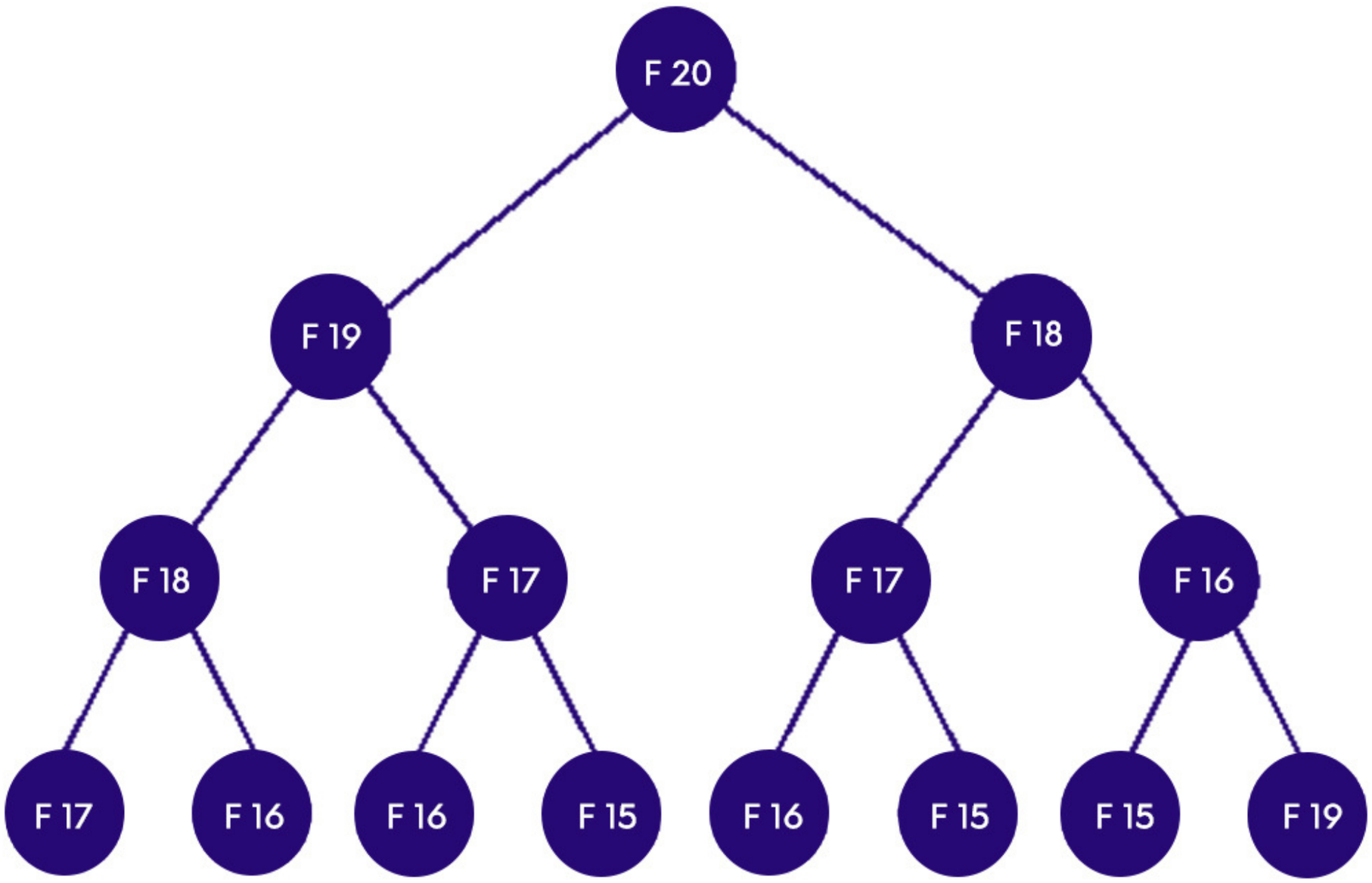
AlgoTutor

# MASTER DYNAMIC PROGRAMMING IN JUST 10 DAYS



AlgoTutor

Day 1

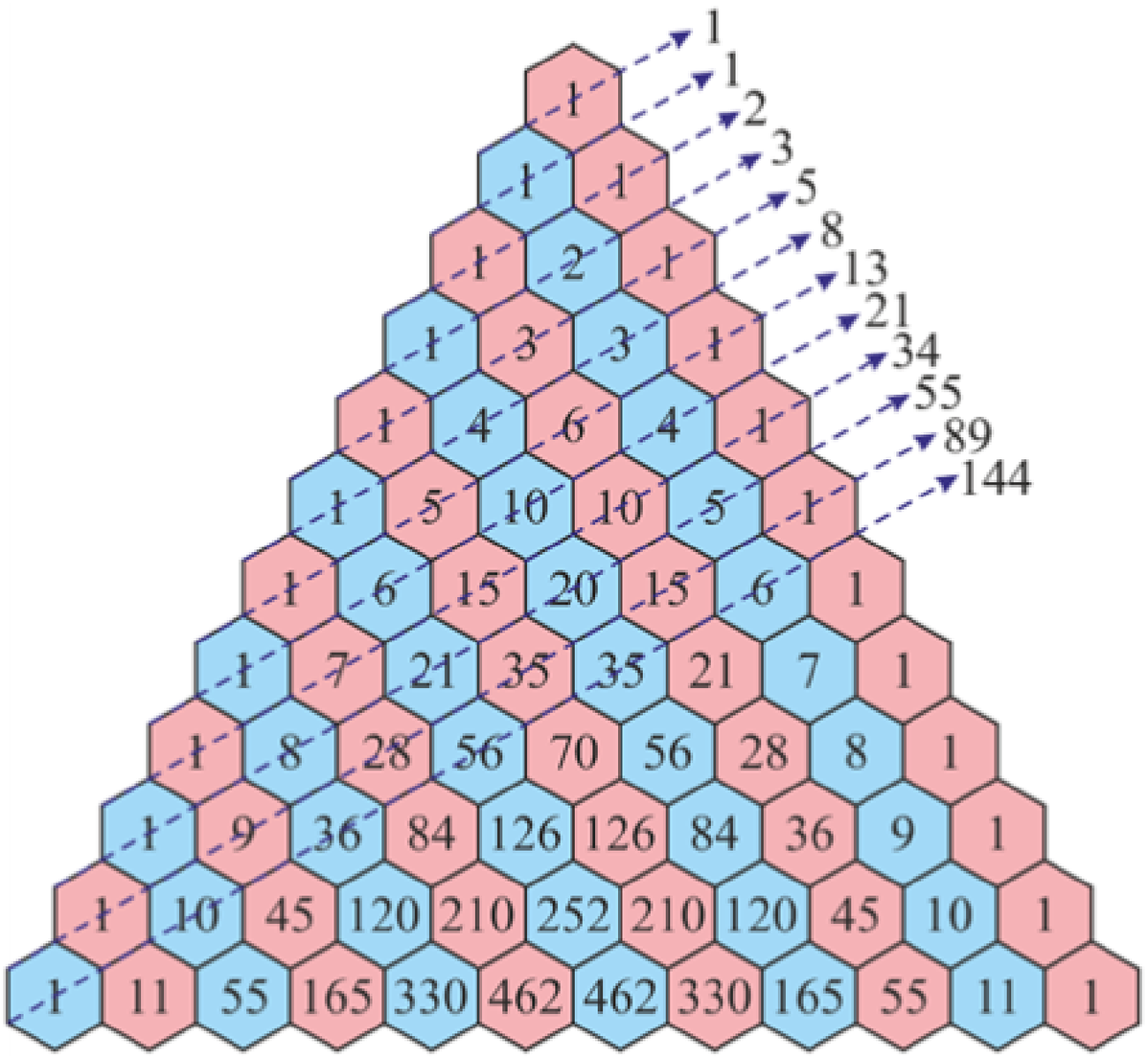
## Understanding the Basics of DP

- Learn the fundamental concepts of dynamic programming, such as memoization and bottom-up approach.
- Study basic examples like Fibonacci and the 0/1 Knapsack problem.

Day 2

## Fibonacci Series

- Solve Fibonacci problems using both recursive and dynamic programming approaches.
- Understand the time and space complexity differences.



AlgoTutor

Day 3

# 0/1 Knapsack Problem

- Study the 0/1 Knapsack problem and solve it using dynamic programming.
- Explore variations of the Knapsack problem, such as fractional and unbounded Knapsack.

| Increasing weights (→) | | 0(w→) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_i$ | $w_i$ | 0(i↓) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

Day 4

# Longest Common Subsequence (LCS)

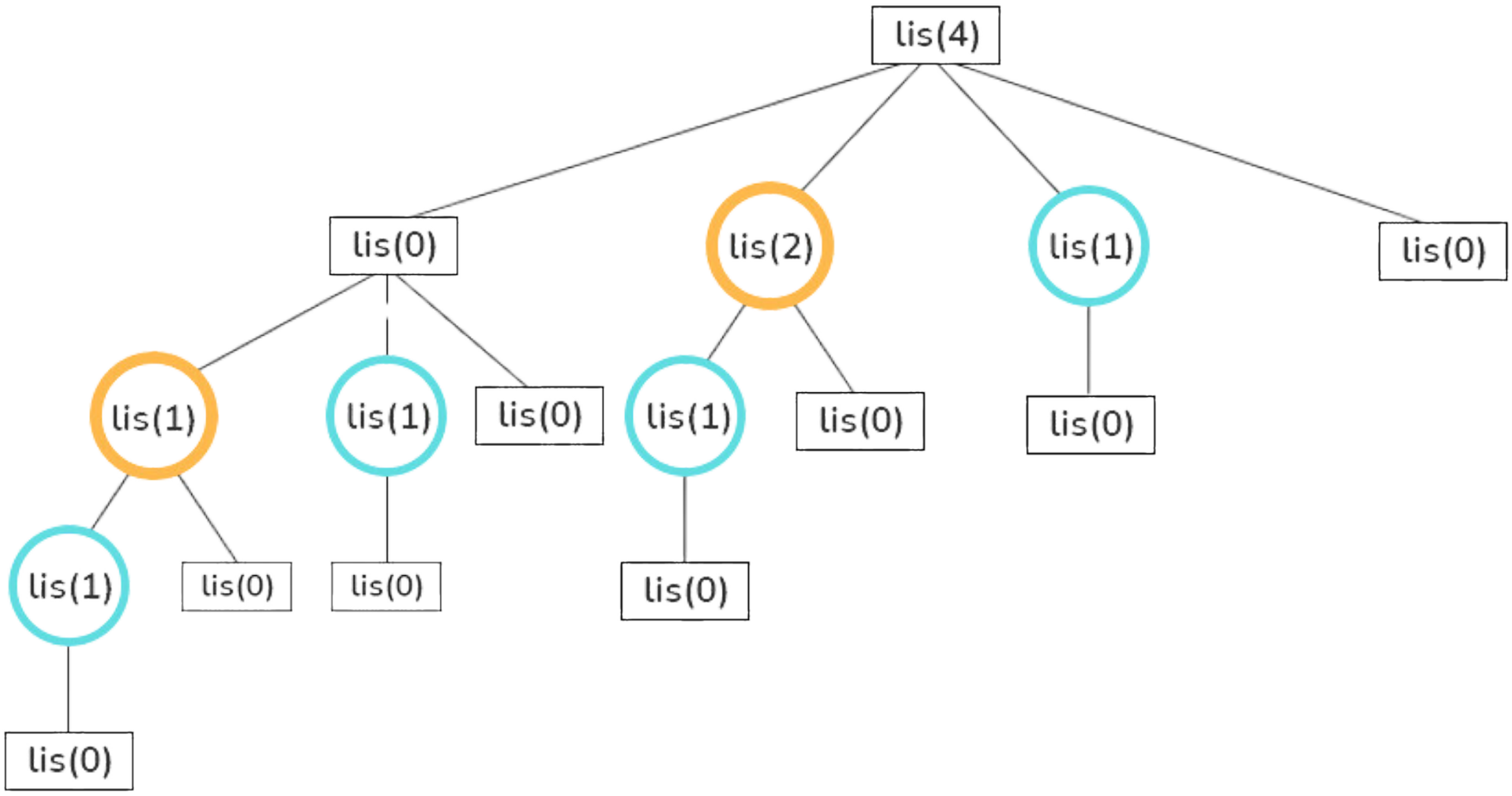
- Learn about the Longest Common Subsequence problem.
- Solve LCS using dynamic programming and understand the table-based approach.

AlgoTutor

Day 5

# Longest Increasing Subsequence (LIS)

- Learn about load balancers and their role in distributing traffic.
- Study load balancing algorithms and strategies.




Day 6

# Coin Change Problem

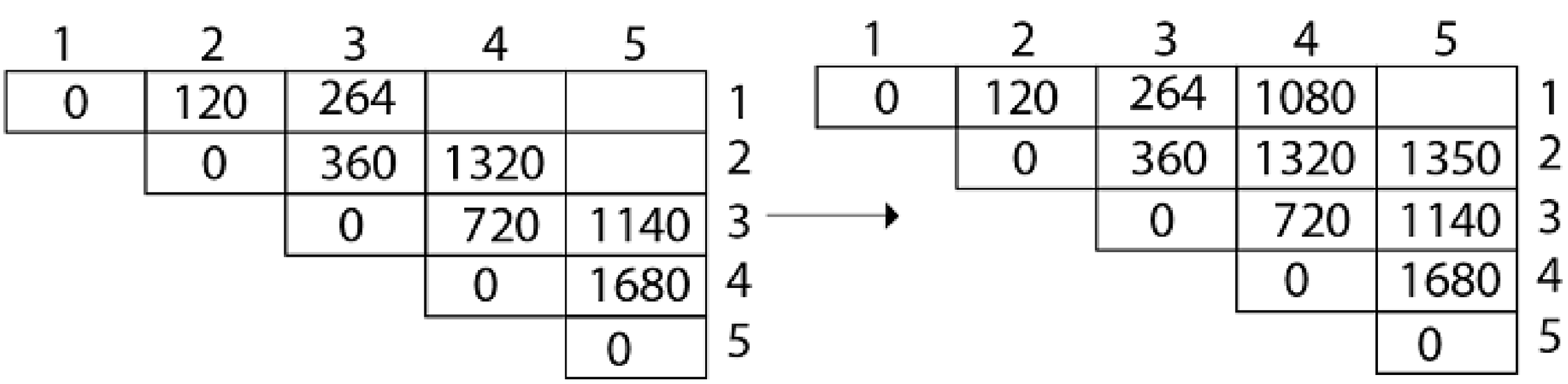
- Explore the Coin Change problem.
- Solve it using dynamic programming and variations like finding the number of ways to make change.

AlgoTutor

Day 7

# Matrix Chain Multiplication

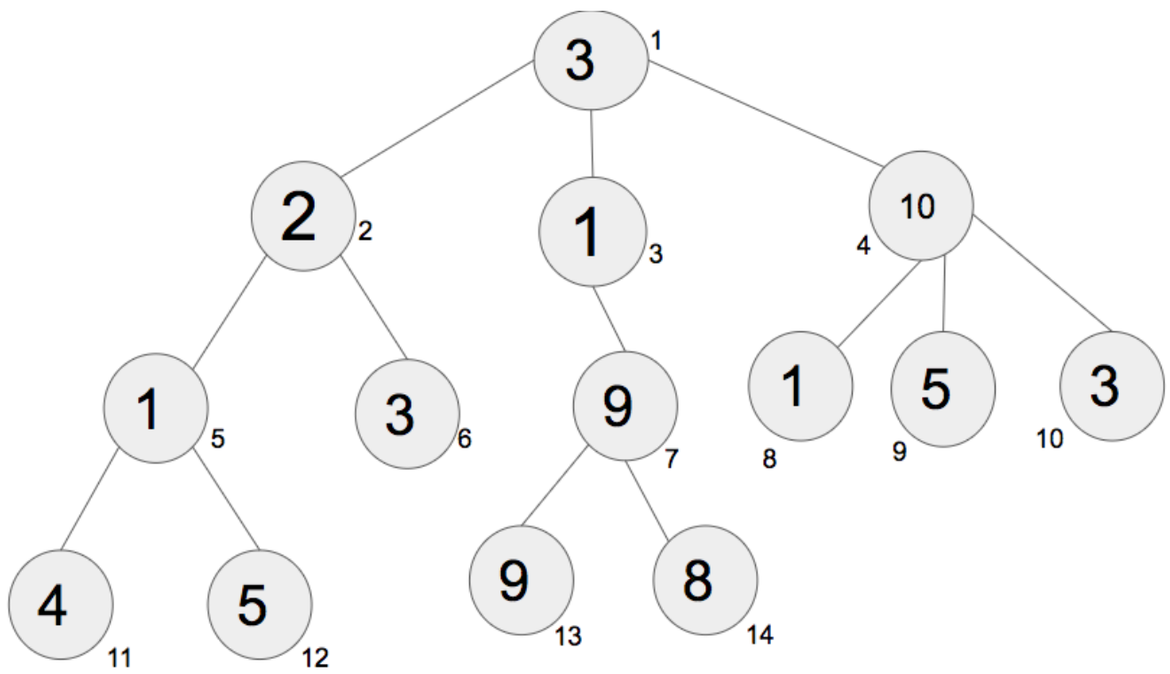
- Understand the Matrix Chain Multiplication problem.
- Solve it using dynamic programming and explore the parenthesization of matrices.




Day 8

# Dynamic Programming on Trees

- Study DP problems on trees, such as finding the diameter of a tree or the largest independent set in a tree.



AlgoTutor

Day 9

## Practice and Review

- Review all the DP problems you've solved so far.
- Understand the trade-offs between Consistency, Availability, and Partition Tolerance.

Day 10

## Advanced Dynamic Programming

- Challenge yourself with advanced DP problems, such as the Travelling Salesman Problem (TSP), the Edit Distance problem, or the Maximum Subarray Sum problem.

AlgoTutor

# Important Practice Questions

## 01. Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Example 1:**

**Input:** n = 2

**Output:** 2

**Example 2:**

**Input:** n= 3

**Output:** 3

**Practice**

# 02. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Example 2:**

**Input:** nums = [2,7,9,3,1]

**Output:** 12

**Practice**

# 03. House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

**Example 1:**

**Input:** nums = [2,3,2]

**Output:** 3

**Example 2:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Practice**

# 04. Unique Paths

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 * 109.

**Example 1:**


**Input:** m = 3, n = 7

**Output:** 28

**Practice**

AlgoTutor

# 05. Unique Paths II

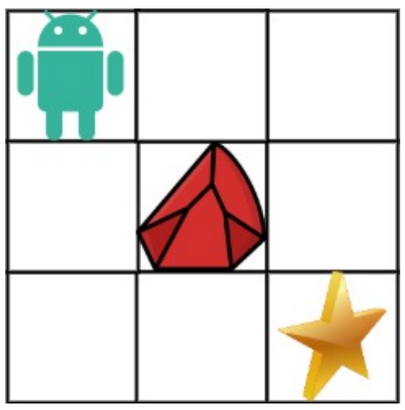
You are given an m x n integer array grid. There is a robot initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

An obstacle and space are marked as 1 or 0 respectively in grid. A path that the robot takes cannot include any square that is an obstacle.

Return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The testcases are generated so that the answer will be less than or equal to 2 * 109.

**Example 1:**



**Input:** obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]

**Output:** 2

**Practice**

# 06. Minimum Path Sum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

**Example 1:**

| | | |
|---|---|---|
| 1 | 3 | 1 |
| 1 | 5 | 1 |
| 4 | 2 | 1 |

**Input:** grid = [[1,3,1],[1,5,1],[4,2,1]]

**Output:** 7

**Example 2:**

**Input:** grid = [[1,2,3],[4,5,6]]

**Output:** 12

**Practice**

# 07. Triangle

Given a triangle array, return the minimum path sum from top to bottom.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index i + 1 on the next row.

**Example 1:**
**Input:** triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
**Output:** 11

**Example 2:**
**Input:** triangle = [[-10]]
**Output:** -10

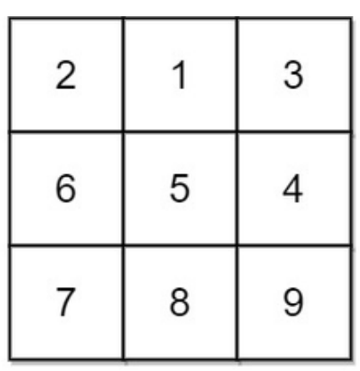
**Practice**

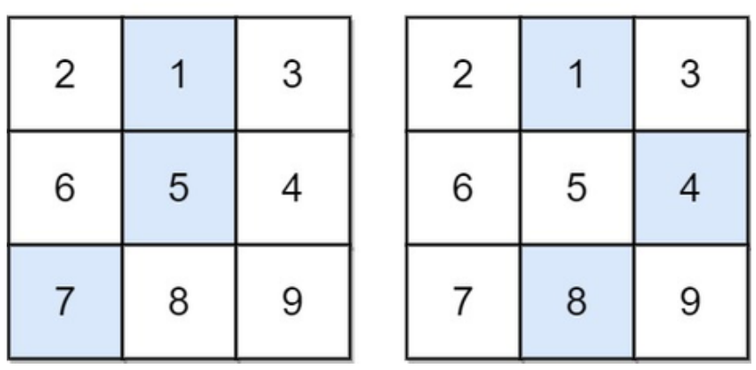
# 08. Minimum Falling Path Sum

Given an n x n array of integers matrix, return the minimum sum of any falling path through matrix.

A **falling path** starts at any element in the first row and chooses the element in the next row that is either directly below or diagonally left/right. Specifically, the next element from position (row, col) will be (row + 1, col - 1), (row + 1, col), or (row + 1, col + 1).

**Example 1:**




**Input:** matrix = [[2,1,3],[6,5,4],[7,8,9]]

**Output:** 13

## Practice

# 09. Partition Equal Subset Sum

Given an integer array nums, return true if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or false otherwise.

**Example 1:**

**Input:** nums = [1,5,11,5]

**Output:** true

**Example 2:**

**Input:** nums = [1,2,3,5]

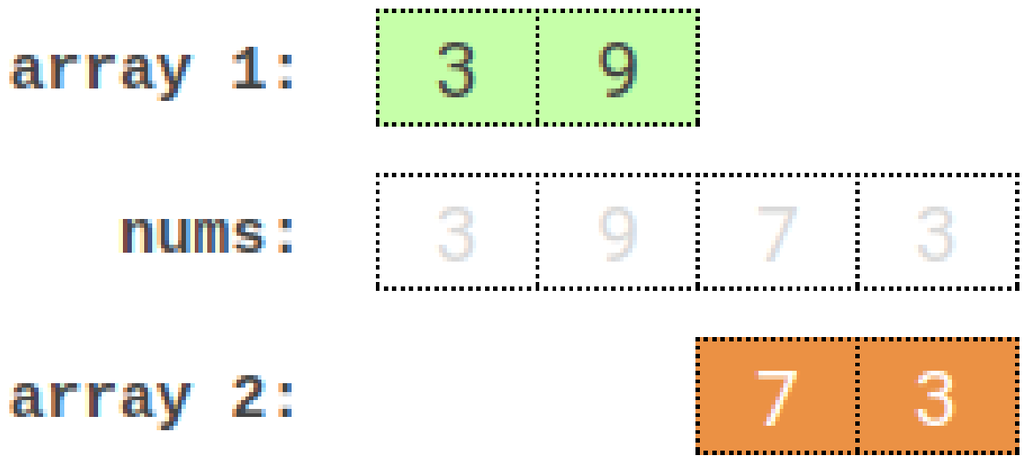
**Output:** false

# 10. Partition Array Into Two Arrays to Minimize Sum Difference

You are given an integer array nums of 2 * n integers. You need to partition nums into two arrays of length n to minimize the absolute difference of the sums of the arrays. To partition nums, put each element of nums into one of the two arrays.

Return the minimum possible absolute difference.

**Example 1:**




**Input:** nums = [3,9,7,3]

**Output:** 2

**Practice**

# 11. Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

**Example 1:**

**Input:** coins = [1,2,5], amount = 11

**Output:** 3

**Example 2:**

**Input:** coins = [2], amount = 3

**Output:** -1

**Practice**

# 12. Target Sum

You are given an integer array nums and an integer target.

You want to build an **expression** out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.

- For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1".

Return the number of different expressions that you can build, which evaluates to target.

**Example 1:**

**Input:** nums = [1,1,1,1,1], target = 3

**Output:** 5

**Example 2:**

**Input:** nums = [1], target = 1

**Output:** 1

**Practice**

# 13. Coin Change II

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the number of combinations that make up that amount. If that amount of money cannot be made up by any combination of the coins, return 0.

You may assume that you have an infinite number of each kind of coin.
The answer is **guaranteed** to fit into a signed **32-bit** integer.

**Example 1:**

**Input:** amount = 5, coins = [1,2,5]

**Output:** 4

**Example 2:**

**Input:** amount = 3, coins = [2]

**Output:** 0

**Practice**

# 14. Longest Common Subsequence

Given two strings text1 and text2, return the length of their longest **common subsequence**. If there is no **common subsequence**, return 0.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, "ace" is a subsequence of "abcde".

A **common subsequence** of two strings is a subsequence that is common to both strings.

**Example 1:**

**Input:** text1 = "abcde", text2 = "ace"

**Output:** 3

**Example 2:**

**Input:** text1 = "abc", text2 = "abc"

**Output:** 3

**Practice**

# 15. Longest Palindromic Subsequence

Given a string s, find the longest palindromic **subsequence**'s length in s.

A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

**Example 1:**

**Input:** s = "bbbab"

**Output:** 4

**Example 2:**

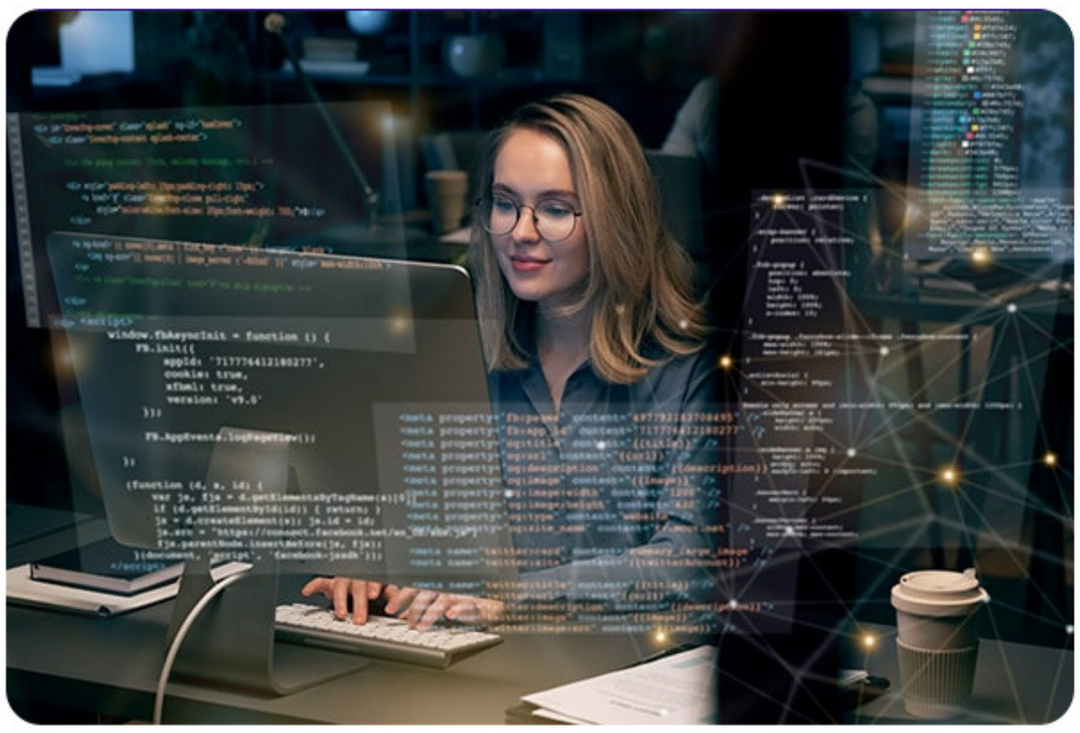
**Input:** s = "cbbd"

**Output:** 2

**Practice**

AlgoTutor

# Take The First Step Towards Fulfilling a Career



## Mastering DSA & System Design


## Advance Data Science & ML


## Full Stack Web Development - MERN


## Mastering DSA & System Design with Full Stack Specialization

**For Admission Enquiry +91 - 72600 58093**

# 16. Minimum Insertion Steps to Make a String Palindrome

Given a string s. In one step you can insert any character at any index of the string.

Return the minimum number of steps to make s palindrome.

A Palindrome String is one that reads the same backward as well as forward.

**Example 1:**

**Input:** s = "zzazz"

**Output:** 0

**Example 2:**

**Input:** s = "mbadm"

**Output:** 2

Practice

# 17. Shortest Common Supersequence

Given two strings str1 and str2, return the shortest string that has both str1 and str2 as **subsequences**. If there are multiple valid strings, return **any** of them.

A string s is a **subsequence** of string t if deleting some number of characters from t (possibly 0) results in the string s.

**Example 1:**

**Input:** str1 = "abac", str2 = "cab"

**Output:** "cabac"

**Example 2:**

**Input:** str1 = "aaaaaaaa", str2 = "aaaaaaaa"

**Output:** "aaaaaaaa"

**Practice**

# 18. Edit Distance

Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

**Example 1:**

**Input:** word1 = "horse", word2 = "ros"

**Output:** 3

**Example 2:**

**Input:** word1 = "intention", word2 = "execution"

**Output:** 5

**Practice**

# 19. Wildcard Matching

AlgoTutor

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

- '?' Matches any single character.
- '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial)

**Example 1:**

**Input:** s = "aa", p = "a"

**Output:** false

**Example 2:**

**Input:** s = "aa", p = "*"

**Output:** true

**Example 3:**

**Input:** s = "cb", p = "?a"

**Output:** false

**Practice**

# 20. Largest Divisible Subset

Given a set of distinct positive integers nums, return the largest subset answer such that every pair (answer[i], answer[j]) of elements in this subset satisfies:

- answer[i] % answer[j] == 0, or
- answer[j] % answer[i] == 0

If there are multiple solutions, return any of them.

**Example 1:**

**Input:** nums = [1,2,3]

**Output:** [1,2]

**Example 2:**

**Input:** nums = [1,2,4,8]

**Output:** [1,2,4,8]

## Practice

AlgoTutor

# WHY ALGOTUTOR

- 100% Placement Assistance
- 1-1 personal mentorship from Indusrty experts
- 200+ Successful Alumni
- 147(Avg.)% Salary Hike
- 100% Success Rate
- 23 LPA (Avg.) CTC
- Learn from scratch
- Career Services

## For Admission Enquiry

+91-7260058093

info@algotutor.io