG**

CODE :

import java.util.ArrayList;

public class fndPaths {

    static ArrayList<String> res;

    static int[][] visited;

    public static void findPaths(int[][] m, int row, int col, String s, int n) {

        if (row == n - 1 && col == n - 1) {

            res.add(s);

            return;

        } else {

            visited[row][col] = 1;

        }

        if (row + 1 < n && visited[row + 1][col] == 0 && m[row + 1][col] == 1) {

            findPaths(m, row + 1, col, s + "D", n); // DOWN

        }

        if (col - 1 >= 0 && visited[row][col - 1] == 0 && m[row][col - 1] == 1) {

            findPaths(m, row, col - 1, s + "L", n); // LEFT

        }

        if (col + 1 < n && visited[row][col + 1] == 0 && m[row][col + 1] == 1) {

            findPaths(m, row, col + 1, s + "R", n); // RIGHT

        }

        if (row - 1 >= 0 && visited[row - 1][col] == 0 && m[row - 1][col] == 1) {

            findPaths(m, row - 1, col, s + "U", n); // UP

        }

        visited[row][col] = 0;

    }

    public static ArrayList<String> findPath(int[][] m, int n) {

        visited = new int[n][n];

        res = new ArrayList<>();

        if (m[0][0] == 0) {

            return res;

        }

        String s = "";

        findPaths(m, 0, 0, s, n);

        return res;

    }

    public static void main(String[] args) {

        int[][] matrix = {

            {1, 0, 0},

            {1, 1, 0},

            {1, 1, 1}

        };

        int size = 3;

        ArrayList<String> paths = findPath(matrix, size);

        if (paths.isEmpty()) {

            System.out.println("No paths found.");

        } else {

            System.out.println("Paths found:");

            for (String path : paths) {

                System.out.println(path);

            }

        }

    }

}

OUTPUT :

A screenshot of a computer

Description automatically generated with medium confidence

**Combination Sum - LC : 39**

CODE :

import java.util.ArrayList;

import java.util.List;

public class Combination\_Sum {

    public static void main(String[] args) {

        Solution solution = new Solution();

        int[] candidates = {2, 3, 6, 7};

        int target = 7;

        List<List<Integer>> combinations = solution.combinationSum(candidates, target);

        System.out.println(combinations);

    }

}

class Solution {

    public void findCombinations(int index, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {

        if (index == arr.length) {

            if (target == 0) {

                ans.add(new ArrayList<>(ds));

            }

            return;

        }

        if (arr[index] <= target) {

            ds.add(arr[index]);

            findCombinations(index, arr, target - arr[index], ans, ds);

            ds.remove(ds.size() - 1);

        }

        findCombinations(index + 1, arr, target, ans, ds);

    }

    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        List<List<Integer>> ans = new ArrayList<>();

        findCombinations(0, candidates, target, ans, new ArrayList<>());

        return ans;

    }

}

OUTPUT :

A screen shot of a computer

Description automatically generated with medium confidence