unique paths

joyce shi

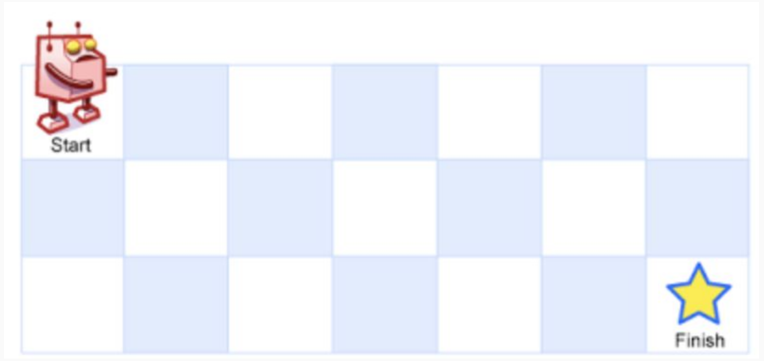
october 25, 2022

unique paths problem 🔎

problem definition

https://leetcode.com/problems/unique-paths/

given a fixed-size grid, how many unique paths are there from the top-left corner to the bottom-right corner? (robot can only move down / right)



<u>input</u>: m = 3, n = 7

<u>output</u>: 28

more formally…

- input: # of columns (m), # of rows (n)
- output: # of unique paths that you can take to reach the bottom-right corner
- constraints:
  - robot can only move down or right
  - 1 <= m, n <= 100

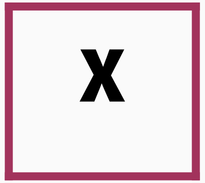
unique paths solution 🛠️

dynamic programming overview

the idea is to solve bigger problems with a series of smaller problems.

1. identify **base cases**
    - solve the smallest “sub-problems”
2. establish a **recurrence** relationship
    - identify the next set of sub-problems
    - how can you use the previous set of smaller sub-problems to compute the next set of bigger sub-problems?
3. solve all the sub-problems to **compute the final solution**

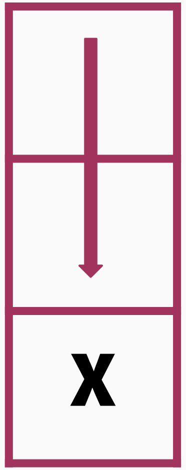
identify base cases



case 1: one square

m = 1, n = 1

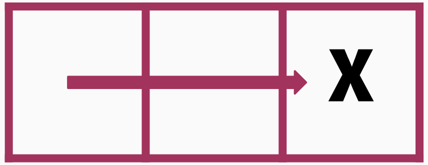
1 unique path



case 2: **one column**

n = 1

1 unique path

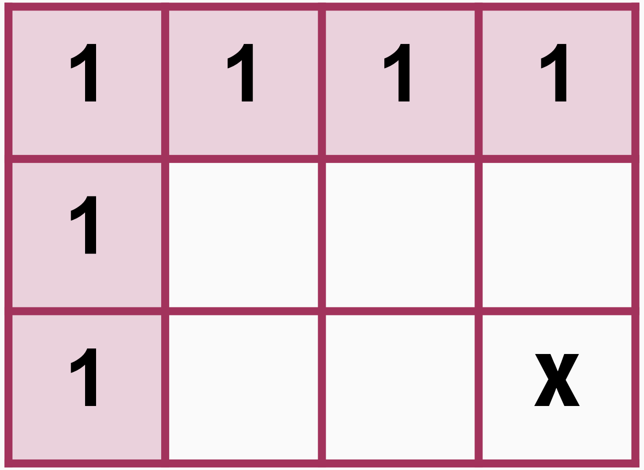


case 3: one row

m = 1

1 unique path

establish a recurrence

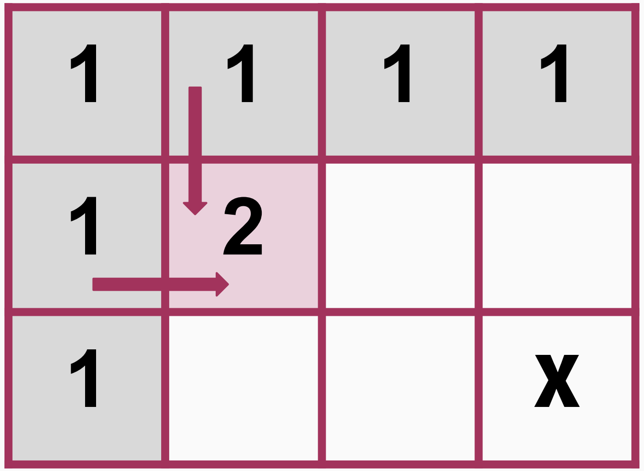


1	1	1	1
1			
1			X

we don’t yet know how many unique paths exist to get to the bottom-right corner, but for a lot of squares, we do know how many unique paths exist to reach those squares.

the number of unique paths to reach every specific square make up our set of “sub-problems”

establish a recurrence

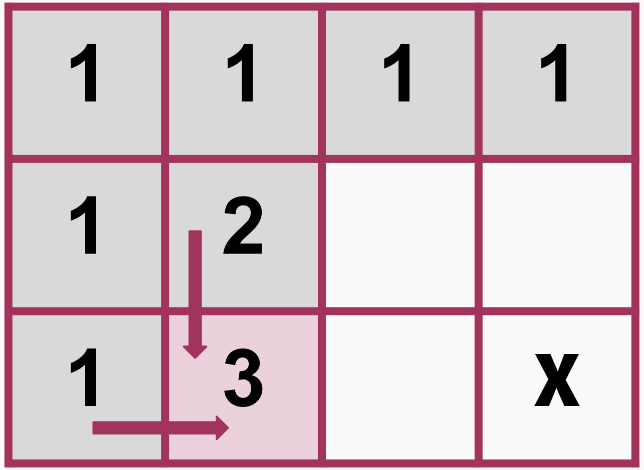


but how many unique paths does it take to get to the next square?

case 1: path from top square
case 2: path from left square

clearly, there are 2 unique paths

establish a recurrence



notice that the same 2 cases hold:

case 1: path from top square
case 2: path from left square

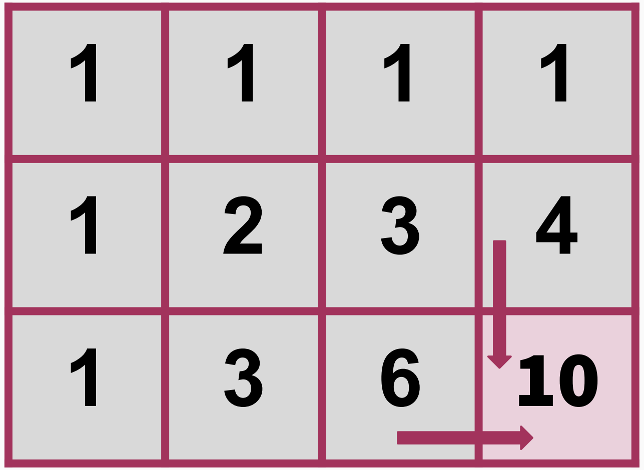
but, there were 2 unique paths to get to the top square, so there are now **3 unique paths**

establish a recurrence

1	1	1	1
1	2	3	4
1	3	6	x

we can follow the exact same pattern to fill out the remaining squares.

compute the final solution



1	1	1	1
1	2	3	4
1	3	6	10

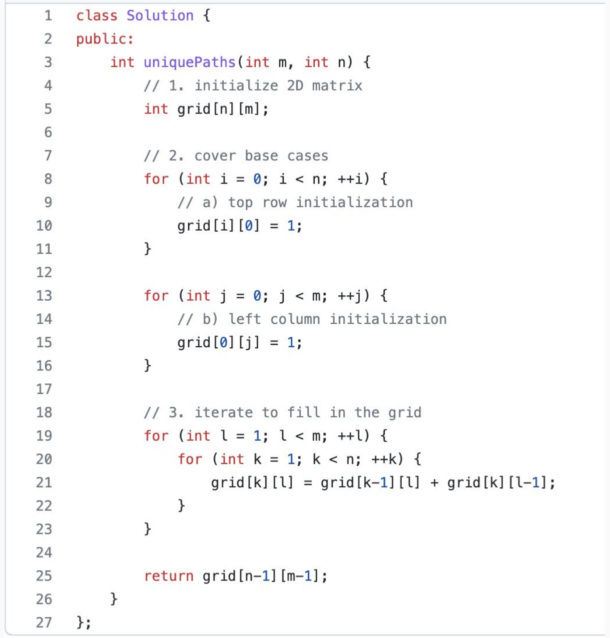
finally, following the same pattern, we can compute the bottom-right square and show that there are **10 unique paths** from the top-left to the bottom-right.

pseudocode

```
1. initialize a 2D matrix: grid[n][m]
    ● grid[i][j] contains the # of unique paths to reach that particular square
2. compute base cases
    a. top-left square
    b. top row
    c. left row
3. iterate to fill the grid: grid[i][j] = grid[i-1][j] + grid[i][j-1]
4. return solution in the bottom-right square: grid[n-1][m-1]
```

full code solution in c++

https://github.com/joyce-shi/unique-paths



```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {
        // 1. initialize 2D matrix
        int grid[n][m];

        // 2. cover base cases
        for (int i = 0; i < n; ++i) {
            // a) top row initialization
            grid[i][0] = 1;
        }

        for (int j = 0; j < m; ++j) {
            // b) left column initialization
            grid[0][j] = 1;
        }

        // 3. iterate to fill in the grid
        for (int l = 1; l < m; ++l) {
            for (int k = 1; k < n; ++k) {
                grid[k][l] = grid[k-1][l] + grid[k][l-1];
            }
        }

        return grid[n-1][m-1];
    }
};
```

questions? 🙋🏻‍♀️

next steps

- **unique paths ii**: https://leetcode.com/problems/unique-paths-ii/
- **unique paths iii**: https://leetcode.com/problems/unique-paths-iii/

feel free to reach out to me with questions (: