# DYNAMIC PROGRAMMING

## **Dynamic Programming**

**Dynamic Programming (DP)** is an algorithm technique used to solve problems that can be broken down into **simpler, overlapping subproblems.** 

#### **Key Concepts of Dynamic Programming**

- Overlapping subproblems: a problem has overlapping subproblems if it can be broken down into subproblems.
- **Memoization (Top-Down Approach)**: store the results in a cache (typically a dictionary or array) to avoid recalculation recursion and caching approach.
- **Tabulation (Bottom-Up Approach)**: first solve all possible subproblems iteratively, and store them in a table.

## **Common Patterns in Dynamic Programming**

- Toy example (Fibonacci): Climbing Stairs, N-th Tribonacci Number, Perfect Squares
- Constant Transition: Min Cost Climbing Stairs, House Robber, Decode Ways, Minimum Cost For Tickets, Solving Questions With Brainpower
- Grid: Unique Paths, Unique Paths II, Minimum Path Sum, Count Square Submatrices with All Ones, Maximal Square,
   Dungeon Game
- Dual-Sequence: Longest Common Subsequence, Uncrossed Lines, Minimum ASCII Delete Sum for Two Strings, Edit
   Distance, Distinct Subsequences, Shortest Common Supersequence
- Interval: Longest Palindromic Subsequence, Stone Game VII, Palindromic Substrings, Minimum Cost Tree From Leaf Values, Burst Balloons, Strange Printer
- Longest Increasing Subsequence: Count Number of Teams, Longest Increasing Subsequence, Partition Array for Maximum Sum, Largest Sum of Averages, Filling Bookcase Shelves
- Knapsack: Partition Equal Subset Sum, Number of Dice Rolls With Target Sum, Combination Sum IV, Ones and Zeroes,
   Coin Change, Coin Change II, Target Sum, Last Stone Weight II, Profitable Schemes
- Topological Sort on Graphs: Longest Increasing Path in a Matrix, Longest String Chain, Course Schedule III
- DP on Trees: House Robber III, Binary Tree Cameras
- Other problems: 2 Keys Keyboard, Word Break, Minimum Number of Removals to Make Mountain Array, Out of Boundary Paths

#### **Credits**

### Dynamic Programming – Example – Fibonacci Sequence

```
Naive Recursive Approach

int fib(int n) {
   if (n <= 1) {
      return n;
   }
   return fib(n - 1) + fib(n - 2);
}</pre>
```

```
Memoization (Top-Down DP)

std::unordered_map<int, int> memo;

int fib(int n) {
    if (n <= 1) {
        return n;
    }
    if (memo.find(n) != memo.end()) {
        return memo[n];
    }
    memo[n] = fib(n - 1) + fib(n - 2);
    return memo[n];
}</pre>
```

```
Tabulation (Bottom-up DP)

int fib(int n) {
    if (n <= 1) {
        return n;
    }
    int dp[n + 1];
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}</pre>
```

## **Problem – Climbing Stairs**



leetcode.com/problems/climbing-stairs

#### **Problem Statement**

You need to climb a staircase with n steps to get to the top. Each time you can choose to climb either 1 step or 2 steps at a time. Find out how many different ways you can climb to the top of the staircase.

#### Example 1

Input: n = 2

Output: 2

Explanation: There are two ways to get to the top

- 1. Climb 1 step at a time, twice
- 2. Climb 2 steps in one go

#### Example 2:

**Input:** n = 3

Output: 3

**Explanation:** There are three ways to get to the top:

- 1. Climb 1 step at a time, three times
- 2. Climb 1 step, then 2 steps
- 3. Climb 2 steps, then 1 ste.

### Solution – Climbing Stairs



leetcode.com/problems/climbing-stairs

```
std::unordered map<int, int> memo;
int climbStairs(int n) {
   // Identify the sequence, when:
   // n = 0 (0 way), there is no way to get up
   // n = 1 (1 way): only one way : 1-step
   // n = 2 (2 ways): 1s + 1s | 2s
   // n = 3 (3 ways): 1s + 1s + 1s | 1s + 2s | 2s + 1s
   // n = 4 (5 ways): 1s + 1s + 1s + 1s | 1s + 1s + 2s | 1s + 2s + 1s | 2s + 1s + 1s | 2s + 2s |
   if (n <= 2) {
       return n;
   if (memo.find(n) != memo.end()) {
       return memo[n];
   memo[n] = climbStairs(n - 1) + climbStairs(n - 2);
   return memo[n];
```

## **Problem – 1143. Longest Common Subsequence**



**LeetCode** 

https://leetcode.com/problems/longest-common-subsequence

Problem Statement / Solution / Code Time: O(-) Space: O(-)

•

# Problem – 62. Unique Paths



**LeetCode** 

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

#### Problem Statement / Solution / Code Time: O(-) Space: O(-)

• ...

# **EOF**

# Tips

#### Problem Statement / Solution / Code Time: O(n) Space: O(n)

• ..

### Problem – number. name









leetcode.com/problems/...

#### Problem Statement / Solution / Code Time: O(-) Space: O(-)

• ..