

# Unique Paths

There is a robot on an  $m \times n$  grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers  $m$  and  $n$ , return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to  $2 * 10^9$ .

## Example 1:



**Input:**  $m = 3, n = 7$

**Output:** 28

## Approach 1: Brute force:

Try all the possible ways recursively

Time :  $O(2^n)$

Space:  $O(m+n)$

## Approach 2: Dynamic programming

```
class Solution {  
    public int uniquePaths(int m, int n) {  
        int[][] map=new int[m+1][n+1];  
        for(int i=0;i<=m;i++)  
        {  
            Arrays.fill(map[i],-1);  
        }  
    }  
}
```

```

int s=solve(m,n,1,1,map);

return s;

}

public int solve(int m,int n,int i,int j,int[][] map)
{
    if(i==m&& j==n)
    {
        return 1;
    }
    if(map[i][j]!=-1)
    {
        return map[i][j];
    }
    int count=0;
    if(i<m)
    {
        count+=solve(m,n,i+1,j,map);
    }
    if(j<n)
    {
        count+=solve(m,n,i,j+1,map);
    }
    return map[i][j]=count;
}
}

```

Time :  $O(m*n)$

Space:  $O(m*n)$

### Approach 3:Combinatorics

In this approach We have to calculate  $m+n-2 \text{ C } n-1$  here which will be  $(m+n-2)! / (n-1)! (m-1)!$

Now, let us see how this formula is giving the correct answer ([Reference 1](#)) ([Reference 2](#)) **read reference 1 and reference 2 for a better understanding**

$m$  = number of rows

$n$  = number of columns

Total number of moves in which we have to move down to reach the last row =  $m - 1$  ( $m$  rows, since we are starting from (1, 1) that is not included)

Total number of moves in which we have to move right to reach the last column =  $n - 1$  ( $n$  column, since we are starting from (1, 1) that is not included)

**Down moves =  $(m - 1)$**

**Right moves =  $(n - 1)$**

**Total moves = Down moves + Right Moves =  $(m - 1) + (n - 1)$**

**Now think moves as a string of 'R' and 'D' character where 'R' at any  $i$ th index will tell us to move 'Right' and 'D' will tell us to move 'Down'**

**Now think of how many unique strings (moves) we can make where in total there should be  $(n - 1 + m - 1)$  characters and there should be  $(m - 1)$  'D' character and  $(n - 1)$  'R' character?**

Choosing positions of  $(n - 1)$  'R' characters, results in automatic choosing of  $(m - 1)$  'D' character positions

Calculate  ${}^nC_r$

Number of ways to choose positions for  $(n - 1)$  'R' character =  $\text{Total positions } {}^{n-1}C_{n-1} = \text{Total positions } {}^{m-1}C_{m-1} = (n - 1 + m - 1) !=$

**Another way to think about this problem:** Count the Number of ways to make an **N digit Binary String** (String with 0s and 1s only) with '**X**' **zeros** and '**Y**' **ones** (here we have replaced 'R' with '0' or '1' and 'D' with '1' or '0' respectively whichever suits you better)

```
static int numberOfPaths(int m, int n)
{
    // We have to calculate m+n-2 C n-1 here
    // which will be (m+n-2)! / (n-1)! (m-1)!
    int path = 1;
    for (int i = n; i < (m + n - 1); i++) {
        path *= i;
        path /= (i - n + 1);
    }
    return path;
}
```

Time: $O(m-1)$  or  $O(n-1)$

Space:  $O(1)$ ;

