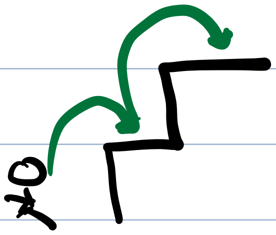
Agenda

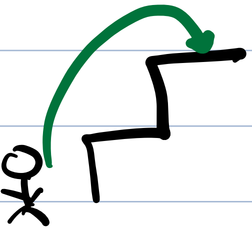
1. Print paths in staircase problem
2. Print all paths from source to destination
3. Shortest Path in Matrix

1. You're climbing a staircase and it takes A steps to reach the top. Each time you can climb 1 or 2 steps. Print all distinct ways to climb the top in lexicographical order.

A = 2



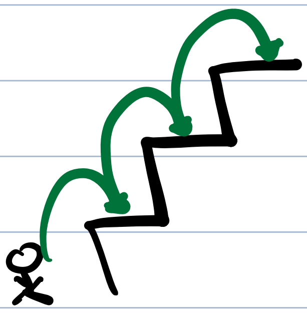
[1,1]



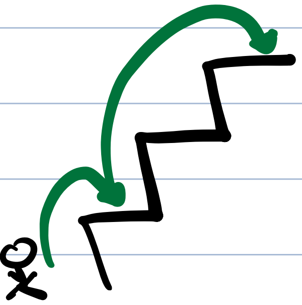
[2]

ans : [[1,1] [2]]

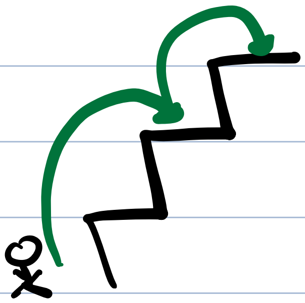
A=3



[1,1,1]

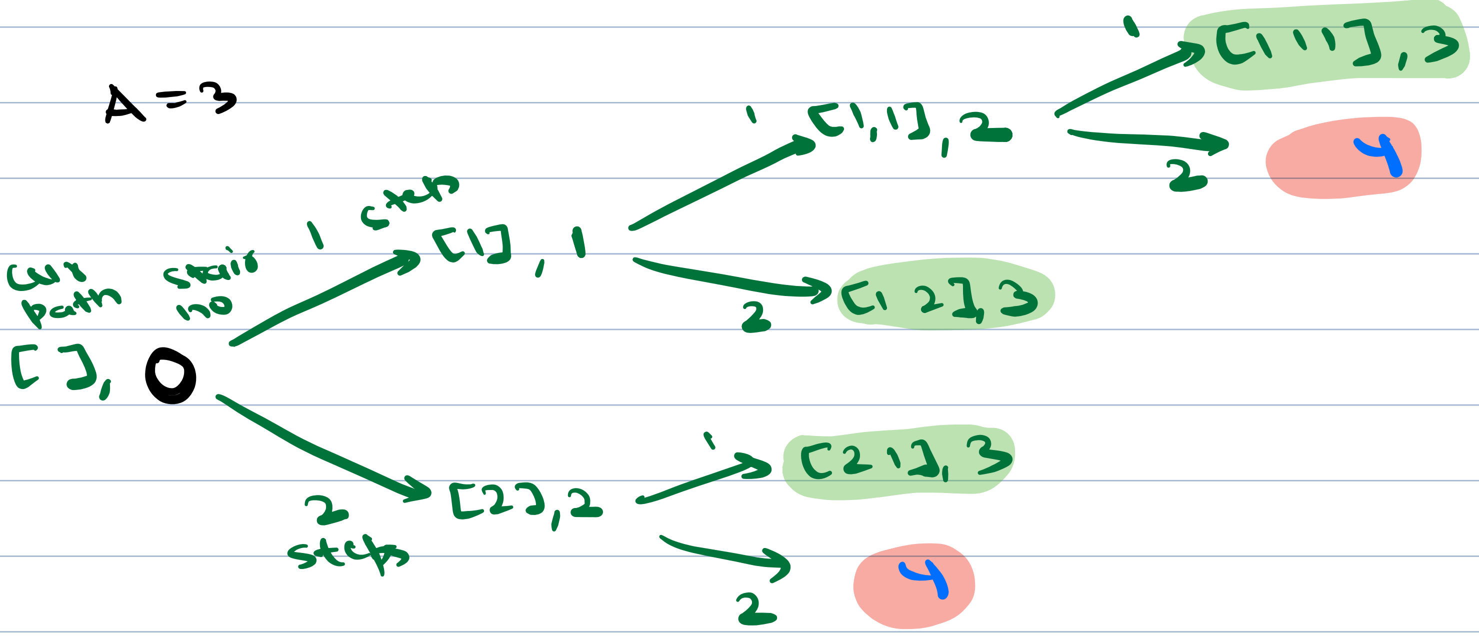


[1,2]



[2,1]

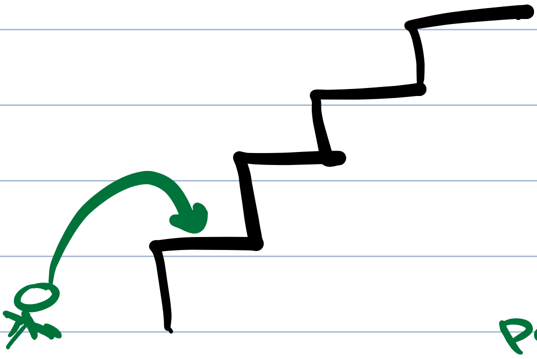
ans : [1,1,1]
[1,2]
[2,1]



// Given A stairs, print all paths to cover A stairs

```
void   generatePaths(A, currentPath) {

        // 1 stair
        generatePaths(A-1, currentPath+[1])

        // 2 stairs
        generatePaths(A-2, currentPath+
                                      [2])
```

A=4



[]
generate(4, cur)

① 1 step

Path → [1] + paths(3)

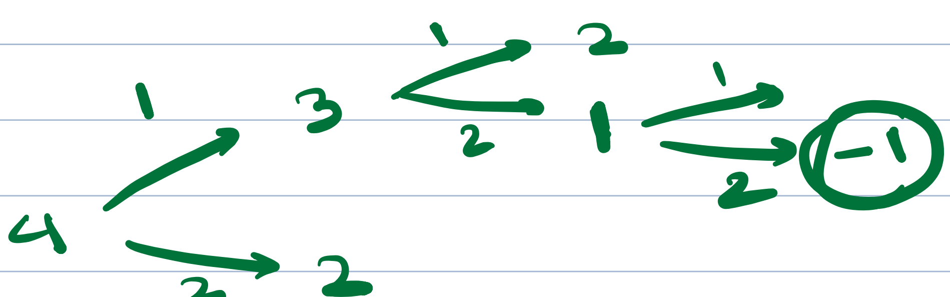
[1] [1, 1, 1]
[1] [1,2]
[1] [2,1]

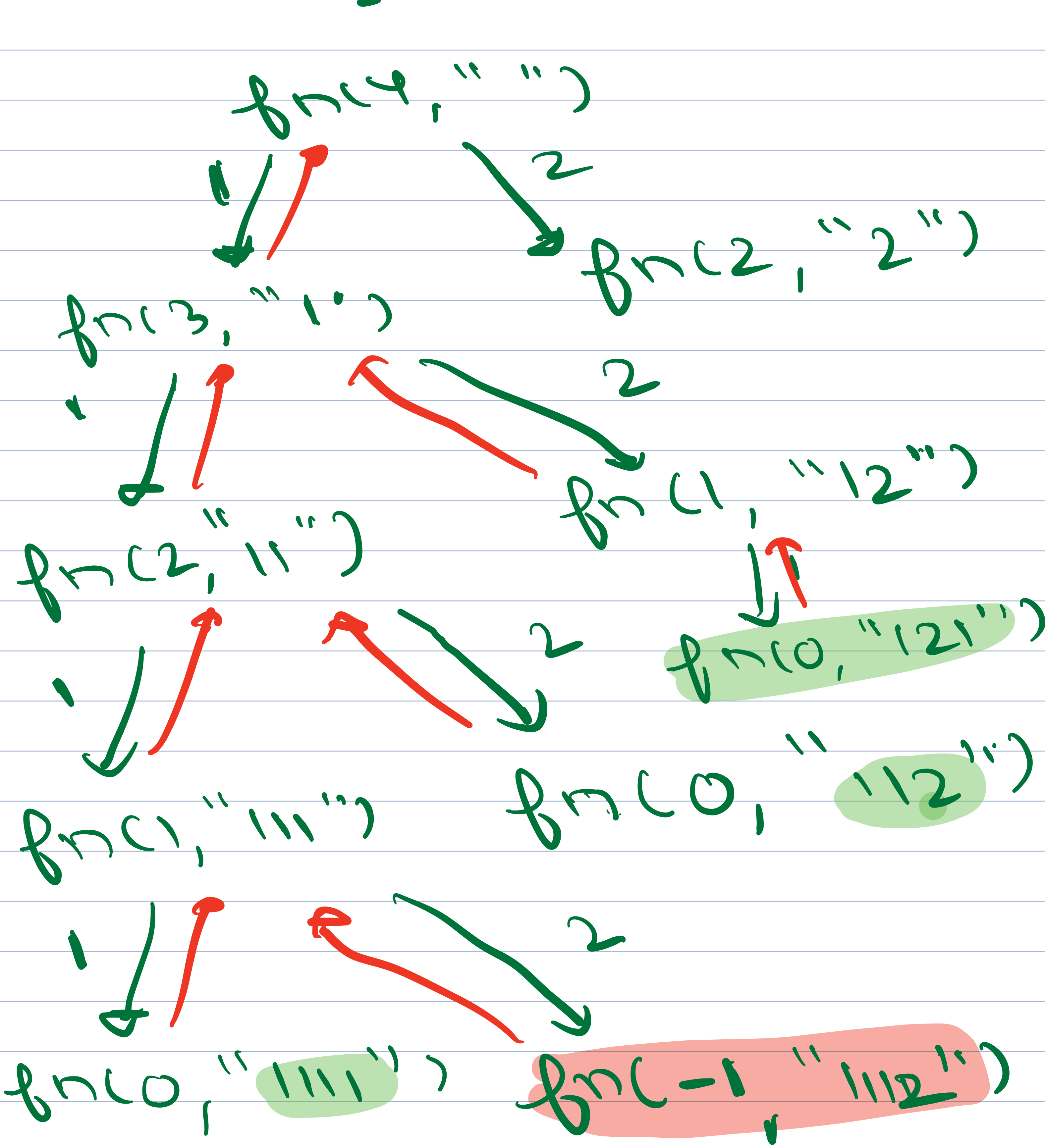
② 2 steps

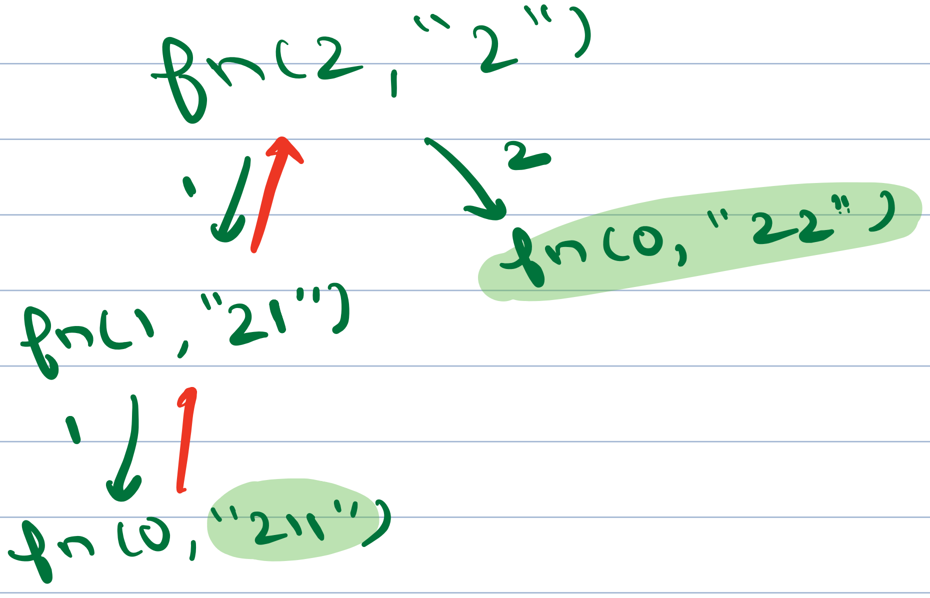
Path → [2] + paths(2)
[2] [1, 1]
[2] [2]

A ↓ , " "

```
void  generatePaths(A, string curPath){
    if (A < 0)   return
    if (A == 0){
        print (curPath)     return
    }

    // 1 step
    generatePaths(A-1, curPath + "1")
    // 2 steps
    generatePaths(A-2, curPath + "2")
}
```




fn(4, "")
1
2
fn(3, "1")
fn(2, "2")
1
2
fn(2, "11")
fn(1, "12")
1
2
fn(0, "121")
fn(1, "111")
fn(0, "112")
1
2
fn(0, "1111")
fn(-1, "1112")



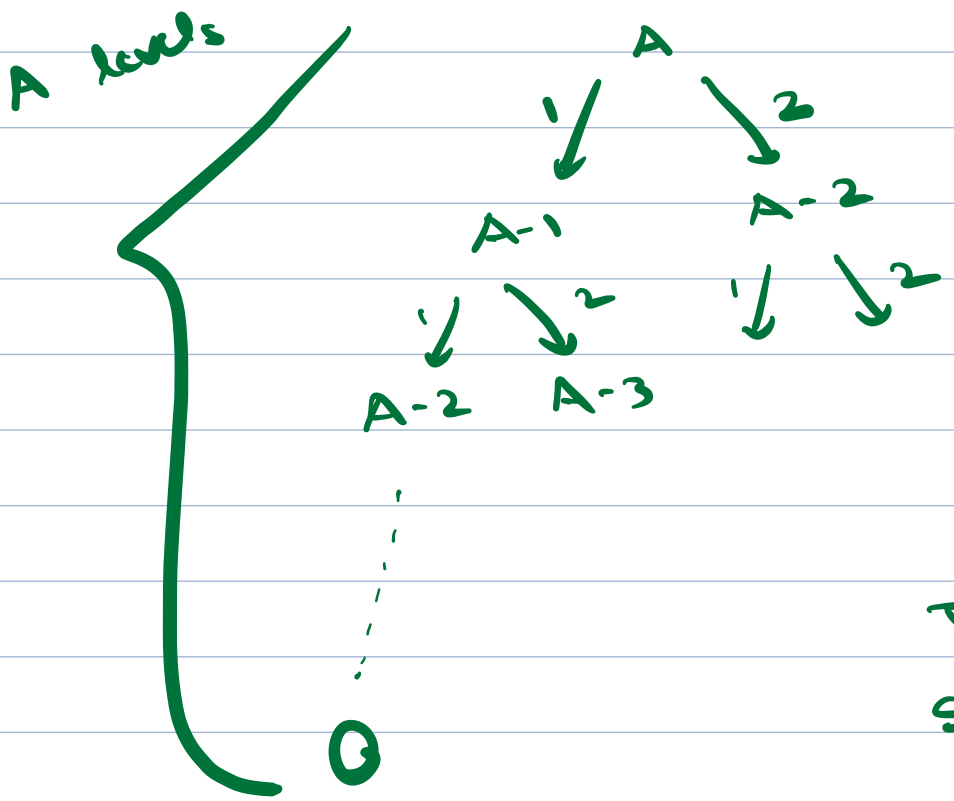
1111

112

121

211

22



TC : $O(2^A)$

SC : $O(\underset{\substack{\downarrow \\ \text{stack space}}}{A} \times A) = A^2$

stack space × space taken by a fn string (max space = A)

A,   []

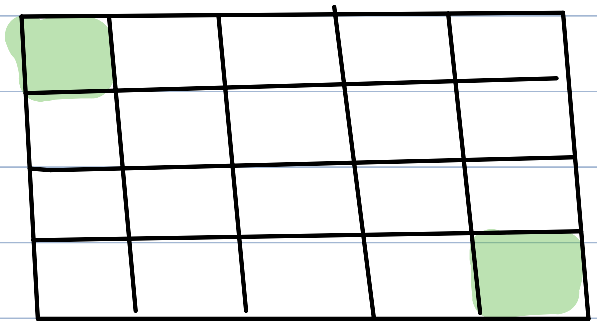
```
void    generatePaths(A, list<int> curPath){
    if (A < 0)    return
    if (A == 0){
        print (curPath)    return
    }
    // 1 step
    curPath.add(1)
    generatePaths(A-1, curPath)
    curPath.pop_back()

    // 2 steps
    curPath.add(2)
    generatePaths(A-2, curPath)
    curPath.pop_back()
}
```

$TC: O(2^A)$

$SC: O(\underset{\text{stack space}}{\underset{\downarrow}{A}} + \underset{\text{cur path}}{\underset{\downarrow}{A}})$

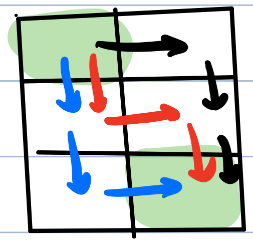
$= O(A)$

2. Given a rectangular board of N×M. Print all possible paths from top left to bottom right corner of the board. You can only move down (D) or right (R) at any point in time. Print all paths in lexicographical order.



4×5

N×M
3×2



ans :

DDR
DRD
RDD

Each cell $\xrightarrow{1}$ R

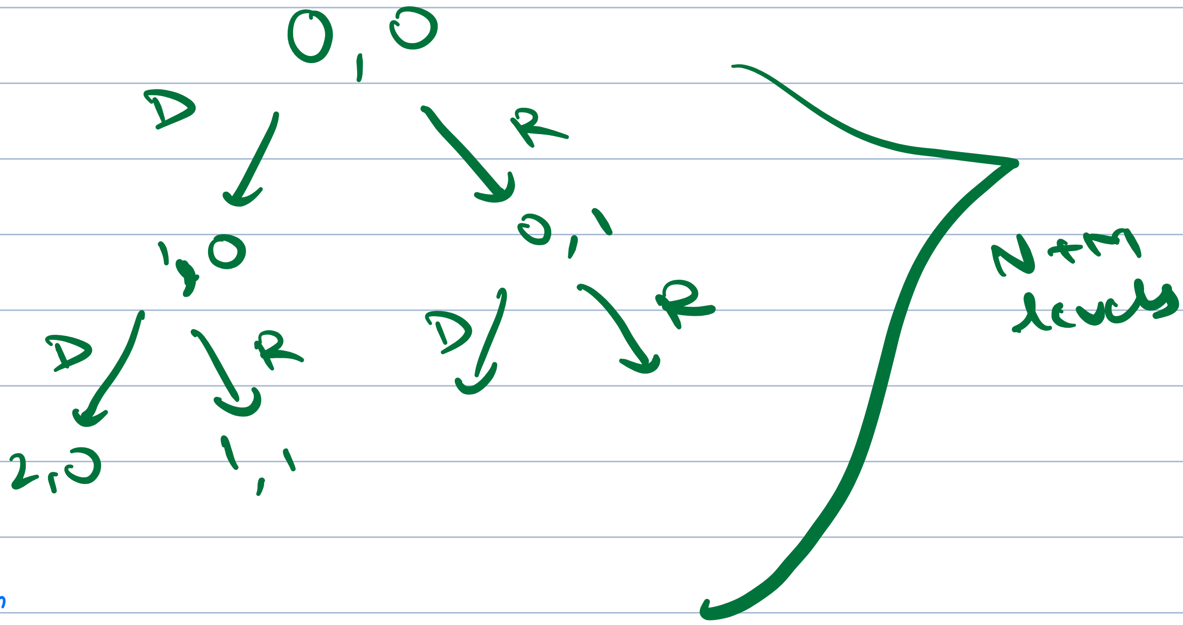
$\xrightarrow[2]{}$ D

For lexicographical order : ① go down

② go right

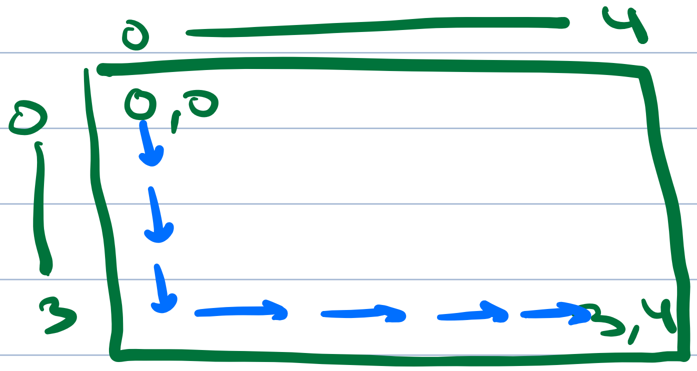
// Generate all paths from src_r, src_c to
   bottom right

```
                                                   [ ]
                  0   ,   0                  list<char>
void  printPaths (src_r, src_c, N, M,           path ){
    if (src_r == N-1 && src_c == M-1 ){
      |> print (path)     return

       // explore D option

   if ( src_r < N-1 ){
        path. add ('D')
      printpaths ( src_r+1, src_c, N, M , path)
        path. pop_back() / removeLast()
   }

       //explore R option
   if (src_c < M-1) {
        path. add ('R')
      printpaths (src_r, src_c+1 , N, M , path)
        path. pop_back() / removeLast()
   }
}
```



TC : O(2^{N+M})

SC : O(N+M)



4x5

N×M

0th → N-1th row : N-1 steps

0th → M-1 col : M-1 steps

Total = N+M-2 steps

0th → 3rd row → 3 steps

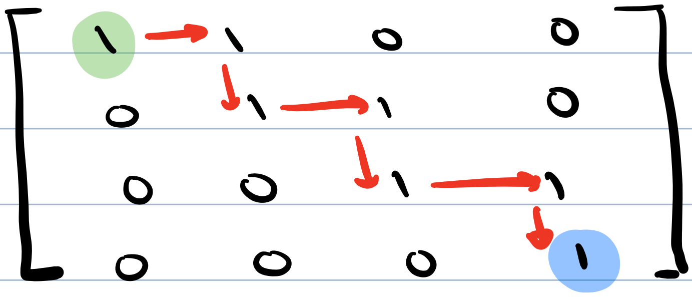
0th → 4th col → 4 steps

10:45

3. Given a N*M matrix where each element is either 0 or 1. Find length of shortest path from given source to destination. 0 means hurdle. Path can only be created from cells with value 1. If no path exists, print -1.

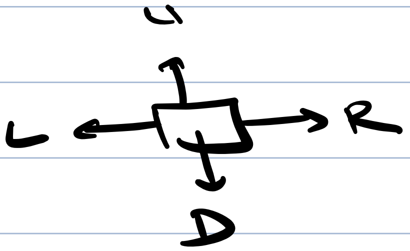
hurdle
↓
wall/obstacle

A =

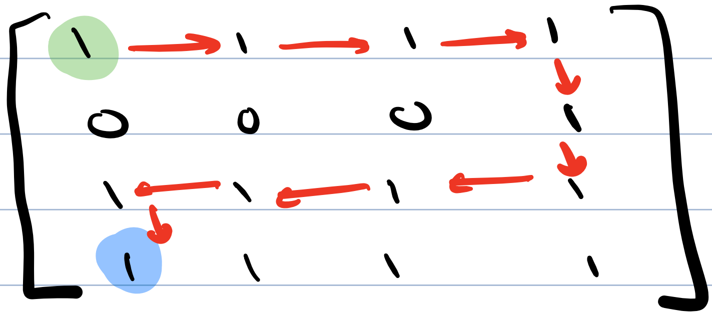


$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ans = 6



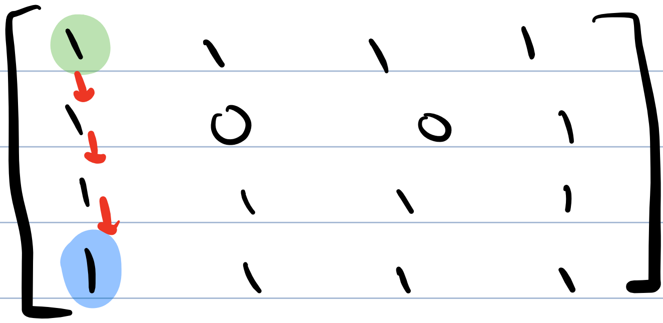
A =



$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

ans = 9

A =



$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

ans = 3

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

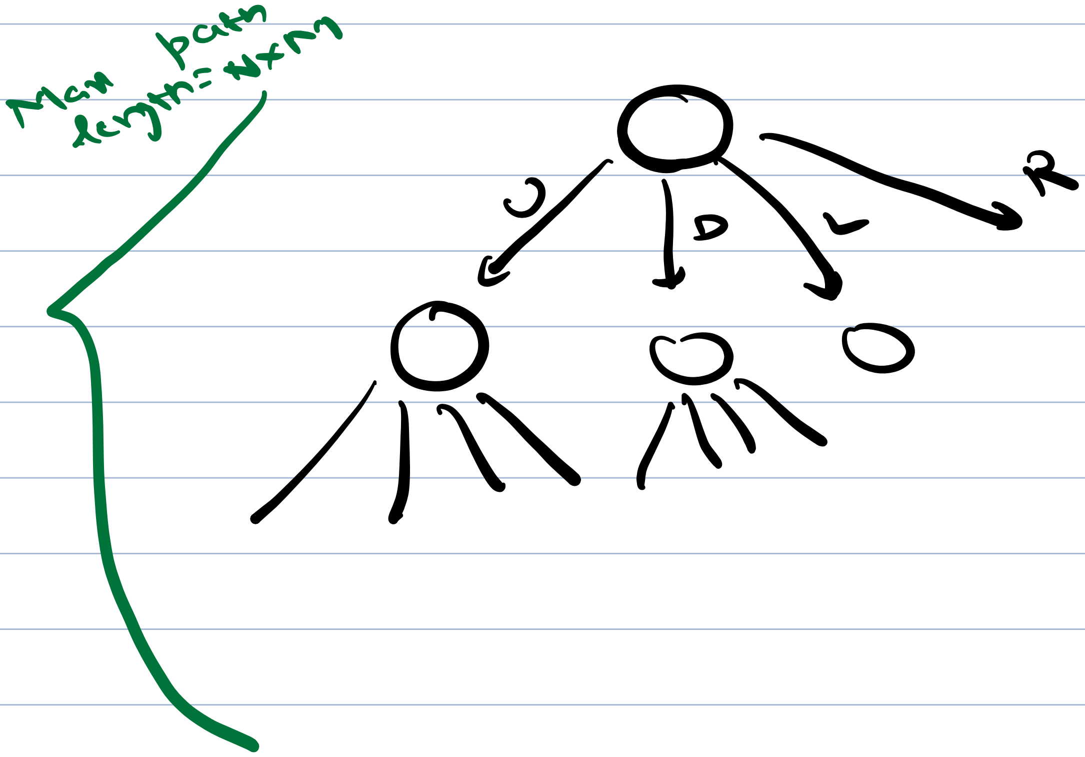
No path possible

ans = -1

Each cell has 4 options → U
→ D
→ L
→ R

```
int  ans = INT_MAX
void  shortestPath (int  src_r, int src_c,
   int  dst_r, dest_c, int N, int M, int steps,
      bool  vis [N][M], int mat[N][M]) {
 if (src_r <0  ||  src_r ≥ N  ||  src_c<0  ||
       src_c ≥ M)       return
  if (mat [src_r][src_c] = =0)
                                   return
   if (vis [src_r][src_c] = = true)
                                        return
  if (src_r = = dest_r  &&  src_c = =dst_c){
        ans  = min (ans, steps)    return
  }

   vis [src_r] [src_c] =true
   // up
   shortest Path(src_r-1, src_c, dst_x, dest_c,
                N, M, steps+1, vis)
   // Down
   shortest Path(src_r+1, src_c, dst_x, dest_c,
                N, M, steps+1, vis)
   // Left
  shortest Path(src_r, src_c-1, dst_x, dest_c,
               N, M, steps+1, vis)
  // Right
 shortest Path(src_r, src+1  , dst_x, dest_c,
              N, M, steps+1, vis)
   vis [src_r] [src_c] = false
}
```

```
int main(){
    bool vis[N][M] = {false}
    shortestPath(_, _, _, _, N, M, 0, vis)
}
```



TC: $O(4^{NM})$

SC: $O(NM)$