Previous Next

62. Unique Paths

Feb. 9, 2020 | 22.7K views

Average Rating: 4.83 (23 votes)

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Above is a 7 x 3 grid. How many possible unique paths are there?

Example 1:

```
Input: m = 3, n = 2
Output: 3
Explanation:
From the top-left corner, there are a total of 3 ways to reach the bottom-right corner
1. Right -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right
```

Example 2:

```
Input: m = 7, n = 3
Output: 28
```

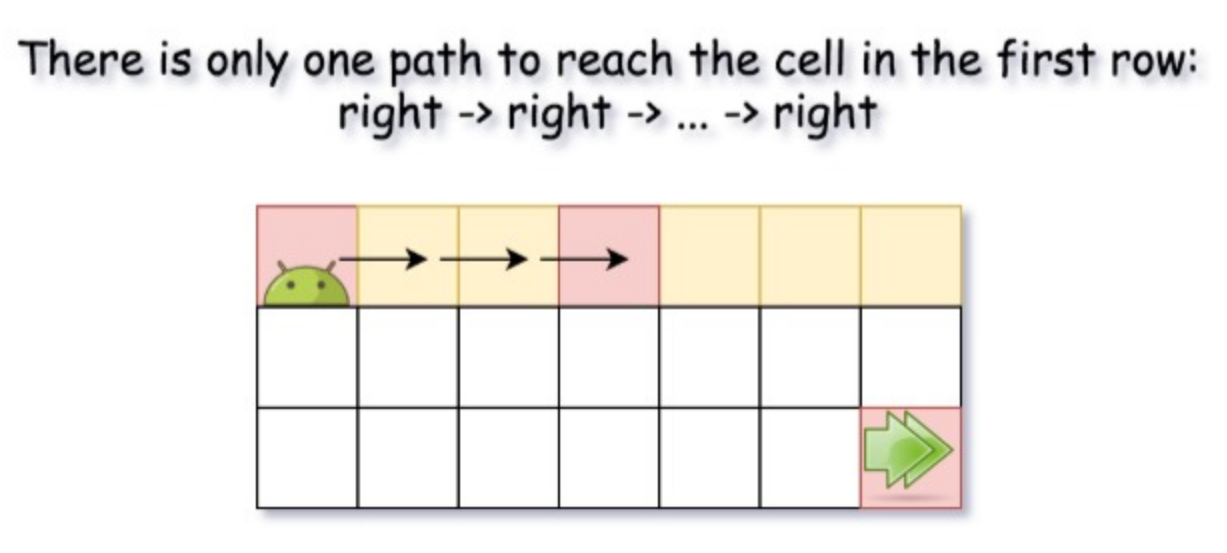
Constraints:

- `1 <= m, n <= 100`
- It's guaranteed that the answer will be less than or equal to `2 * 10 ^ 9`.

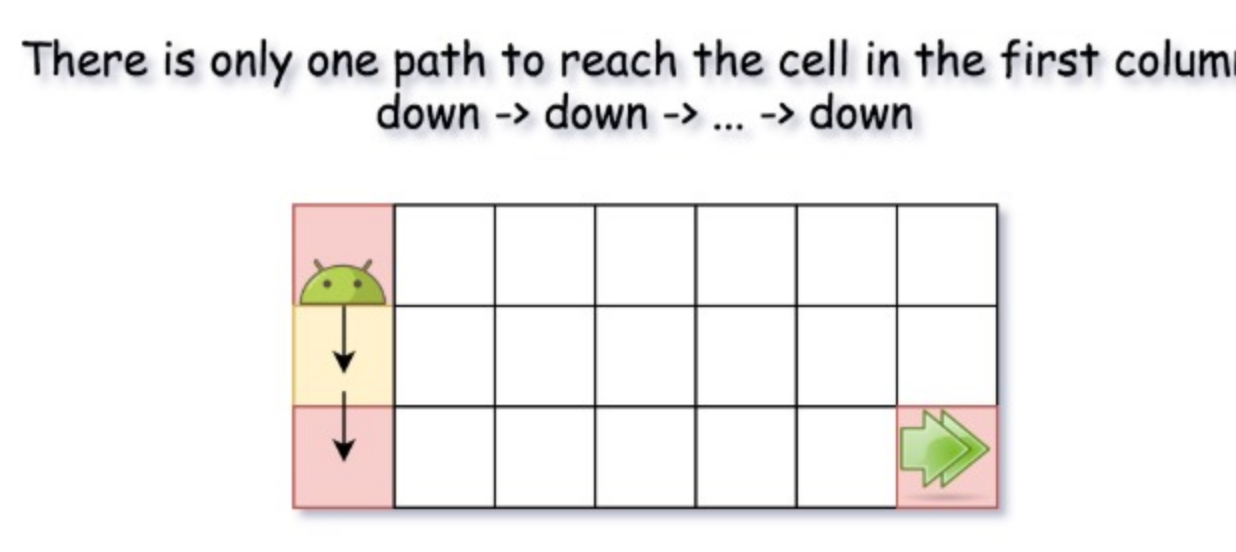
Solution

Overview

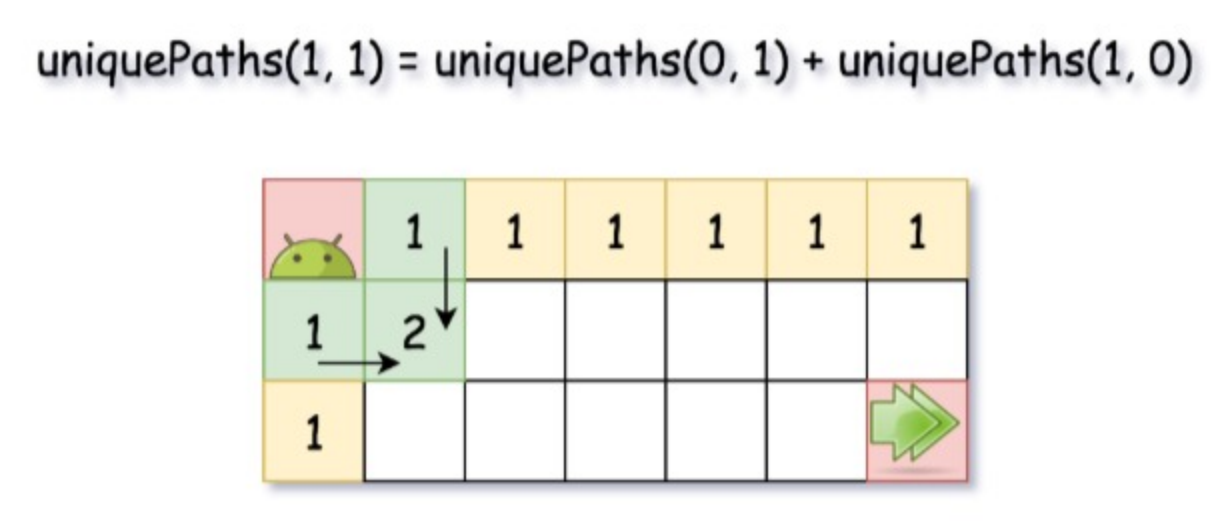
Since robot can move either down or right, there is only one path to reach the cells in the first row: right->right->...->right.



The same is valid for the first column, though the path here is down->down-> ...->down.



What about the "inner" cells `(m, n)`? To such cell one could move either from the upper cell `(m, n - 1)`, or from the cell on the right `(m - 1, n)`. That means that the total number of paths to move into `(m, n)` cell is `uniquePaths(m - 1, n) + uniquePaths(m, n - 1)`.



Now, one could transform these ideas into 3-liner recursive solution:

Java Python Copy

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        if m == 1 or n == 1:
            return 1

        return self.uniquePaths(m - 1, n) + self.uniquePaths(m, n - 1)
```

This solution is not fast enough to pass all the testcases, though it could be used as a starting point for the DP solution.

Approach 1: Dynamic Programming

One could rewrite recursive approach into dynamic programming one.

Algorithm

- Initiate 2D array `d[m][n] = number of paths`. To start, put number of paths equal to 1 for the first row and the first column. For the simplicity, one could initiate the whole 2D array by ones.
- Iterate over all "inner" cells: `d[col][row] = d[col - 1][row] + d[col][row - 1]`.
- Return `d[m - 1][n - 1]`.

Implementation

1 / 13

Java Python Copy

```python
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        d = [[1] * n for _ in range(m)]

        for col in range(1, m):
            for row in range(1, n):
                d[col][row] = d[col - 1][row] + d[col][row - 1]

        return d[m - 1][n - 1]
```

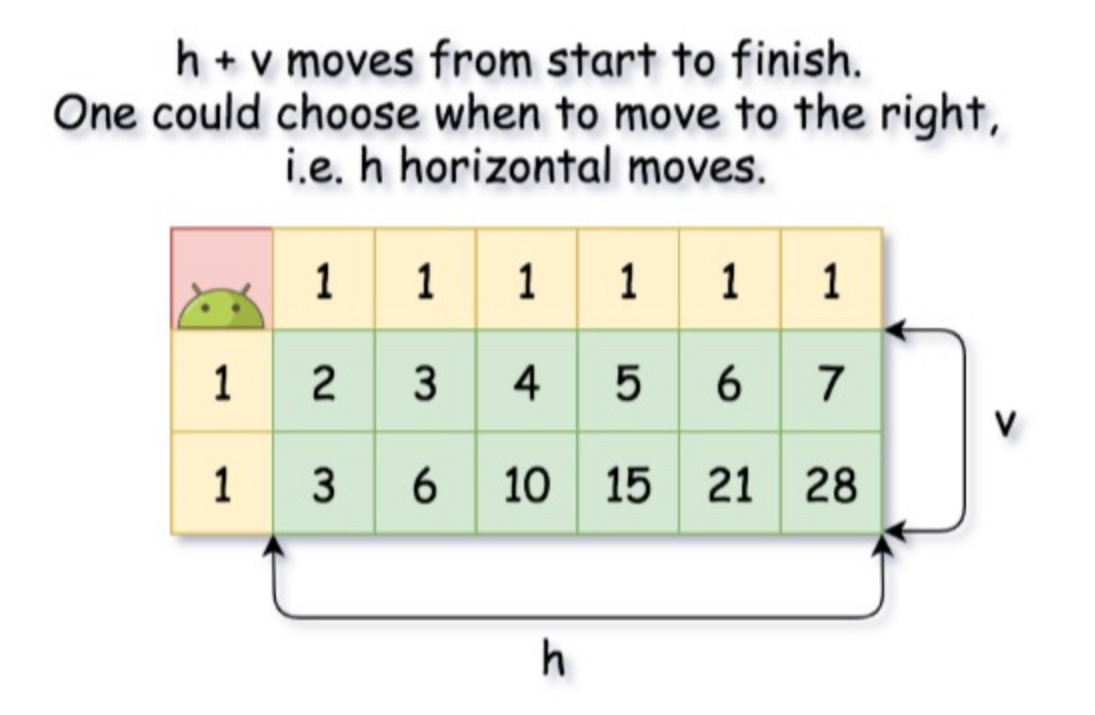
Complexity Analysis

- Time complexity: $\mathcal{O}(N \times M)$.
- Space complexity: $\mathcal{O}(N \times M)$.

Approach 2: Math (Python3 only)

Could one do better than $\mathcal{O}(N \times M)$? The answer is yes.

The problem is a classical combinatorial problem: there are $h + v$ moves to do from start to finish, $h = m - 1$ horizontal moves, and $v = n - 1$ vertical ones. One could choose when to move to the right, i.e. to define h horizontal moves, and that will fix vertical ones. Or, one could choose when to move down, i.e. to define v vertical moves, and that will fix horizontal ones.



In other words, we're asked to compute in how many ways one could choose p elements from $p + k$ elements. In mathematics, that's called binomial coefficients

$$C^h_{h+v} = C^v_{h+v} = \frac{(h+v)!}{h!v!}$$

The number of horizontal moves to do is $h = m - 1$, the number of vertical moves is $v = n - 1$. That results in a simple formula

$$C^h_{h+v} = \frac{(m+n-2)!}{(m-1)!(n-1)!}$$

The job is done. Now time complexity will depend on the algorithm to compute factorial function $(m + n - 2)!$. In short, standard computation for $k!$ using the definition requires $\mathcal{O}(k^2 \log k)$ time, and that will be not as good as DP algorithm.

The best known algorithm to compute factorial function is done by Peter Borwein. The idea is to express the factorial as a product of prime powers, so that $k!$ can be computed in $\mathcal{O}(k (\log k \log \log k)^2)$ time. That's better than $\mathcal{O}(k^2)$ and hence beats DP algorithm.

The authors prefer not to discuss here various factorial function implementations, and hence provide Python3 solution only, with built-in divide and conquer factorial algorithm. If you're interested in factorial algorithms, please check out good review on this page.

Implementation

Python Copy

```python
from math import factorial
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        return factorial(m + n - 2) // factorial(n - 1) // factorial(m - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}((M+N)(\log(M+N) \log\log(M+N))^2)$.
- Space complexity: $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 14 Sort By

Type comment here... (Markdown is supported)

Preview Post

campiador ★ 23 April 13, 2020 10:35 AM

In the second solution (DP) we only need one row to keep, and we can keep reading from and updating it. That would result in O(n) space instead of O(m*n);

23 Share Reply

SHOW 1 REPLY

warland ★ 20 March 2, 2020 5:39 PM

Here is an easy Java Version.

```java
// Runtime: O(M*N) Memory: O(N)
public int uniquePaths(int m, int n) {
    int[] dp = new int[n];
```

Read More

12 Share Reply

ntkw ★ 52 May 24, 2020 6:10 PM

3rd solution time complexity is something evil

5 Share Reply

theseungjin ★ 163 March 28, 2020 4:28 AM

A bit confused by the code for DP implementation. When you create DP table, you use 'n' as the columns. But in the for loop, to iterate over 'col' you use `range(1, m)`. I understand the output is not affected, but the code causes confusion in understanding

3 Share Reply

SHOW 1 REPLY

Xyzzy123 ★ 114 May 29, 2020 12:04 AM

In Python 3.8, there's a new function that calculates n-choose-k directly:

```python
def uniquePaths(self, m: int, n: int) -> int:
    return math.comb(m + n - 2, n - 1)
```

2 Share Reply

rite2riddhi ★ 2 June 8, 2020 8:56 PM

are we really expected to come up with the apporach 3 solution in a real interview?

2 Share Reply

ztztzt8888 ★ 53 June 30, 2020 12:31 PM

The recursion method is quite intuitive, it only needs memorization to pass the tests.

```java
    public int uniquePaths(int m, int n) {
        int[][] memo = new int[m][n];
        return uniquePaths(memo, m - 1, n - 1);
```

Read More

1 Share Reply

vladlazar94 ★ 1 April 15, 2020 11:34 AM

I love the illustrations ;) Very subtle Android commercial :P

1 Share Reply

yaligar ★ 0 3 days ago

Better explanation: https://towardsdatascience.com/understanding-combinatorics-number-of-paths-on-a-grid-bddf08e28384

0 Share Reply

adamkhakhar ★ 3 July 2, 2020 6:56 AM

It is also worth adding that there is on O(Max(n, m)) solution: we can derive the formula for the answer: (n + m - 2) Choose (n - 1). Please reply if this solution is not easy to understand and I can show my calculations.

0 Share Reply

‹ 1 2 ›