**Problem link**: [Unique paths I](https://leetcode.com/problems/unique-paths/)

\*\***1. recursive- solution: TC =** O(2^ (m\*n), for every index we have 2option(left/up)

**//SC** = O(m-1 + n-1) recursion stack space

**#include <bits/stdc++.h>**

**int f(int i, int j){**

**if(i==0 && j==0) return 1; //reached to target, got 1 path possible**

**if(i<0 || j<0) return 0; //out of boundary, cant be contribute in path**

**//go in all possible direction**

**int left = f(i, j-1);**

**int up = f(i-1, j);**

**//all possible ways = sum up and return**

**return left+up;**

**}**

**int uniquePaths(int m, int n) {**

**return f(m-1, n-1);**

**}**

\*\***2. DP- memoization solution: //tc =** O(m\*n) for all new calls (0, 0 to m-1, n-1)

// **sc** = O(m-1 + n-1) for recursion stack space + O(m\*n) for dp[]

**#include <bits/stdc++.h>**

**int f(int i, int j, vector<vector<int>> &dp){**

**if(i==0 && j==0) return 1;**

**if(i<0 || j<0) return 0;**

**if(dp[i][j] != -1) return dp[i][j]; //return if we have ans**

**int left = f(i, j-1, dp);**

**int up = f(i-1, j, dp);**

**dp[i][j] = left+up; //calc and store in dp**

**return dp[i][j];**

**}**

**int uniquePaths(int m, int n) {**

**vector<vector<int>> dp(m, vector<int>(n, -1)); //step1 - declare dp[m][n]**

**return f(m-1, n-1, dp);**

**}**

\*\***3. DP- tabulation: //TC = O(m\*n), SC = O(m\*n)**

**//DP- tabulation: bottom- up approach**

**#include <bits/stdc++.h>**

**int uniquePaths(int m, int n) {**

**vector<vector<int>> dp(m, vector<int>(n, -1));**

**dp[0][0] = 1;**

**for(int i=0; i<m; i++){**

**for(int j=0; j<n; j++){**

**if(i==0 && j==0) dp[i][j] = 1;**

**else{ int left = 0; int up = 0;**

**if(j>0)**

**left += dp[i][j-1];**

**if(i>0)**

**up += dp[i-1][j];**

**dp[i][j] = left+up;**

**}**

**}**

**}**

**return dp[m-1][n-1];**

**}**

\***\*4. DP- optimized space:** //TC = O(m\*n), SC = O(2n)

**//use a prev[n] to keep track of [i-1][j] value and use temp[n] for [i][j-1] value**

**#include <bits/stdc++.h>**

**int uniquePaths(int m, int n) {**

**vector<int> prev(n, 0); //stores previous row**

**for(int i=0; i<m; i++){**

**vector<int> temp(n, 0); //stores current values**

**for(int j=0; j<n; j++){**

**if(i==0 && j==0) temp[j] = 1;**

**else{**

**temp[j] = ((j>0) ? temp[j-1]: 0) + prev[j]; //left+up**

**}**

**}**

**prev = temp;**

**}**

**return prev[n-1];**

**}**