L22 Rat in a Maze Problem - I

Monday, June 19, 2023 1:52 PM

Consider a rat placed at **(0, 0)** in a square matrix of order N * N. It has to reach the destination at **(N - 1, N - 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up), 'D'(down), 'L' (left), 'R' (right)**. Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell.

Example 1:

Input:
N = 4
m[][] = {{1, 0, 0, 0},
{1, 1, 0, 1},
{1, 1, 0, 0},
{0, 1, 1, 1}}

Output:
DDRDRR DRDDRR

→ Basically we finding all the path that take Rat reach Source to destination

Explanation:
The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

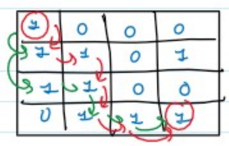
Example 2:

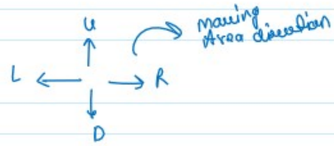
Input:
N = 2
m[][] = {{1, 0},
{1, 0}}

Output:
-1

Explanation:
No path exists and destination cell is blocked.

Thought Process





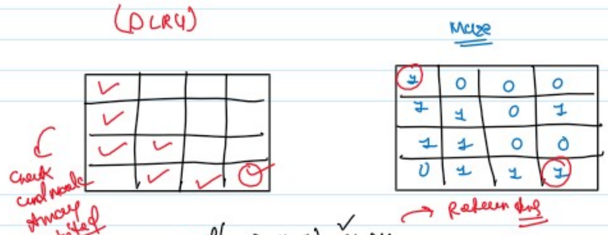
DDRDRR, DRDDRR → Answer is Lexicographical Order

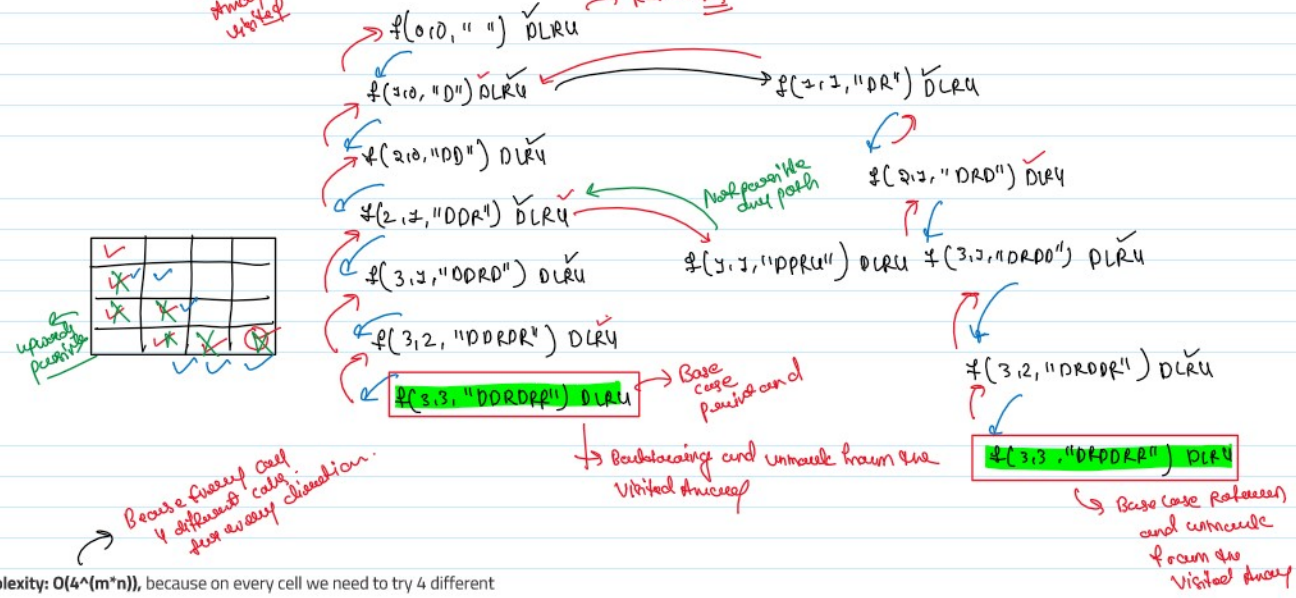
→ This type of problem we solved using Recursion

→ We follow the lexicographical order

(DLRU)

→ Maintaining the Visited Array





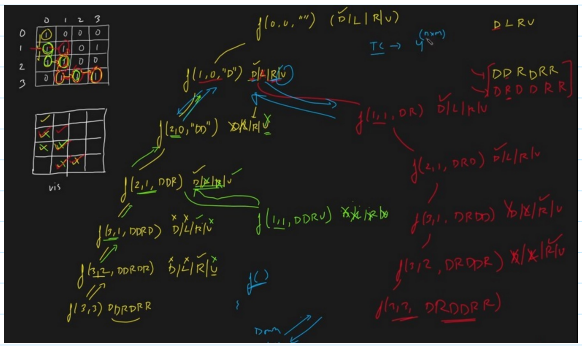
Time Complexity: O(4^(m*n)), because on every cell we need to try 4 different directions.

Space Complexity: O(m*n), Maximum Depth of the recursion tree(auxiliary space).

directions.

Space Complexity: **O(m*n)** ,Maximum Depth of the recursion tree(auxiliary space).

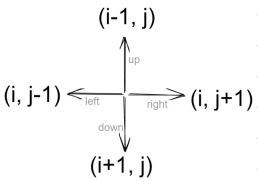
↳ maximum depth of the recursion



```python
from typing import List


class Solution:

    5 usages
    def findPathHelper(self, i: int, j: int, a: List[List[int]], n: int, ans: List[str], move: str,
                       vis: List[List[int]]):

        # Base Case
        if i == n - 1 and j == n - 1:
            ans.append(move)
            return

        # downward
        if i + 1 < n and not vis[i + 1][j] and a[i + 1][j] == 1:
            vis[i][j] = 1
            self.findPathHelper(i + 1, j, a, n, ans, move + 'D', vis)
            vis[i][j] = 0

        # left
        if j - 1 >= 0 and not vis[i][j - 1] and a[i][j - 1] == 1:
            vis[i][j] = 1
            self.findPathHelper(i, j - 1, a, n, ans, move + 'L', vis)
            vis[i][j] = 0

        # right
        if j + 1 < n and not vis[i][j + 1] and a[i][j + 1] == 1:
            vis[i][j] = 1
            self.findPathHelper(i, j + 1, a, n, ans, move + 'R', vis)
            vis[i][j] = 0

        # upward
        if i - 1 >= 0 and not vis[i - 1][j] and a[i - 1][j] == 1:
            vis[i][j] = 1
            self.findPathHelper(i - 1, j, a, n, ans, move + 'U', vis)
            vis[i][j] = 0

    def findPath(self, m: List[List[int]], n: int) -> List[str]:
        ans = []
        vis = [[0 for _ in range(n)] for _ in range(n)]

        if m[0][0] == 1:
            self.findPathHelper(0, 0, m, n, ans, "", vis)
        return ans
```

```java
import java.util.ArrayList;

public class L21_Rate_Maze_1 {
    public static void main(String[] args) {
        System.out.println("L21_Rate_Maze_1");
    }
}

no usages
class Solution17 {
    5 usages
    private static void solve(int i, int j, int a[][], int n, ArrayList < String > ans, String move,
                              int vis[][]) {

        if (i == n - 1 && j == n - 1) {
            ans.add(move);
            return;
        }

        // downward
        if (i + 1 < n && vis[i + 1][j] == 0 && a[i + 1][j] == 1) {
            vis[i][j] = 1;
            solve(i + 1, j, a, n, ans, move + 'D', vis);
            vis[i][j] = 0;
        }

        // left
        if (j - 1 >= 0 && vis[i][j - 1] == 0 && a[i][j - 1] == 1) {
            vis[i][j] = 1;
            solve(i, j - 1, a, n, ans, move + 'L', vis);
            vis[i][j] = 0;
        }

        // right
        if (j + 1 < n && vis[i][j + 1] == 0 && a[i][j + 1] == 1) {
            vis[i][j] = 1;
            solve(i, j + 1, a, n, ans, move + 'R', vis);
            vis[i][j] = 0;
        }

        // upward
        if (i - 1 >= 0 && vis[i - 1][j] == 0 && a[i - 1][j] == 1) {
            vis[i][j] = 1;
            solve(i - 1, j, a, n, ans, move + 'U', vis);
            vis[i][j] = 0;
        }
    }
    no usages
    public static ArrayList< String > findPath(int[][] m, int n) {
        int vis[][] = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vis[i][j] = 0;
            }
        }

        ArrayList < String > ans = new ArrayList < > ();
        if (m[0][0] == 1) solve(0, 0, n, n, ans, "", vis);
        return ans;
    }
}
```

But, writing an individual code for every direction is a lengthy process therefore we truncate the 4 “if statements” into a single for loop using the following approach.



	D	L	R	U
di[ ]	+1	+0	+0	-1
dj[ ]	+0	-1	+1	+0

```python
# Optimized Code
from typing import List

class Solution:
    def solve(self, i: int, j: int, a: List[List[int]], n: int, ans: List[str], move: str, vis: List[List[int]], di: List[int], dj: List[int]):
        if i == n - 1 and j == n - 1:
            ans.append(move)
            return
        dir = "DLRU"
        for ind in range(4):
            nexti = i + di[ind]
            nextj = j + dj[ind]
            if nexti >= 0 and nextj >= 0 and nexti < n and nextj < n and not vis[nexti][nextj] and a[nexti][nextj] == 1:
                vis[i][j] = 1
                self.solve(nexti, nextj, a, n, ans,
                           move + dir[ind], vis, di, dj)
                vis[i][j] = 0


    def findPath(self, m: List[List[int]], n: int) -> List[str]:
        ans = []
        vis = [[0 for _ in range(n)] for _ in range(n)]
        di = [+1, 0, 0, -1]
        dj = [0, -1, 1, 0]
        if m[0][0] == 1:
            self.solve(0, 0, m, n, ans, "", vis, di, dj)
        return ans
```

→ 4 direction

→ x and y coordinates

↳ More Better Approach

```java
//More Optimized Code
// m is the given matrix and n is the order of matrix
class Solution18 {
    private static void solve(int i, int j, int a[][], int n, ArrayList < String > ans, String move,
                              int vis[][], int di[], int dj[]) {
        if (i == n - 1 && j == n - 1) {
            ans.add(move);
            return;
        }
        String dir = "DLRU";
        for (int ind = 0; ind < 4; ind++) {
            int nexti = i + di[ind];
            int nextj = j + dj[ind];
            if (nexti >= 0 && nextj >= 0 && nexti < n && nextj < n &&
                    vis[nexti][nextj] == 0 && a[nexti][nextj] == 1) {

                vis[i][j] = 1;
                solve(nexti, nextj, a, n, ans, move + dir.charAt(ind), vis, di, dj);
                vis[i][j] = 0;

            }
        }
    }
    public static ArrayList < String > findPath(int[][] m, int n) {
        int vis[][] = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vis[i][j] = 0;
            }
        }
        int di[] = {
                +1,
                0,
                0,
                -1
        };
        int dj[] = {
                0,
                -1,
                1,
                0
        };
        ArrayList < String > ans = new ArrayList < > ();
        if (m[0][0] == 1) solve(0, 0, m, n, ans, "", vis, di, dj);
        return ans;
    }
}
```