Backtracking 2

Agenda

- Print paths in staircase problem
- Print all paths from source to destination
- Shortest path in matrix with huddles

Question 1

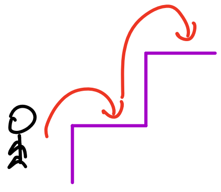
You are climbing a staircase and it takes N steps to reach the top.

Each time, you can take either 1 or 2 steps.

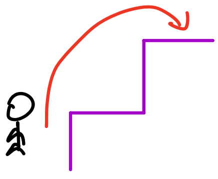
In how many distinct ways you can climb.

Return all distinct ways in lexographical order.

N = 2

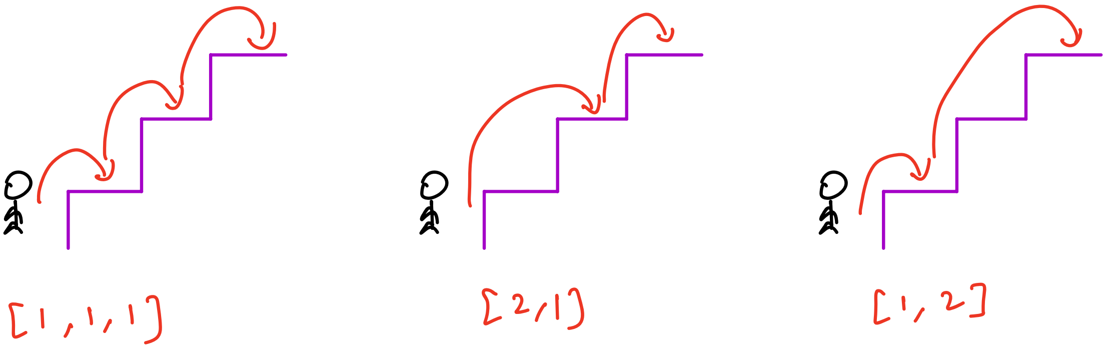


[1, 1]



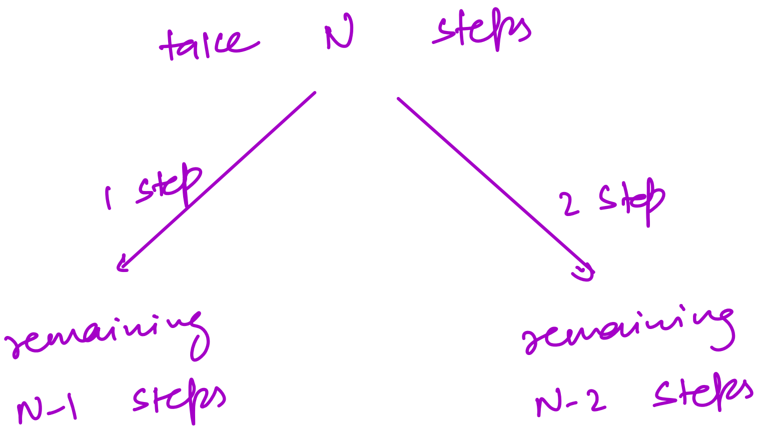
[2]

N=3



o/p : [[1,1,1], [1,2], [2,1]]

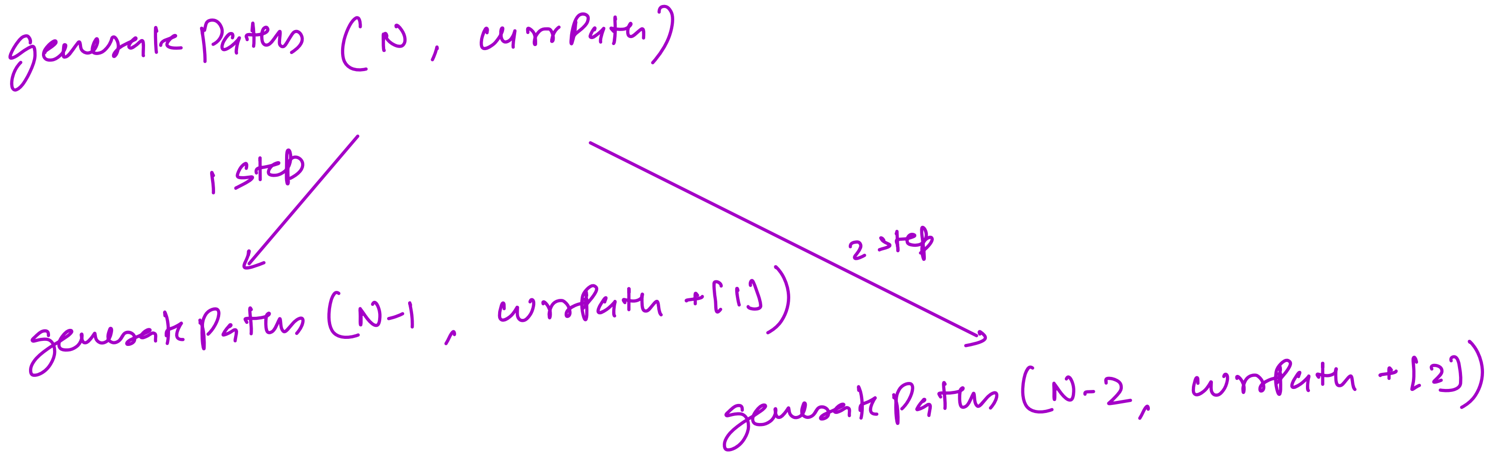
Solution



if N becomes 0, we have reached top and store the current in answer.

how to keep lexographically sorted ?

→ always choose step 1 befor step 2.



generatePaths (N, currPath)

1 step

2 step

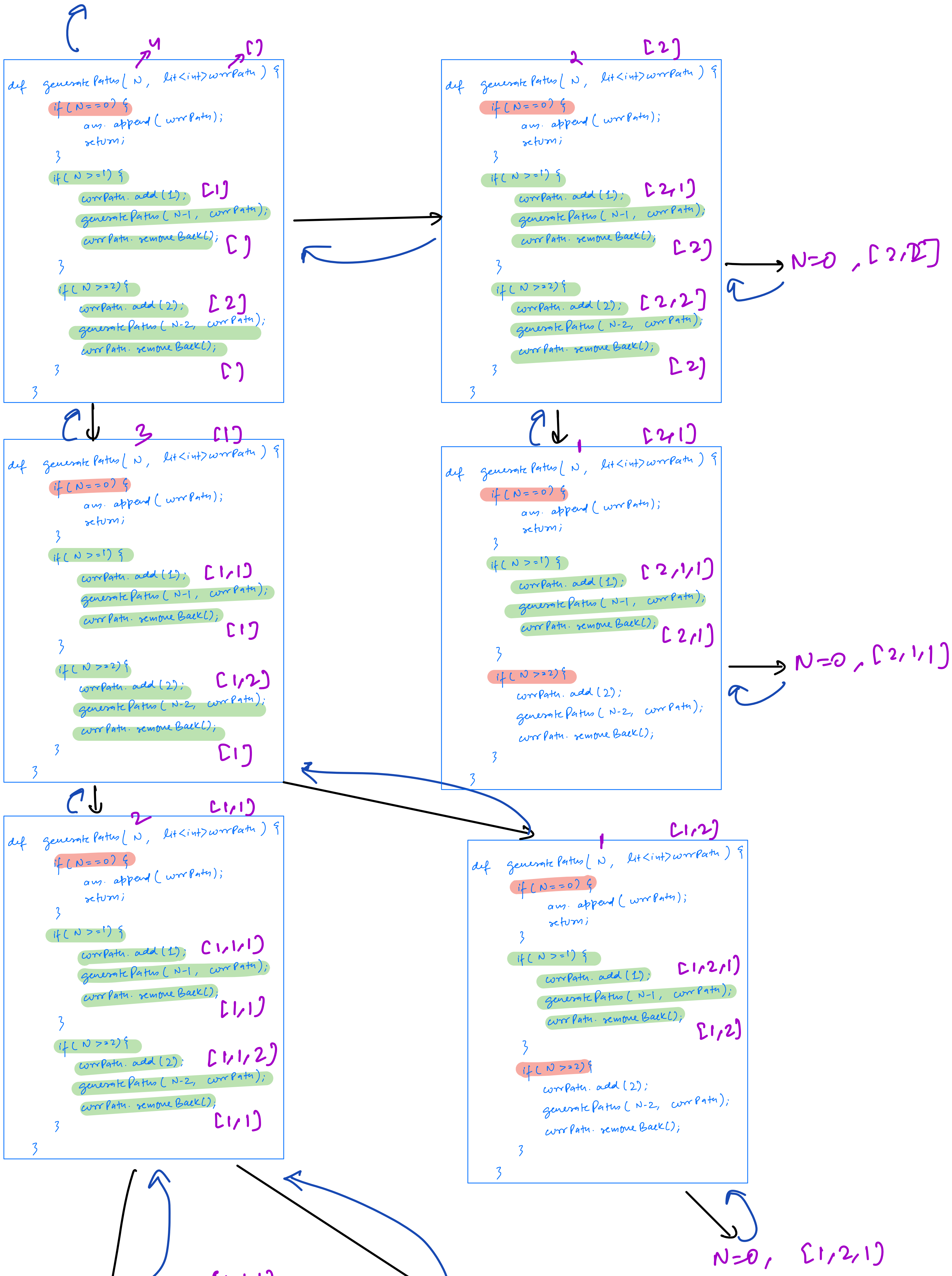
generatePaths (N-1, currPath + [1])

generatePaths (N-2, currPath + [2])

Code

```
    List< List<int>>   ans;
def   generatePaths( N,   lit<int>currPath ) {
        if (N==0) {
            ans. append( currPath);
            return;
        }
        if ( N >=1) {
            currPath. add (1);
            generatePaths ( N-1,  currPath);
            currPath. removeBack();

        }
        if ( N >=2) {
            currPath. add (2);
            generatePaths ( N-2,  currPath);
            currPath. removeBack();
        }
```

}



```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

```
def generatePaths( N, list<int>currPath ) {
    if ( N==0 ) {
        ans.append( currPath );
        return;
    }
    if ( N>=1 ) {
        currPath.add(1);
        generatePaths( N-1, currPath );
        currPath.removeBack();
    }
    if ( N>=2 ) {
        currPath.add(2);
        generatePaths( N-2, currPath );
        currPath.removeBack();
    }
}
```

1 [1,1,1]

```
def  generatePaths( N,  list<int> currPath ) {
        if ( N==0) {
                ans. append ( currPath);
                return;
        }
        if ( N >=1) {
                currPath. add (1);
                generatePaths ( N-1,  currPath);
                currPath. removeBack();
        }
        if ( N >=2) {
                currPath. add (2);
                generatePaths ( N-2,  currPath);
                currPath. removeBack();
        }
}
```

[1,1,1,1]

[1,1,1]

0 [1,1,1,1]

```
def  generatePaths( N,  list<int> currPath ) {
        if ( N==0) {
                ans. append ( currPath);
                return;
        }
        if ( N >=1) {
                currPath. add (1);
                generatePaths ( N-1,  currPath);
                currPath. removeBack();
        }
        if ( N >=2) {
                currPath. add (2);
                generatePaths ( N-2,  currPath);
                currPath. removeBack();
        }
}
```

0 [1,1,2]

```
def  generatePaths( N,  list<int> currPath ) {
        if ( N==0) {
                ans. append ( currPath);
                return;
        }
        if ( N >=1) {
                currPath. add (1);
                generatePaths ( N-1,  currPath);
                currPath. removeBack();
        }
        if ( N >=2) {
                currPath. add (2);
                generatePaths ( N-2,  currPath);
                currPath. removeBack();
        }
}
```

ans = [[1,1,1,1], [1,1,2], [1; 2,1], [2,1,1], [2,2]]

TC = $O(2^N)$

SC = $O(N)$

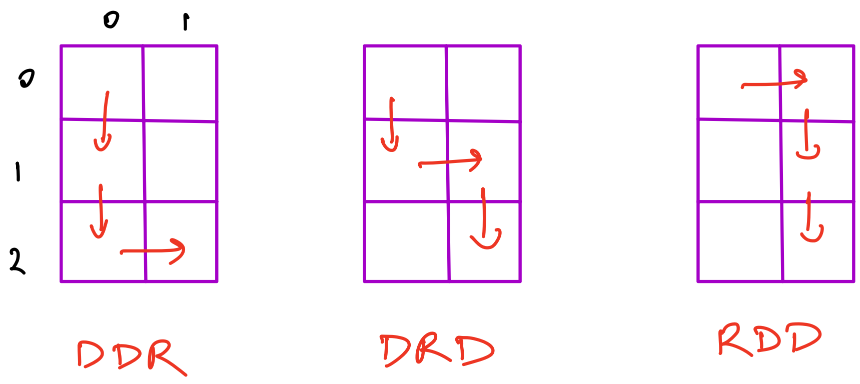
Question 2

You are given a rectangular board of N×M. Print all possible paths from top-left to bottom-right corner of board.

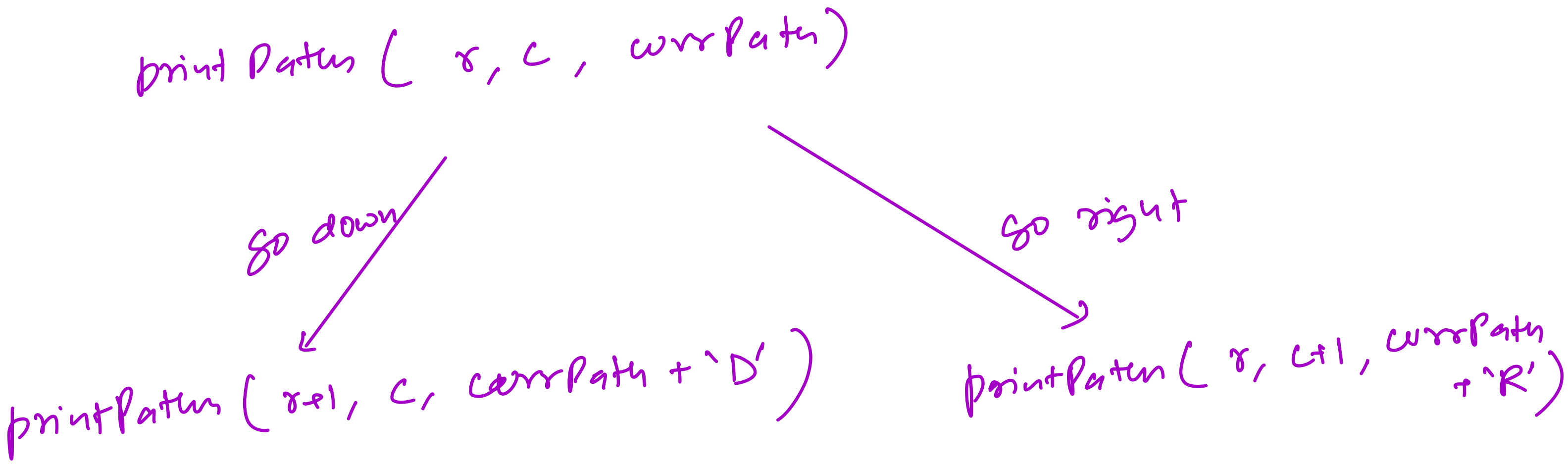
You can only move down (D) or right (R)

Print all paths in lexographical order.

N=3, M=2



DDR DRD RDD



print Paths (r, c, currPath)

go down

printPaths (r+1, c, currPath + 'D')

go right

printPaths (r, c+1, currPath + 'R')

Code

```
list<string> ans
                  ↗0  ↗0
def printPaths ( r,  c,  N, M, currPath ){
        if( r == N-1  &&  c == M-1){
              ans.add( currPath);
              return;
        }

        if( r < N-1 ){
              printPaths( r+1, c, N, M, currPath + "D");
        }

        if( c < M-1 ){
              printPaths ( r, c+1, N, M , currPath + "R");

        }
}
```

no need to remove back since we are not updating currPath

$TC : 2^{\text{path length}}$

path length = # rows + # cols

N + M

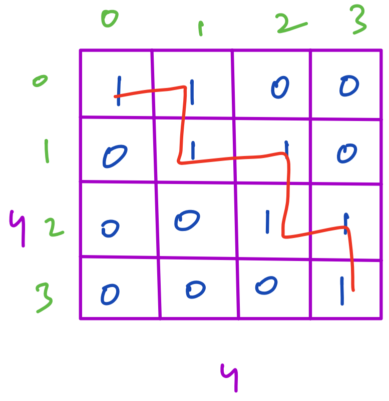
$TC = O(2^{N+M})$

$SC = O(N+M)$

Question 3

Given NxM matrix with 0 or 1 values. to find the shortest path from a given source to a given destination.

A cell with value 0 is a hurdle. The path can only be created with cells of value 1.



	0	1	2	3
0	1	1	0	0
1	0	1	1	0
4 2	0	0	1	1
3	0	0	0	1

4

start = (0,0)

end = (3,3)

path length = 6

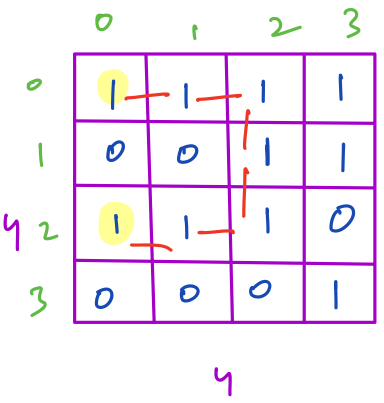
	0	1	2	3
0	1	1	0	0
1	0	1	0	0
4 2	0	0	1	1
3	0	0	0	1

4

start = (0,0)

end = (3,3)

path length = -1



	0	1	2	3
0	1	1	1	1
1	0	0	1	1
4 2	1	1	1	0
3	0	0	0	1

start = (0,0)
end = (2,0)

path length = 6

Solution :

1. start from given source cell and explore all four possible paths.

2. Check if destination is reached or not.

3. Backtrack if not reached

4. Keep track of cells visited.

$$\begin{tikzcd}
 & (i-1,j) & \\
(i,j-1) & (i,j) \arrow[l] \arrow[r] \arrow[u] \arrow[d] & (i,j+1) \\
 & (i+1,j) &
\end{tikzcd}$$

if you are at cell[i][j] , then you can go to

cell[i+1][j] iff

cell[i+1][j] ==1 && i+1 < N

i < N-1

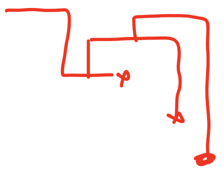
code

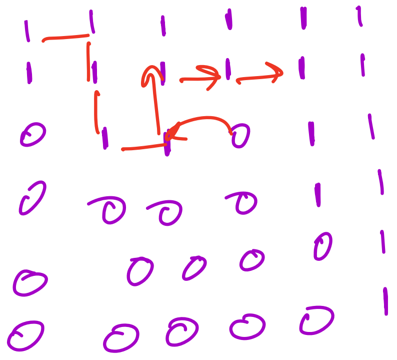
```
             0   1  2  3
row[] = { -1, 1, 0, 0 }
col[] = { 0, 0, 1, -1 }

int ans = INT_MAX
                                       source    destination
def explore Path ( A , visited , i , j , p, q , pathlength ) {

    if ( i ==p && j ==q) {
        ans = min( ans, path length)
        return;

    }

    for ( K =0 to 3) {
        ni = i + row[K]
        nj = j + col[K]

        if( ni >=0 && ni <N && nj >=0 && nj < M
            && A[ni][nj] ==1 && NOT visited[ni][nj] ){

            visited [ni][nj] = true

            explore Path( A, visited, ni, nj, p, q, pathlength
                                                      +1 )
```

```
                visited[ni][nj] = false
            }
        }
}

if ( ans == INT_MAX)
        ans = -1
```

TC = $O(4^{N \times m})$

SC = $O(N \times m)$





```
1 1 1 1 1 1
1 1 1 1 1 1
0 1 1 0 1 1
0 0 0 0 1 1
0 0 0 0 0 1
0 0 0 0 0 1
```