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
https://www.youtube.com/watch?v=9k31KcQmS_U
https://algo.monster/problems/dp-list

Dynamic Programming – Example – Fibonacci Sequence

Naive Recursive Approach $O(2^n)$

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

Memoization (Top-Down DP) $O(n)$

```
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];
}
```

Tabulation (Bottom-up DP) $O(n)$

```
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];
}
```

Problem – Climbing Stairs

Easy

LeetCode 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

Easy

LeetCode 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

Medium

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

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- …

Problem – 62. Unique Paths

Medium

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

Problem

- The robot is placed in a m x n grid
- It starts at the top-left cell (0,0) and must reach the bottom-right (m – 1, n – 1)
- The robot can only move right or down at any point
- Return the number of unique paths the robot can take to reach the destination

Solution – 62. Unique Paths

Medium

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

Solution 1 (recursive)

- Define a recursive function countPaths(m, n)
- **Base case**

  If m == 1 or n == 1, there's only one way to reach that cell (either all downs or all rights).
- **Recursive case**

  To reach cell (m, n) the robot must come from:

  Cell (m - 1, n) → from above

  Cell (m, n - 1) → from left

  So the number of of paths to (m, n) is the **sum** of the paths to those two cells
- **Memoization**

  Use a 2D vector[m + 1][n + 1]

Code – 62. Unique Paths

Medium

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

Code

Time: **O(m * n)** Space: **O(m * n)**

```
int countPaths(int m, int n, vector<vector<int>>& memo) {
    if (m == 1 || n == 1) return 1;
    if (memo[m][n] != -1) return memo[m][n];
    memo[m][n] = countPaths(m, n - 1, memo) + countPaths(m - 1, n, 
memo);
    return memo[m][n];
}
int uniquePaths(int m, int n) {
    /*
      count(m, n) = count(m, n + 1) + count(m + 1, n)
      same as (imagine robot going from m,n to 0,0 up and left)
      count(m, n) = count(m, n - 1) + count(m - 1, n)
    */
    vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));
    return countPaths(m, n, memo);
}
```

Solution – 62. Unique Paths

Medium

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

Solution 2 (iterative)

- Create a 2D vector dp of size (m+1) × (n+1) to store intermediate results
- Set dp[1][1] = 1 because there is exactly one way to stand on the starting cell
- Iterate through each cell (row, col) from (1, 1) to (m, n):

  - Skip (1, 1) since it's already initialized

  - For every other cell, the number of unique paths to it is the sum of:

    - Paths from the cell above: dp[row-1][col]

    - Paths from the cell to the left: dp[row][col-1]

  dp[row][col] = dp[row - 1][col] + dp[row][col - 1]

Code – 62. Unique Paths

Medium

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

Code

Time: **O(m * n)** Space: **O(m * n)**

```
int uniquePaths(int m, int n) {
    vector<vector<int>> dp(m + 1, vector<int>(n + 1));
    dp[1][1] = 1;
    for (int row = 1; row <= m; ++row) {
        for (int col = 1; col <= n; ++col) {
            if (row == 1 && col == 1) continue;
            dp[row][col] = dp[row-1][col] + dp[row][col-1];
        }
    }
    return dp[m][n];
}
```

```
// Optimized 1DP
int uniquePaths(int m, int n) {
    vector<int> dp(n, 1); // base case: first row is all 1s
    for (int row = 1; row < m; ++row) {
        for (int col = 1; col < n; ++col) {
            dp[col] = dp[col] + dp[col - 1];
        }
    }
    return dp[n - 1];
}
```

Solution – 62. Unique Paths

Medium

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

Solution 3 (combinatorics)

- **Grid size:** m x n
- **Start:** top-left cell (0, 0)
- **End:** bottom-right cell (m – 1, n – 1)
- To get from the top-left to the bottom-right:

  You must move exactly m - 1 times down

  And exactly n - 1 times right

  These two types of moves must be made in some order, with a total of:

$$(m - 1) + (n - 1) = m + n - 2 \ moves$$

- **Hence,** from a sequence of m + n - 2 moves, choose m - 1 of them to be down moves (the rest will be right), or vice versa

$$\boldsymbol{Number\ of\ unique\ paths} = \binom{\boldsymbol{m+n-2}}{\boldsymbol{m-1}} = \frac{(\boldsymbol{m+n-2})!}{(\boldsymbol{m-1})!\,(\boldsymbol{n-1})!}$$

Code – 62. Unique Paths

Medium

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

Code

Time: **O(min(m,n))** Space: **O(1)**

```
int uniquePaths(int m, int n) {
    // we will compute the binomial coefficient:
    // (m + n - 2) choose (m - 1) => total moves choose down moves
    // = (m + n - 2)! / ((m - 1)! * (n - 1)!)

    long long res = 1;

    // we compute the result iteratively to avoid large factorials
    // res = (n) * (n+1) * ... * (m+n-2) / (1 * 2 * ... * (m - 1))

    for (int i = 1; i <= m - 1; ++i) {
        // multiply numerator: (n - 1 + i)
        // divide by denominator: i
        res = res * (n - 1 + i) / i;
    }

    return (int)res;
}
```

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Problem

- You are given two arrays of integers, days and costs
- Days represent

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Solution

- ...

Problem – 983. Minimum Cost For Tickets

Medium

LeetCode leetcode.com/problems/minimum-cost-for-tickets

Code

Time: **O(-)** Space: **O(-)**

- …

EOF

Tips

Problem Statement / Solution / Code

Time: **O(n)** Space: **O(n)**

- …

Problem – number. name

Easy Hard Medium

LeetCode leetcode.com/problems/...

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- 1
- 2