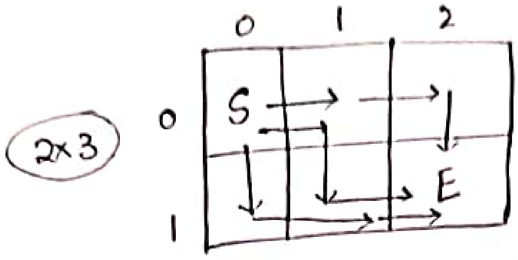
## (17) Grid Unique Paths (Google)

- given a n×m matrix, starting point a[0][0] & ending point a[n-1][m-1]
- can only move down or right at any point of time.
- How many ways / how many unique paths possible?



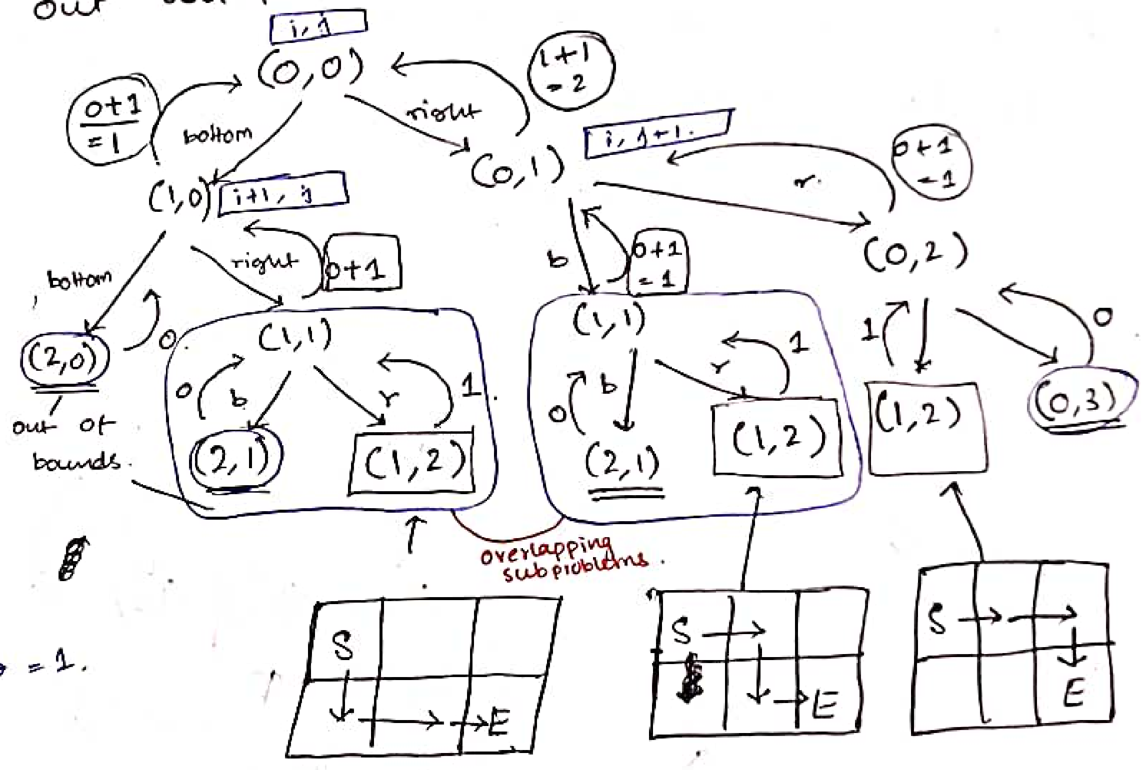

Unique possible paths = 3.

### Brute Force.

(using Recursion)

(•) try out all possible combinations.

(•)




Base cases.

$i >= n$, $j >= m$ → = 0

Destination.

$i = n-1$, $j = m-1$ → = 1.

(•) Algorithm

```
int countPaths(int i, int j, int n, int m) {
    if (i == (n-1) || j == (m-1)) return 1;
    if (i >= n || j >= m) return 0;
    return (countPaths(i+1, j, m, n) + countPaths(i, j+1, m, n));
}
```

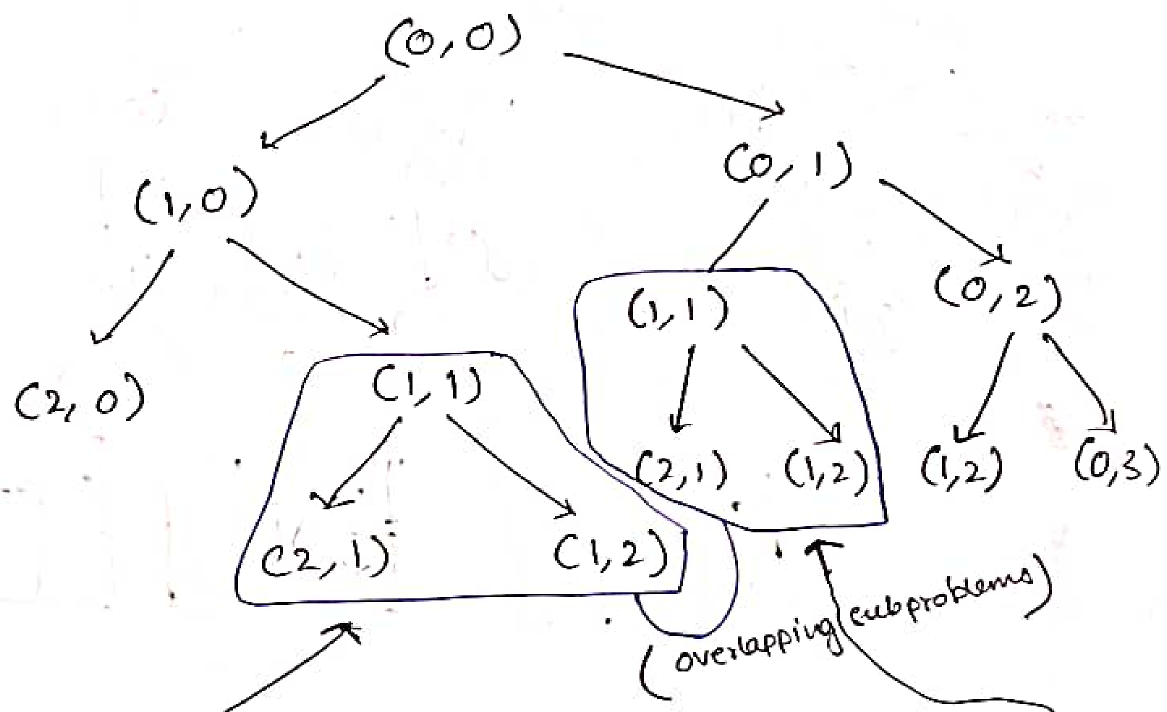
(•) Call countPaths (0,0,n,m) from main().

| T.C. $\rightarrow O(N^N)$ ← exponential |
|---|
| S.C. $\rightarrow O(N^N)$ ← exponential. |

## Optimised Solution.

If we can convert this recursive solution to a Dynamic Programming solution, the complexity will drastically reduce from exponential to quadratic.

(*) Lets understand how to convert recursive code to DP solution. (whenever returning answers, store it in dp table. when calling a new recursion, check if value already exists!)




(•) So if we already found this, we can store it somewhere so that we don't need to find it again here

~~Here~~,

(•) Thus we can use a hash table or DP table.

(•) Now the index i, j can be maximum (n-1) & (m-1).

So total no. of combination of states will be -

0×0, 0×1, 1×0 . . . . . , (n-1) × (m-1)

So, dp[n][m] ., total n×m combinations!

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 3 ~~-1~~ | 2 ~~-1~~ | 1 ~~-1~~ |
| 1 | 1 ~~-1~~ | 1 ~~-1~~ | -1 |

DP

(•) Initialize everything by -1.

(•) From the recursion tree, first calculated value is at (1, 1). So update it in DP.

(•) then (1,0).

(•) then (1,1), but we already calculated that so no need to do a recursive call again.

(•) then (0,2)

(•) then (0,1)

(•) finally (0,0)

This is Dynamic Programming. It helps us not compute values that has already been computed in the past. So, we can reuse values for overlapping subproblems.

Algorithm

```
int countPaths(int i, int j, int m, int n,
               vector<vector<int>> &dp){
    if (i>=m || j>=n) return 0;
    if (i==m-1 || j==n-1) return 1;
    if (dp[i][j] != -1) return dp[i][j];
    else return dp[i][j] = countPaths(i+1, j, m,
                 n, dp) + countPaths(i,
                 j+1, m, n, dp);
}

main() {
    vector<vector<int>> dp (m, vector<int>
                               (n, -1));
    int count = countPaths(0,0,m,n,dp) }
```

initializing 2D vector with -1.

Thus, here 2 recursive calls, (2,1) & (1,2) were avoided.

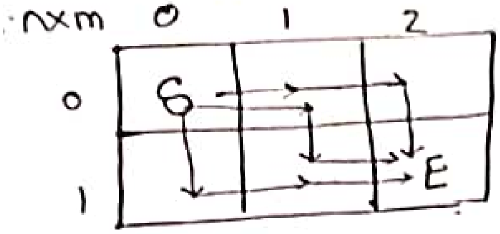
As n & m increases, the no. of overlapping subproblems increases.

T.C. → O(N×M)

S.C. → O(N×M) + O(N×M) ≈ O(N×M)

# Most Optimal Solution

(Using Combination)




RRD
RDR
DRR.

Observations:

1. 3 steps in each case.
2. 2 right (R)
   1 down (D)

So go from S to E, have to go $(n-1)$ steps ~~right~~ down & $(m-1)$ steps ~~down~~ right.

$\therefore$ Total no. of steps $= (m-1) + (n-1)$

$= \underline{n+m-2}$

Here, steps $= 3$ $[2+3-2]$.

So, we have

| | | |
|---|---|---|
| D | R | R |
| R | D | R |
| R | R | D |

$\underbrace{{}^{3}C_{2}}_{\text{right}} = 3$ or. $\underbrace{{}^{3}C_{1}}_{\text{down}} = 3$.

So, if total no. of ways is $(m+n-2)$ & we can go right only $(m-1)$ times & down $(n-1)$ time,

Total no. of unique paths

$= \boxed{{}^{m+n-2}C_{m-1}}$ or $\boxed{{}^{m+n-2}C_{n-1}}$.

no. of ways I can go right in total m+n-2 steps

no. of ways I can go down in m+n-2 steps

Now, ${}^{n}C_r$ can be found very easily.

(from Pascal's $\Delta$)

I wrote

$$\left[\frac{(n-(r-1)) \ldots (n-2) \times (n-1) \times n}{r \times (r-1) \cdots \times 1}\right]$$

$${}^{10}C_3 = \frac{8 \times 9 \times 10}{3 \times 2 \times 1}$$

$$\boxed{\begin{array}{l} T.C. \rightarrow O(m-1) \text{ or } O(n-1). \\ S.C. \rightarrow O(1). \end{array}}$$