Rat in a Maze

Rat in a Maze

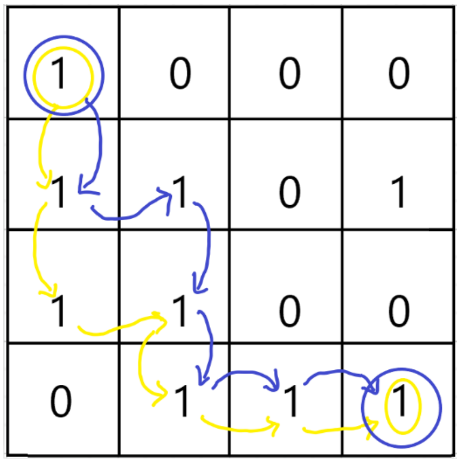
Consider a rat placed at **(0, 0)** in a square matrix of order **N * N**. It has to reach the destination at **(N – 1, N – 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **‘U'(up)**, **‘D'(down)**, **‘L’ (left)**,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

Note: In a path, no cell can be visited more than one time.

Print the answer in lexicographical(sorted) order

Examples:

```
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



Explanation:
```



1	0	0	0
1	1	0	1
1	1	0	0
0	1	1	1

```
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:

```
 No path exists and the destination cell is blocked.
```

Solution

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution 1: Recursion

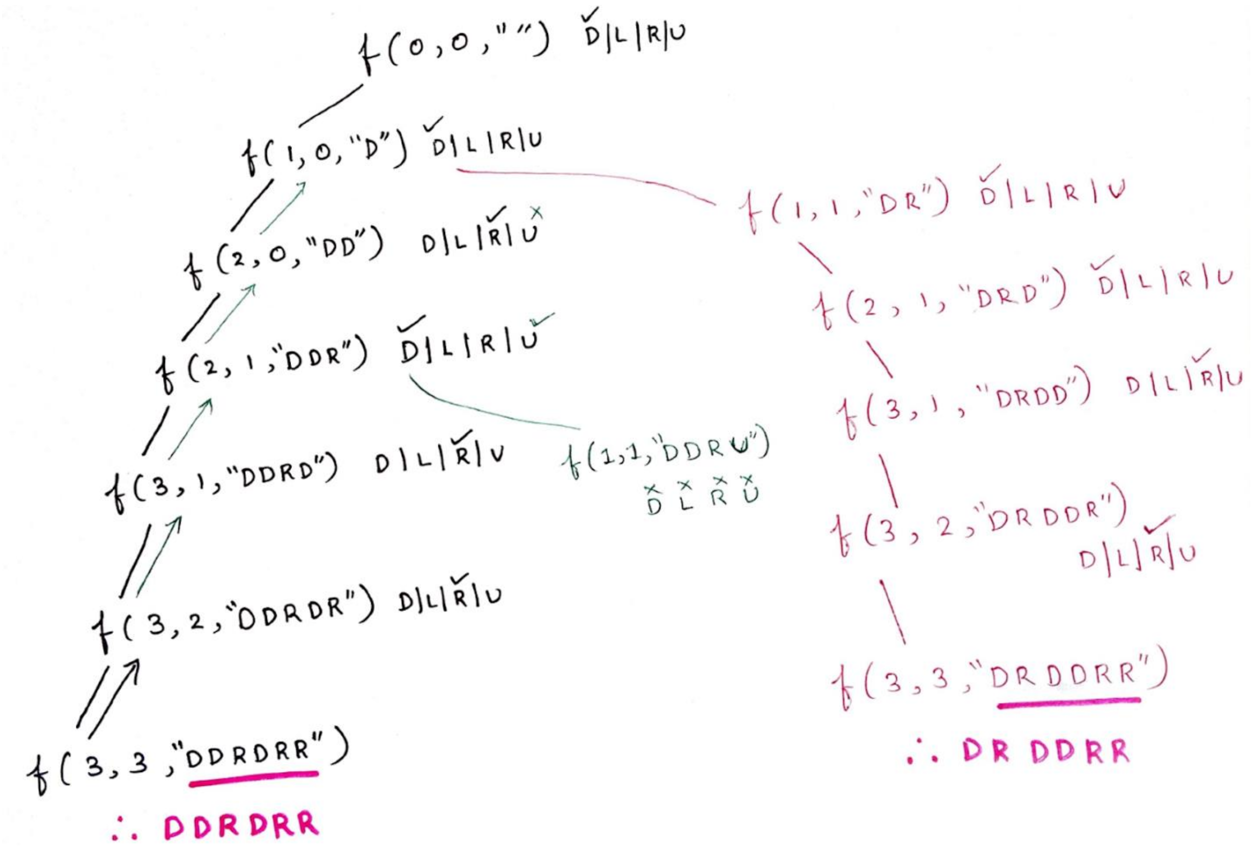
Intuition:

The best way to solve such problems is using recursion.

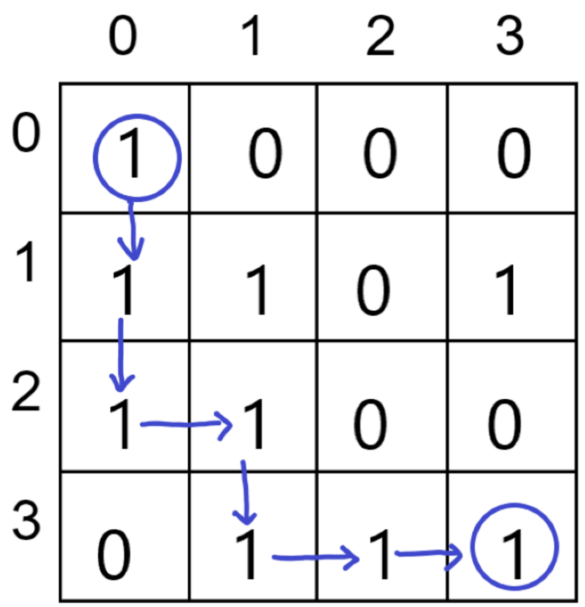
Approach:

- Start at the source(0,0) with an empty string and try every possible path i.e upwards**(U)**, downwards**(D)**, leftwards**(L)** and rightwards**(R)**.
- As the **answer** should be in lexicographical order so it's better to try the **directions** in lexicographical order i.e (D,L,R,U)
- Declare a 2D-array named visited because the question states that a single cell should be included only once in the path,so it's important to keep track of the visited cells in a particular path.
- If a cell is in path, mark it in the visited array.
- Also keep a check of the **"out of bound"** conditions while going in a particular direction in the matrix.
- Whenever you reach the destination**(n,n)** it's very important to get back as shown in the recursion tree.
- While getting back, keep on unmarking the visited array for the respective direction.Also check whether there is a different path possible while getting back and if yes, then mark that cell in the visited array.

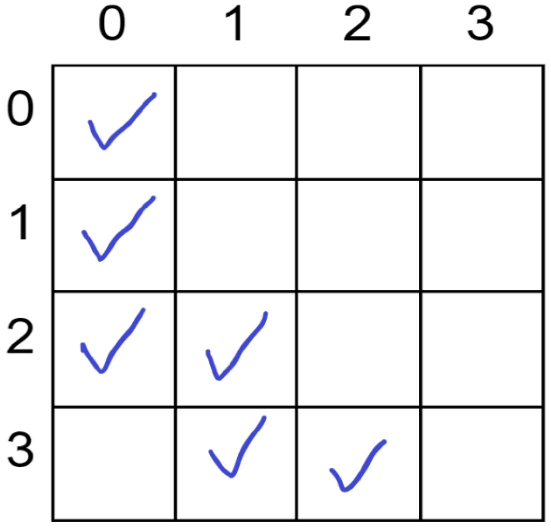
Recursive tree:



For "DDRDRR" :



	0	1	2	3
0	1	0	0	0
1	1	1	0	1
2	1	1	0	0
3	0	1	1	1



	0	1	2	3
0	✓			
1	✓			
2	✓	✓		
3		✓	✓	

Visited

Code:

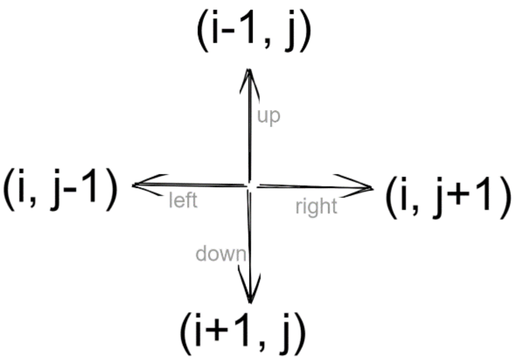
```
class Solution {

    public static void helper(int row,int col,int[][] m,int n,boolean[][]
visited,ArrayList<String> res,String path)

    {

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

        {

            if(m[row][col]==1)

            res.add(new String(path));
            return;
        }
        if(m[row][col]==0)
        {
            return;
        }
        visited[row][col]=true;
        if(row<n-1)
        {
            if(!visited[row+1][col])
            helper(row+1,col,m,n,visited,res,path+"D");

        }
        if(col>0)
        {
            if(!visited[row][col-1])
            helper(row,col-1,m,n,visited,res,path+"L");
        }
        if(col<n-1)
        {
            if(!visited[row][col+1])
            helper(row,col+1,m,n,visited,res,path+"R");
        }
        if(row>0)
        {
            if(!visited[row-1][col])
            helper(row-1,col,m,n,visited,res,path+"U");
```

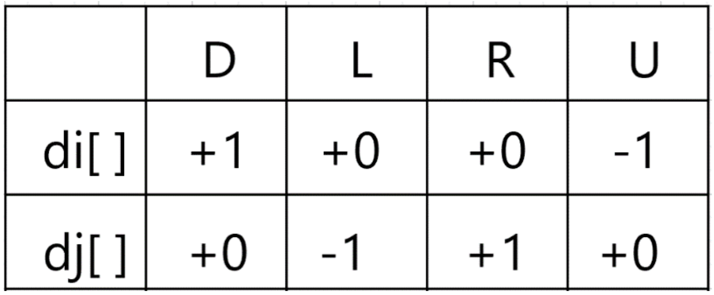
```
        }
        visited[row][col]=false;
    }
    public static ArrayList<String> findPath(int[][] m, int n) {
       boolean[][] visited=new boolean[n][n];
       ArrayList<String> res=new ArrayList<String>();
       helper(0,0,m,n,visited,res,"");
       return res;
    }
}
```

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).

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

Code:

```
class Solution {
  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;
  }
}
```

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).