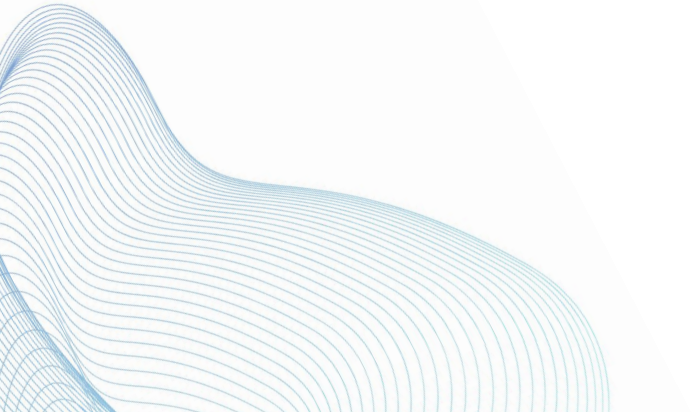
Dynamic Programming

Bottom Up



🌀Lecture Flow

- Prerequisites
- What it is
- How to convert top down to bottom up?
- Common Variants
- Common Pitfalls
- Practice Questions
- Quote of the day

📋 Prerequisites

- DP - Top Down Approach

⏰Recap Questions

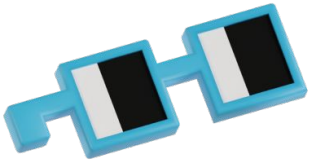
- What is the Time Complexity of DP in terms of States
  - Imagine you have a DP solution with three states
    - i : 0 < i < 1,000
    - j : 0 < j < 10
    - k: 0 < k < 10
- Is DP brute force? Explain?
- What are problems that have Optimal Substructure, Overlapping Subproblems and both together?
- What are the three main things we need to figure out with a top-down approach?

⏰Revision: What is DP❓

A method for solving complex problems by **breaking them down** into simpler subproblems and **reusing solutions** to these subproblems to construct the solution to the original problem.

⏰Revision: Use cases

When the problem exhibits the properties of **overlapping subproblems** and **optimal substructure.**



What is bottom-up (tabulation) dp ❓

⬆️What is bottom-up dp?

- **Bottom-up DP**: Builds from smaller to larger problems.
- **Top-down DP**: Breaks a larger problem into smaller sub-problems.

Fibonacci

- **Bottom-up**: dp[i-2] + dp[i-1] **known**.
- **Top-down**: dp[i-2] + dp[i-1] **initially unknown**.

🥅Goals

- Has better time-efficiency, we don’t have to pay for recursion.
- Opens doors to memory optimization.

🙅‍♂️Non-goals

- Better asymptotic time complexity.

How do we go from top-down to bottom-up dp?

🍳 Recipe: Top-down to bottom-up DP

1. **Represent your state-space** with some kind of data structure
2. Initialize the **base cases** in the data structure
3. Figure-out the **state transitions.**
4. **Fill the answers** in our data-structure (while respecting the dependency order).
5. **Return the answer** for the initial state.

509. Fibonacci Number

Easy 8043 354 Add to List Share YouTube

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

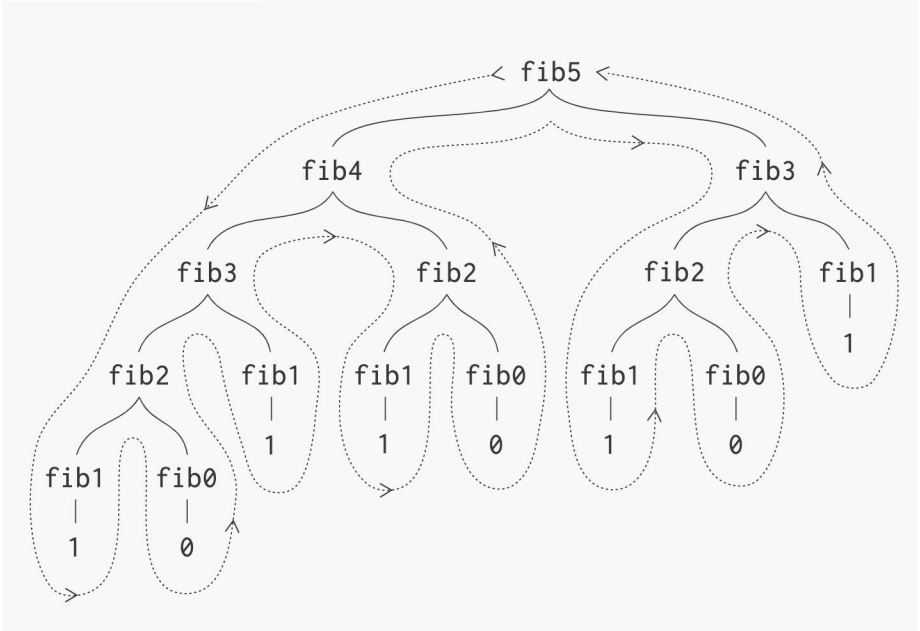
📋Step 1: Represent your state-space.

What would be a good fit?
Array? Matrix?

Step 1: Represent your state-space.

An array, right?

```
dp =[ 0,    1,    1,    2,   ..
    fib0, fib1, fib2, fib3, ..
```



📋Step 2: Initialize the base cases.

What are the base cases?

Step 2: Initialize the base cases.

```
dp[0] = 0
dp[1] = 1
```

```
dp = [0, 1, _, _, _, _, _, …]
```

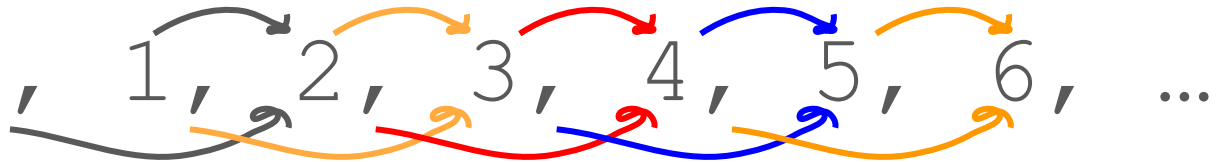
📋Step 3: Figure-out the state transitions.

What do you refer when calculating for fib(x)?

Step 3: Figure-out the state transitions.

Focus on the dependencies.

```
dp = [0, 1, 1, 2, 3, 5, 8, ...]
```



📋Step 4: Fill the answers in our array.

But, in what order?

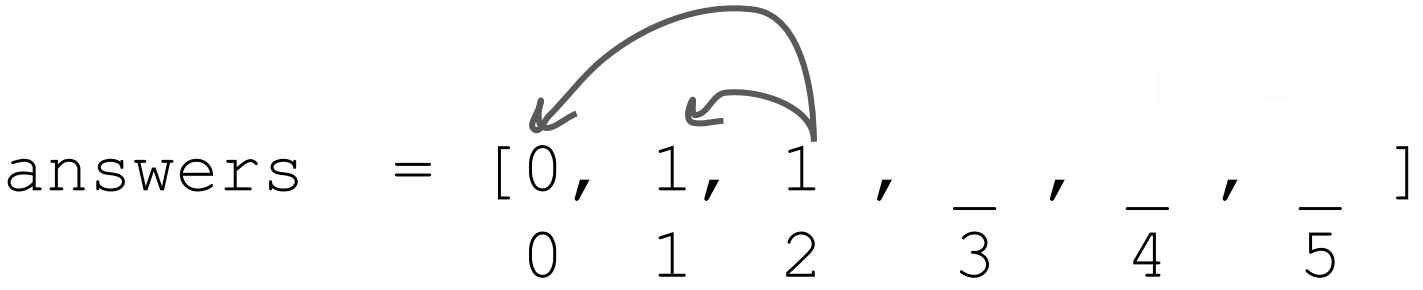
Dependency order:

[0, 1] => 2 => 3 => 4 => 5

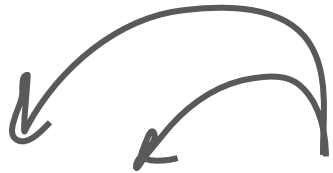
Step 4: Fill the answers in our array.

```
answers  = [0, 1, _ , _ , _ , _ ]
            0  1  2   3   4   5
```

Step 4: Fill the answers in our array.

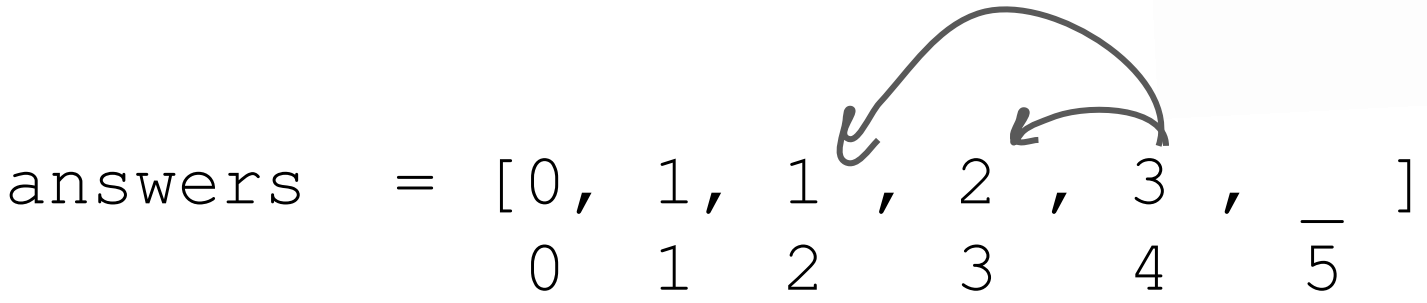


Step 4: Fill the answers in our array.

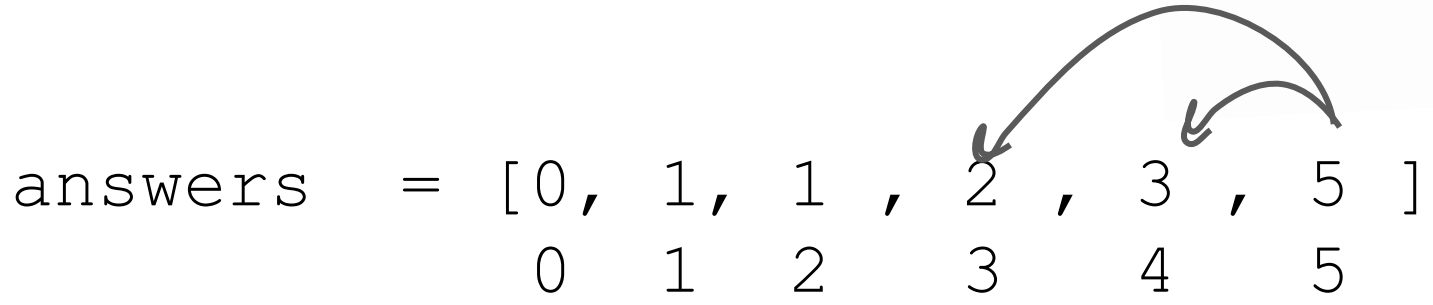


```
answers  = [0, 1, 1 , 2 , _ , _ ]
            0  1  2   3   4   5
```

Step 4: Fill the answers in our array.



Step 4: Fill the answers in our array.



📋Step 5: Extract our answer.

Which cell contains our answer?

Step 5: Extract our answer.

```
answers  = [0, 1, 1 , 2 , 3 , 5 ]
            0  1  2   3   4   5
```

💻Implementation

```python
class Solution:
    def fib(self, n: int) -> int:
        # State space representation
        dp = [0 for _ in range(n + 1)]
        # Base cases
        dp[0], dp[1]  = 0, 1

        # Fill the dp table
        for num in range(2, n + 1):
            dp[num] = dp[num - 1] + dp[num - 2]

        # Extract our answer
        return dp[n]
```

Questions?

Let’s do another one.

62. Unique Paths

Medium 13791 385 Add to List Share

There is a robot on an `m x 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.

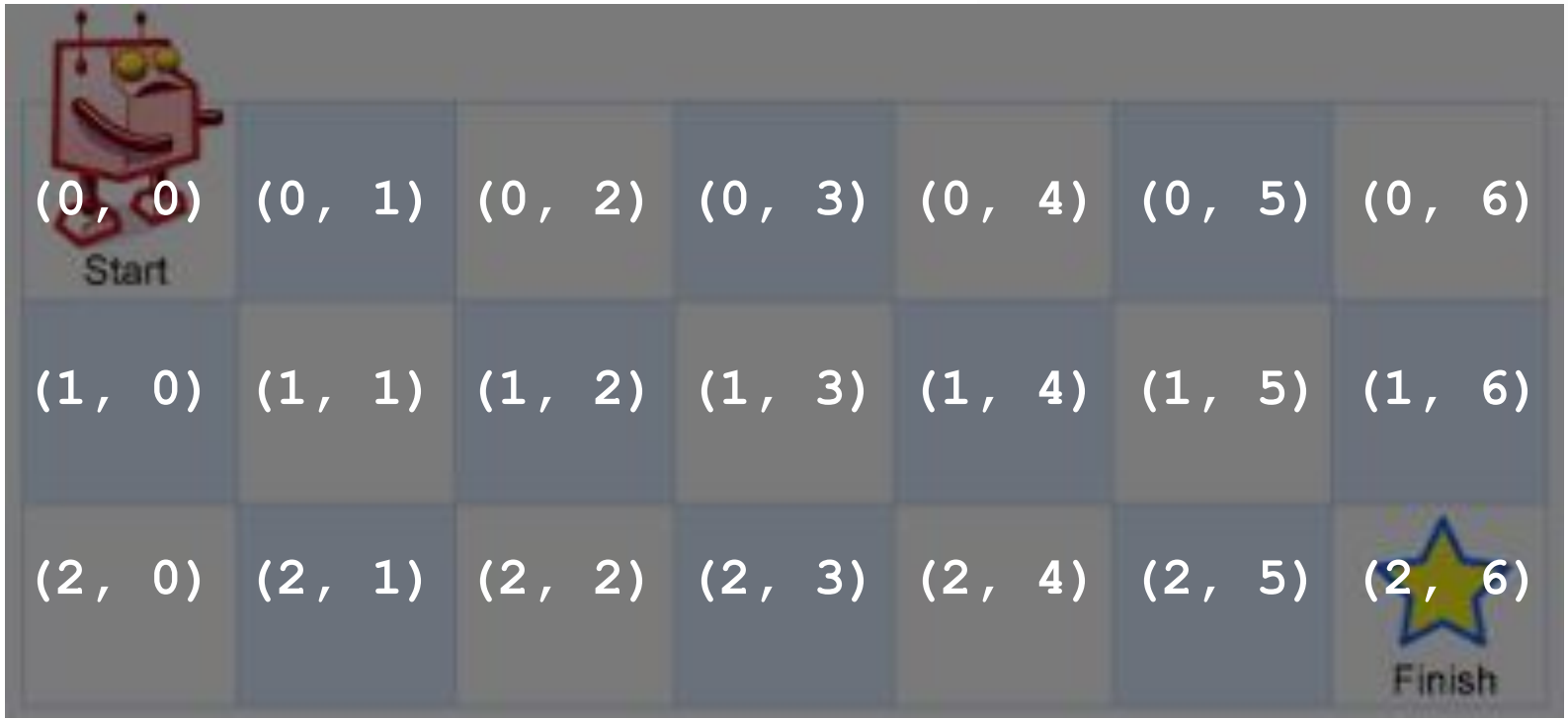
Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

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

⬇️Top-down approach

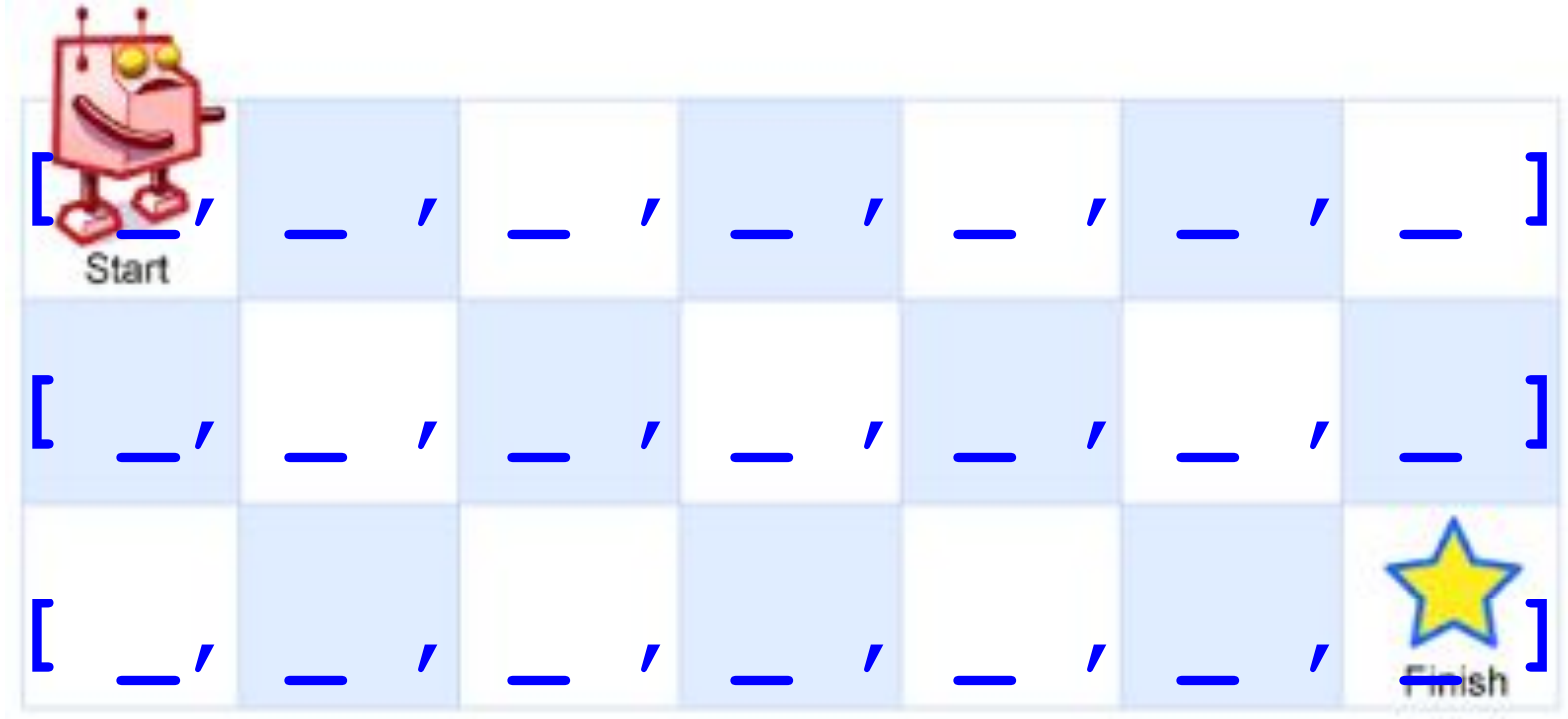
```
class Solution:
    def uniquePaths(self, m: int, n: int) -> in

        @cache
        def uniquePathsRec(coord):
            row , col = coord
            if coord == (m - 1,n - 1):
                return 1
            ways = 0
            if row + 1 < m:
                ways += uniquePathsRec((row + 1, col))
            if col + 1 < n:
                ways += uniquePathsRec((row, col + 1))
            return ways

        return uniquePathsRec((0,0))
```

Step 1: What does the state space look like?



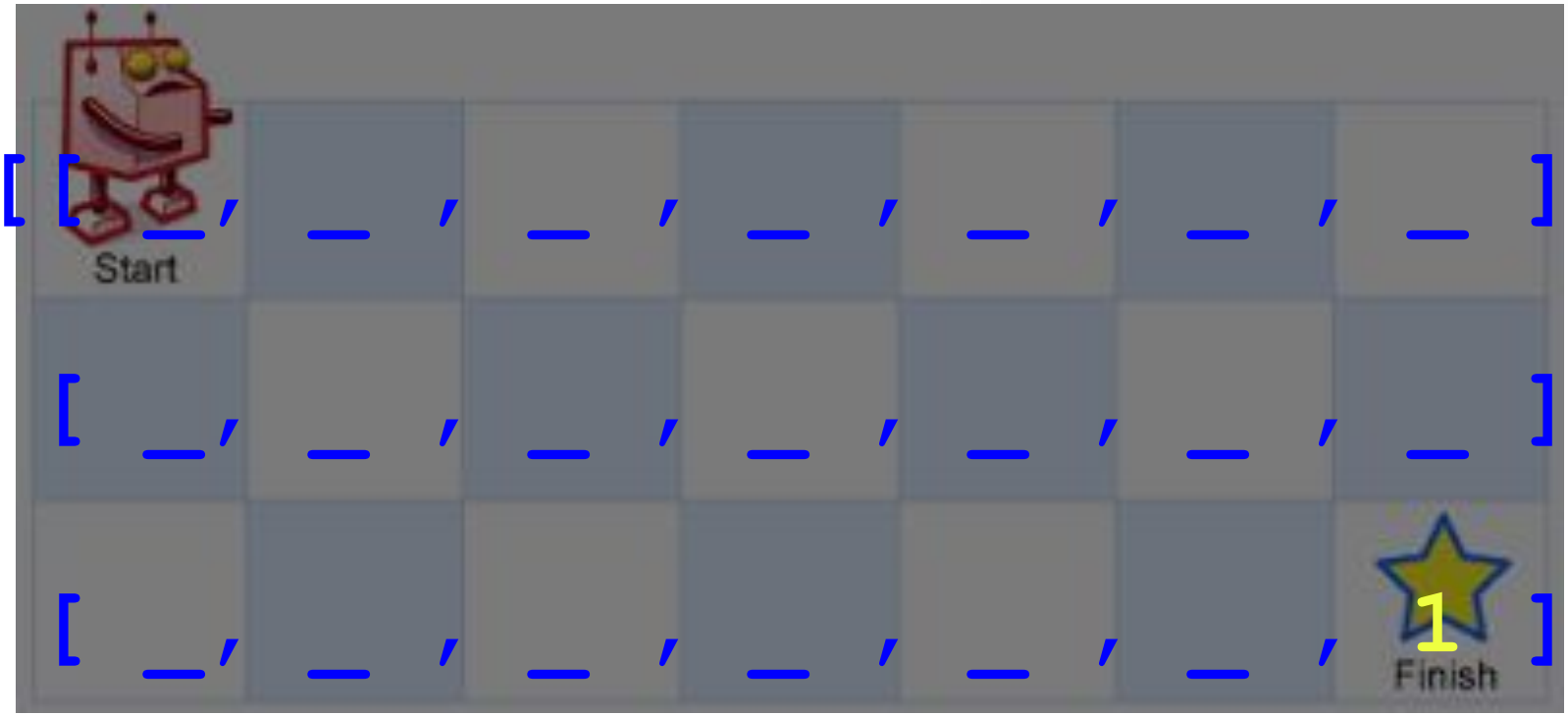
Step 1: What does the state space look like?

```
dp = [ [ _ , _ , _ , _ , _ , _ , _ ]
       [ _ , _ , _ , _ , _ , _ , _ ]
       [ _ , _ , _ , _ , _ , _ , _ ]
]
```

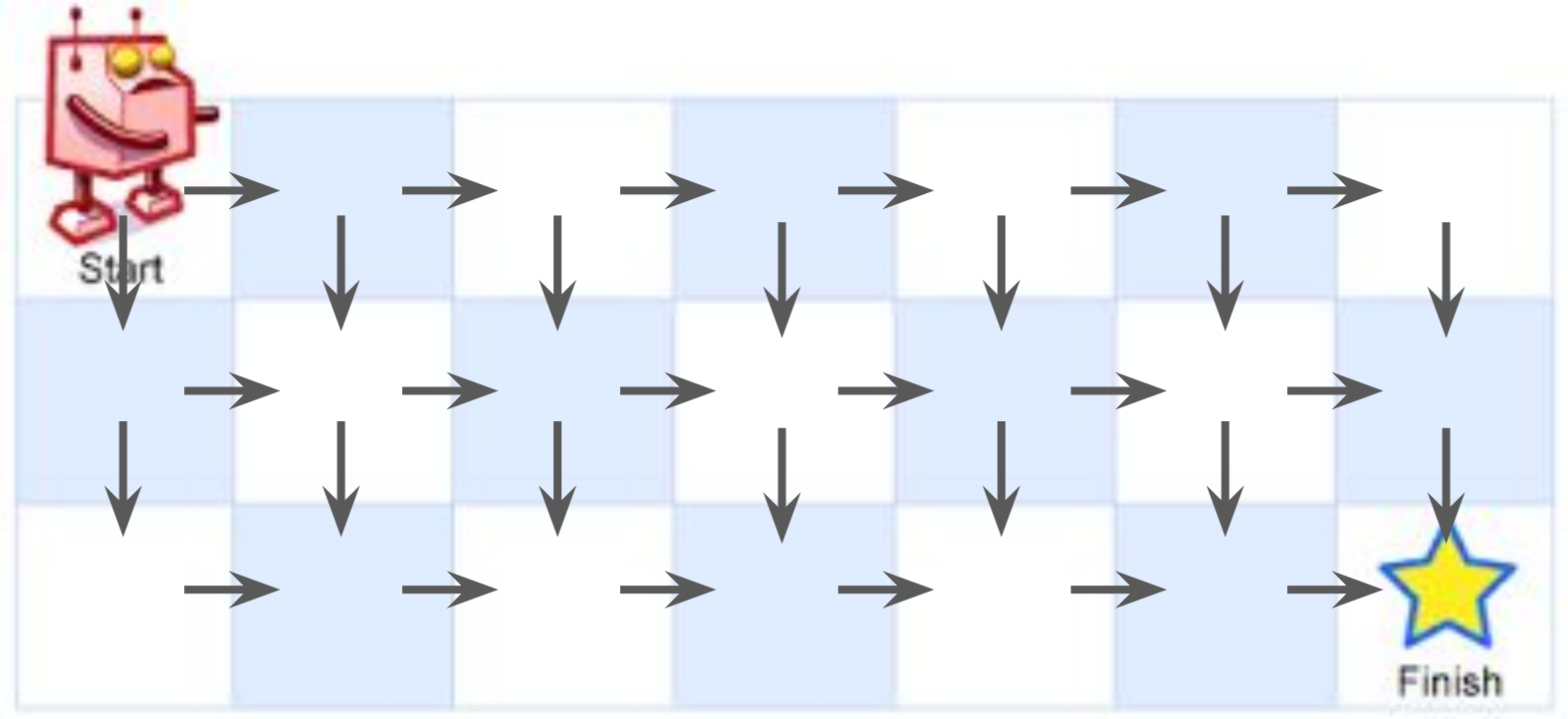


Step 2: What is the base case?

```
dp = [ [ _, _, _, _, _, _, _ ]
       [ _, _, _, _, _, _, _ ]
       [ _, _, _, _, _, _, _ ]
]
```

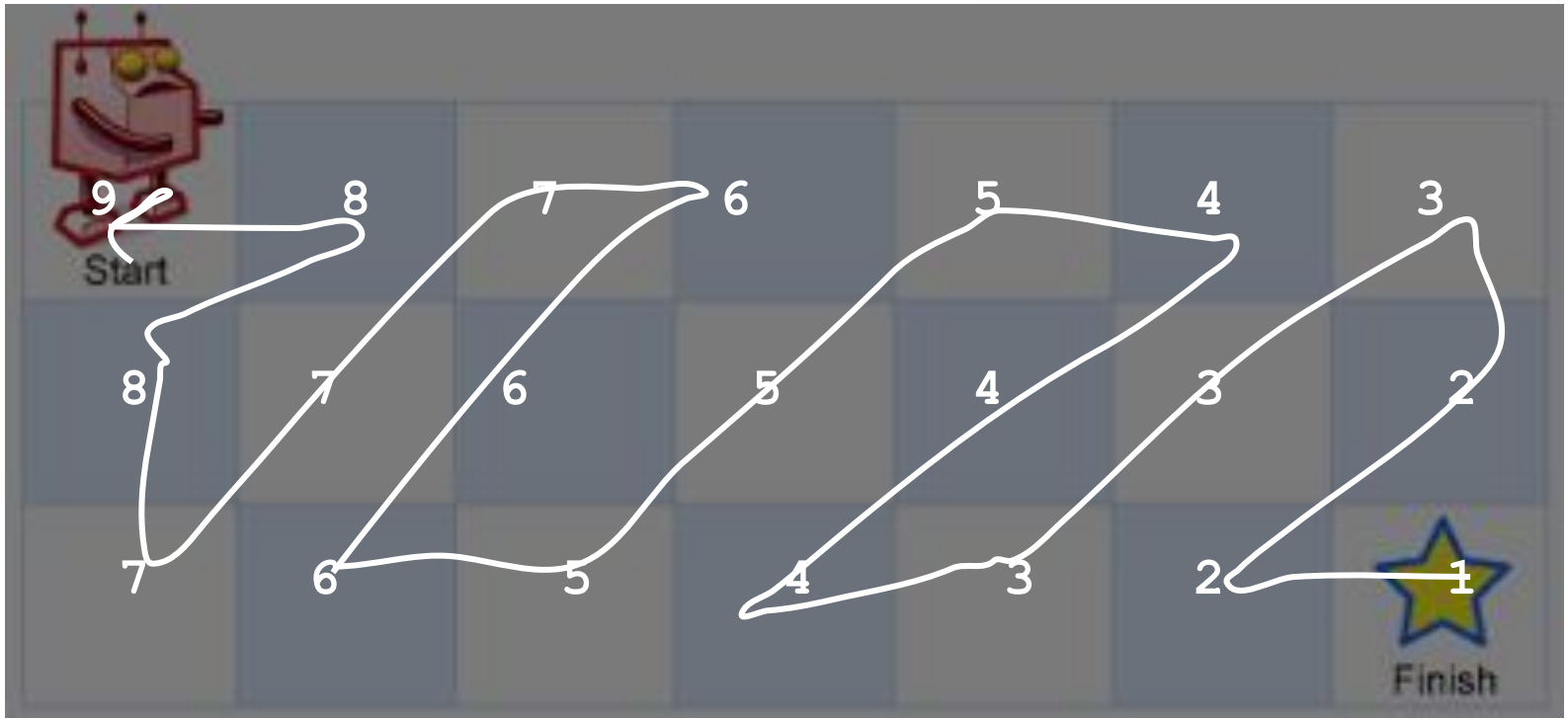


Step 3: What do the moves look like?

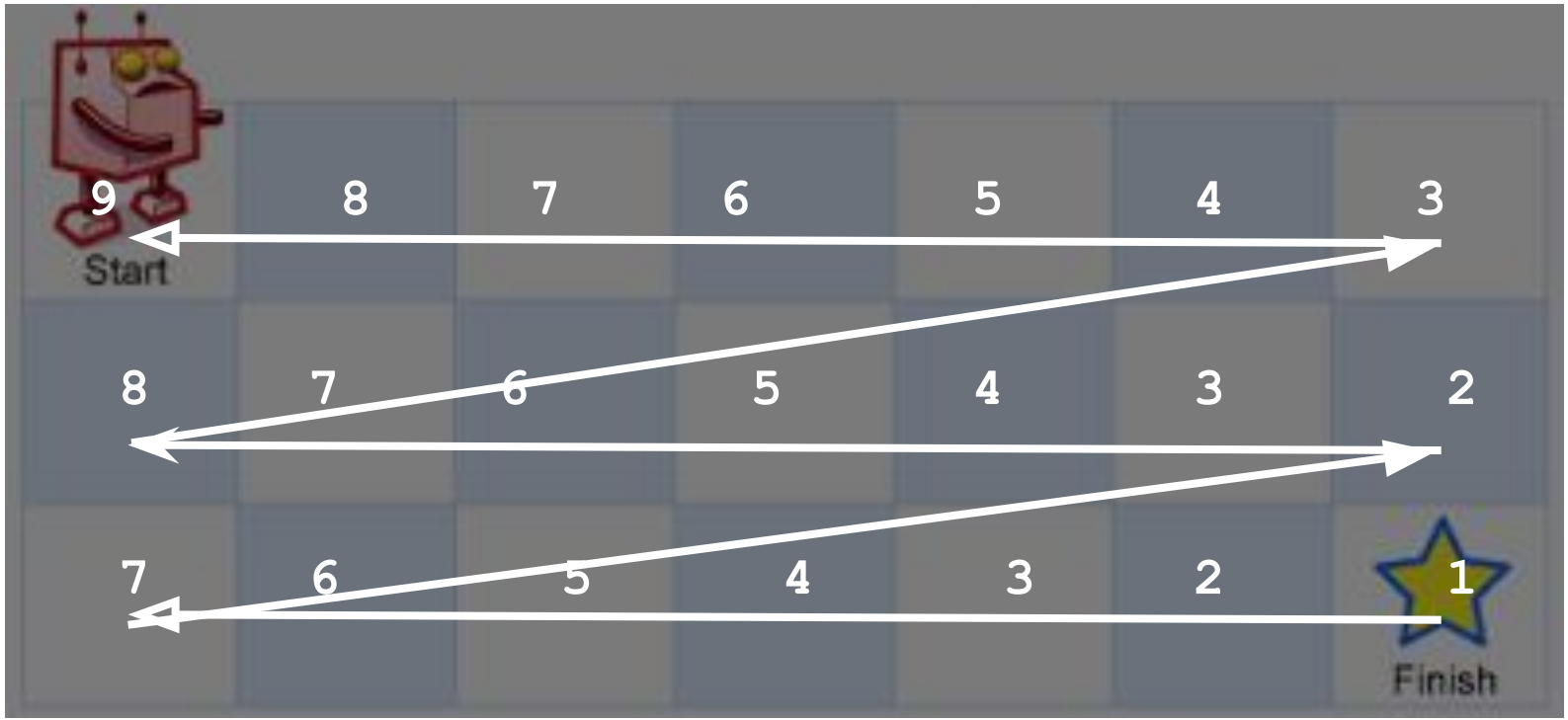


Step 4: Fill the answers in our array.

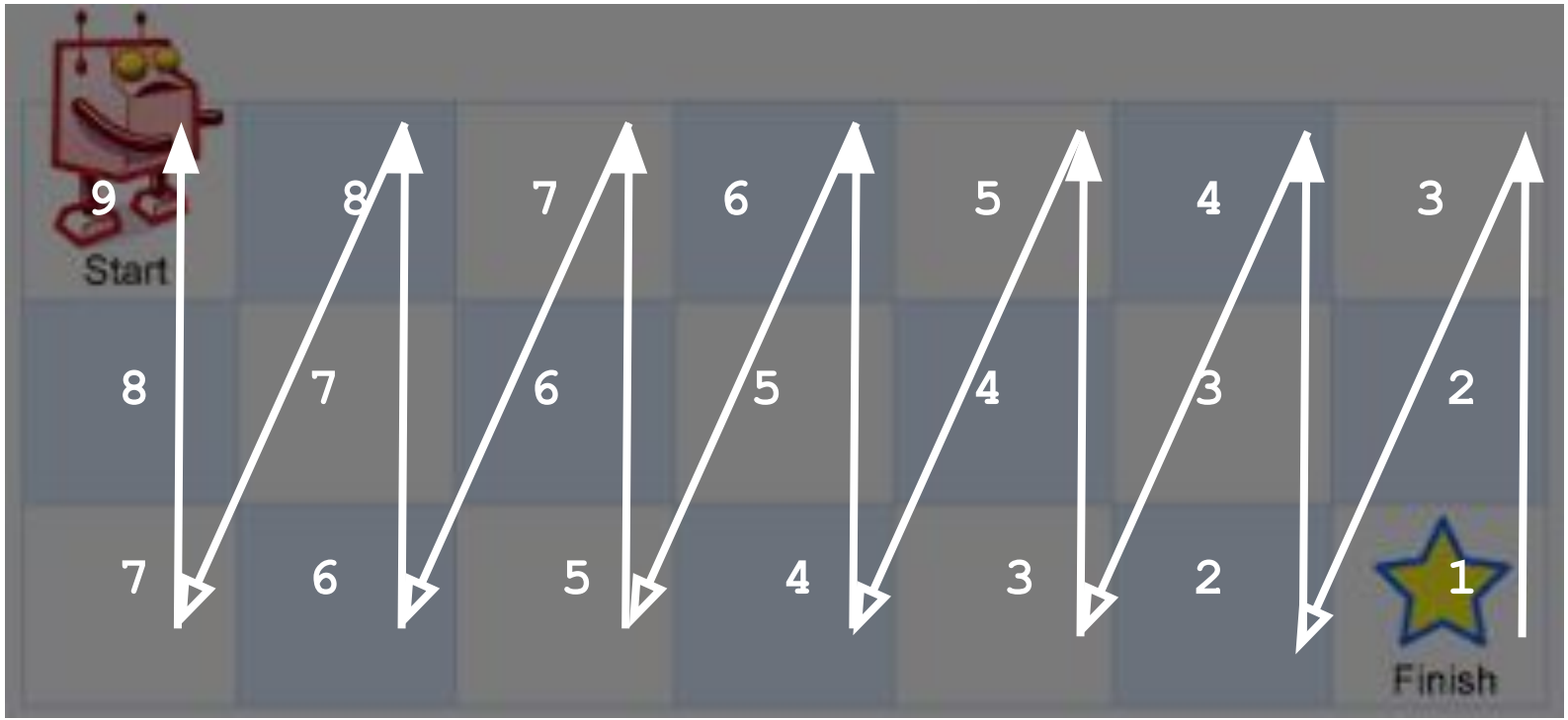
Order 1: Default



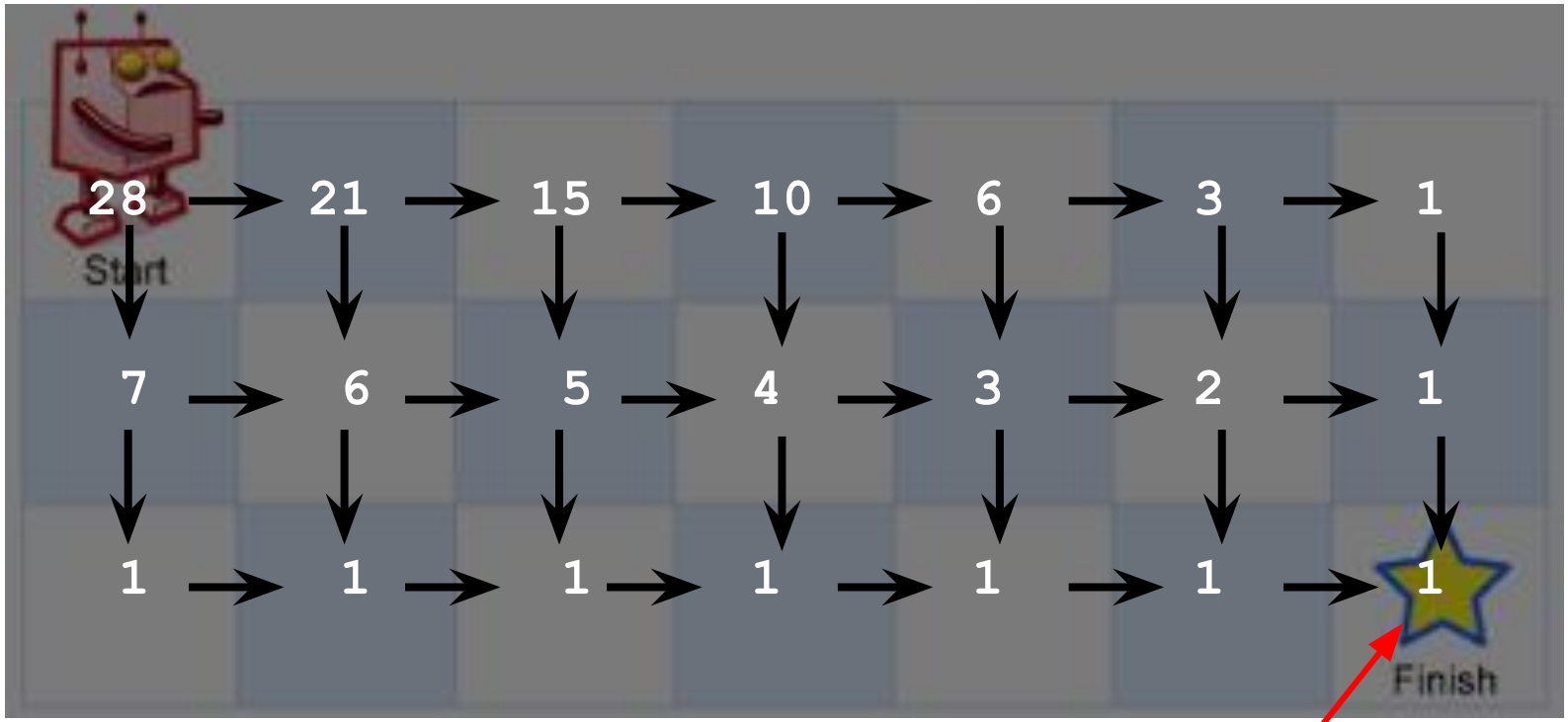
Order 2: Row-wise



Order 3: Column-wise

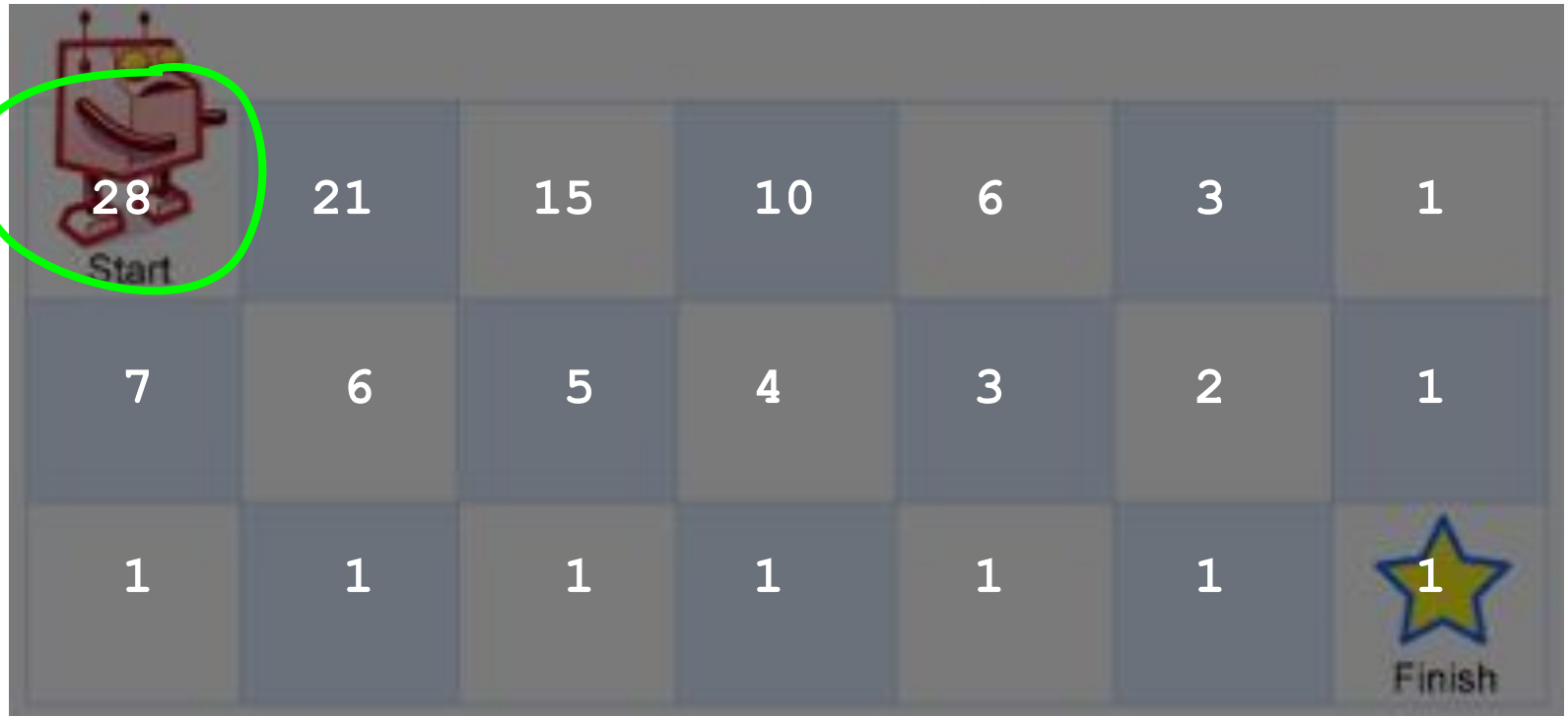


Step 4: Fill the answers in our array.



Base case

Step 5: Extract the answer.



💻Implementation - Row-wise

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[0 for _ in range(n)] for _ in range(m)]
        dp[-1][-1] = 1

        def answerOrDefault(i, j):
            if 0 <= i < m and 0 <= j < n:
                return dp[i][j]
            return 0

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

        return dp[0][0]
```

💻 Implementation - Column-wise

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[0 for _ in range(n)] for _ in range(m)]
        dp[-1][-1] = 1

        def answerOrDefault(i, j):
            if 0 <= i < m and 0 <= j < n:
                return dp[i][j]
            return 0

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

        return dp[0][0]
```

 Practice:

- N-th Tribonacci Number
- Minimum Falling Path Sum

Common Variants

- **Counting** (distinct ways)
  - Usually looks like this: dp[state] = sum(dp[state_1], dp[state_2], dp[..]..)
- **Optimization** (maximum, minimum, and longest/shortest)
  - Usually looks like this: dp[state] = min/max(dp[state_1], dp[state_2], dp[..]..)
- **Decision** (yes or no)
  - Usually looks like this: dp[state] = all/any(dp[state_1], dp[state_2], dp[..]..)

Knapsack:

- Coin Change II
- Solving Questions With Brainpower

DP on Strings:

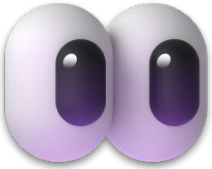
- Is Subsequence
- Distinct Subsequences

DP on Grids:

- Unique Paths II
- Dungeon Game

DP with Bitmask:

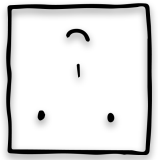
- Partition to K Equal Sum Subsets
- Minimum XOR Sum of Two Arrays



Common Pitfalls

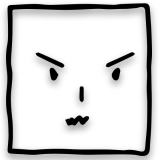
Assuming bottom-up dp has better time complexity

- Both bottom-up and top-down DP **share the same asymptotic time complexity.**
- Despite this, **bottom-up DP is typically more efficient** than top-down DP.



Incorrect Initialization:

```
def dp(states):
    # if we are minimizing, we should use float('inf')
    dp = [[0 for _ in range(len)] for i in range(n+1)]
```



Handling Edge Cases in Bottom-Up DP:

- **Edge cases might be overlooked** in standard bottom-up DP, resulting in errors.
- Ensure consideration of **smallest**, **largest**, and **special input** cases for accurate solutions.

Variant to think about:

- States
  - One Index referring to an element in an array
  - Two indices referring to a range in an array
  - Numerical Constraints like do something in k steps
  - Status on whether some action has taken place or not
  - Bitmask/tuple to indicate something is visited or not visited
- Recurrence Relation
  - Static recurrence relation function: fib(n) = fib(n-1) + fib(n-2)
  - Iteration in your recurrence relation
- State Reduction
  - Trying to represent one state in terms of the others
- Context
  - Array
  - Matrix
  - Graph
  - Simulation

Bottom-up DP Challenge

AtCoder: Educational DP Tasks

Update your status here.

Resources

- **Leetcode General Discussion**
- **Leetcode Explore Card**
- Dynamic Programming lecture #1 - Fibonacci, iteration vs recursion
- Competitive Programmer's Handbook
- Dynamic programming for Coding Interviews: A bottom-up approach to problem solving
- Traveling Salesperson Problem - DP with Bitmask

💪 Practice Questions

Longest Arithmetic Subsequence of Given Difference

Combination Sum IV

Perfect Squares

Triangle

Solving Questions With Brainpower

Minimum Falling Path Sum

Minimum Path Cost in a Grid

Champagne Tower


“We worry top-down, but we invest bottom-up.”
- Seth Klarman