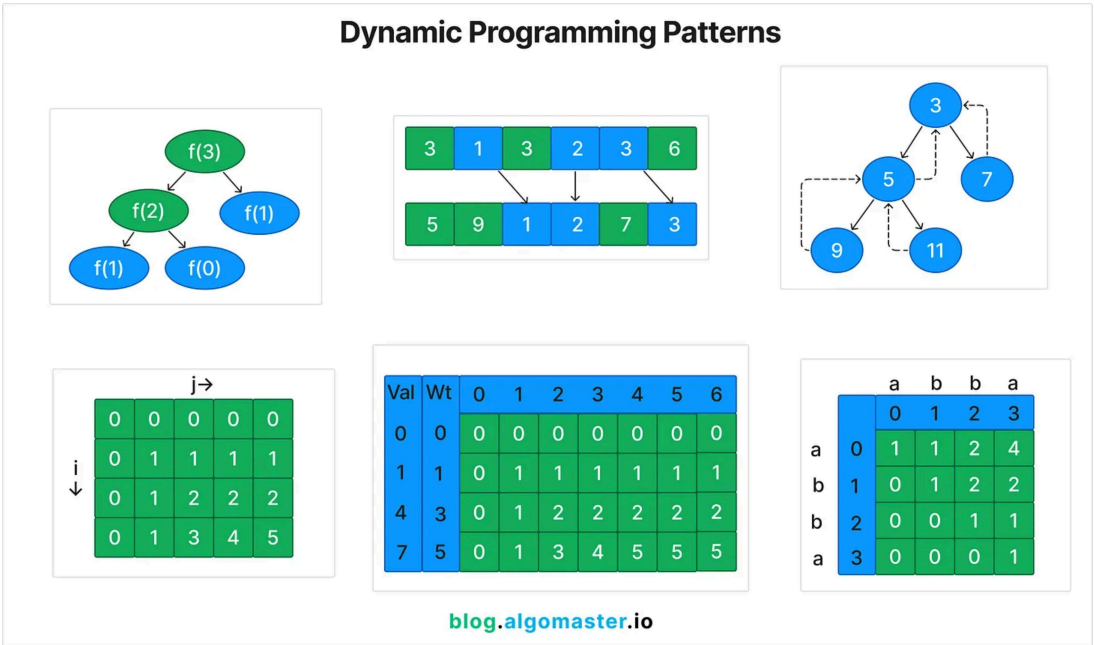
20 Patterns to Master Dynamic Programming

23 - Dynamic Programming Patterns

ASHISH PRATAP SINGH
JUL 28, 2024

Dynamic Programming (DP) is arguably the **most difficult** topic for coding interviews.

But, like any other topic, the fastest way to learn it is by understanding different **patterns** that can help you solve a wide variety of problems.

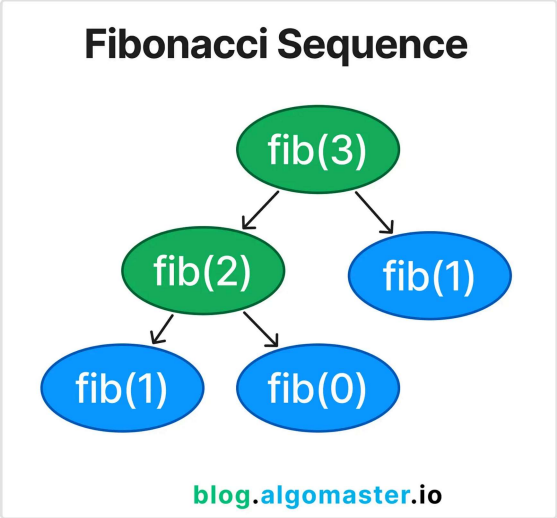


In this article, I'll walk you through **20 patterns** that will make learning DP much easier.

I'll share when to use each pattern and provide links to **LeetCode** problems you can practice to learn them better.

I have listed them from easy to hard and also linked resources to learn each pattern.

1. Fibonacci Sequence



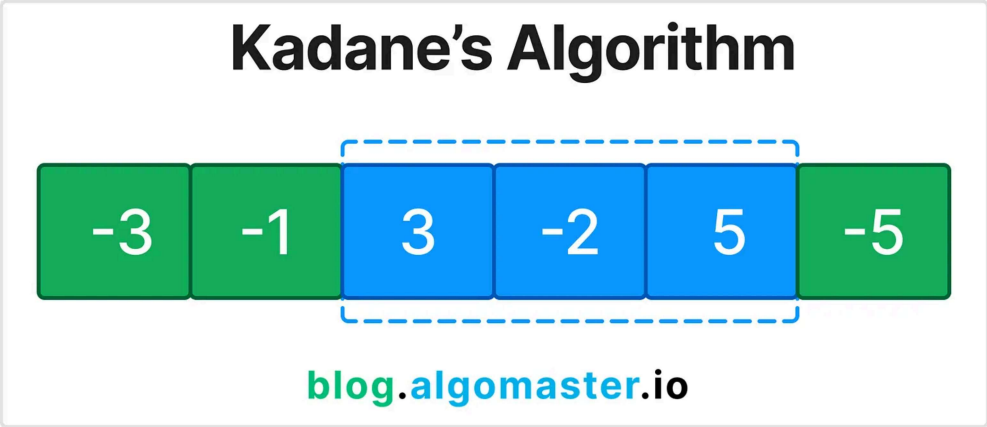
The [Fibonacci sequence] pattern is useful when the solution to a problem depends on the solutions of smaller instances of the same problem.

There is a clear recursive relationship, often resembling the classic Fibonacci sequence `F(n) = F(n-1) + F(n-2)`.

LeetCode Problems:

- LeetCode 70: Climbing Stairs
- LeetCode 509: Fibonacci Number
- LeetCode 746. Min Cost Climbing Stairs

2. Kadane's Algorithm

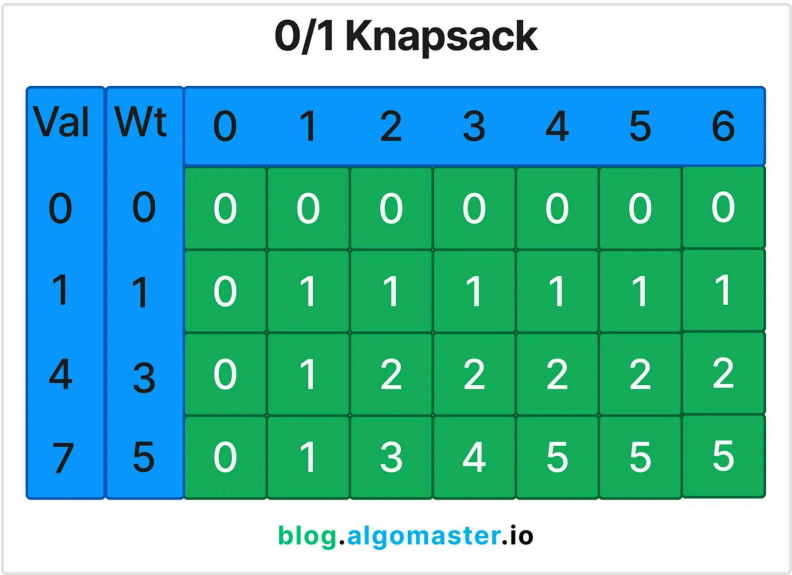


Kadane's Algorithm is primarily used for solving the Maximum Subarray Problem and its variations where the problem asks to optimize a contiguous subarray within a one-dimensional numeric array

LeetCode Problems:

- LeetCode 53: Maximum Subarray
- LeetCode 918: Maximum Sum Circular Subarray
- LeetCode 152: Maximum Product Subarray

3. 0/1 Knapsack



The 0/1 Knapsack pattern is useful when:

1. You have a set of items, each with a weight and a value.
2. You need to select a subset of these items.
3. There's a constraint on the total weight (or some other resource) you can use.
4. You want to maximize (or minimize) the total value of the selected items.
5. Each item can be chosen only once (hence the "0/1" - you either take it or you don't).

LeetCode Problems:

- LeetCode 416: Partition Equal Subset Sum
- LeetCode 494: Target Sum
- LeetCode 1049. Last Stone Weight II

4. Unbounded Knapsack

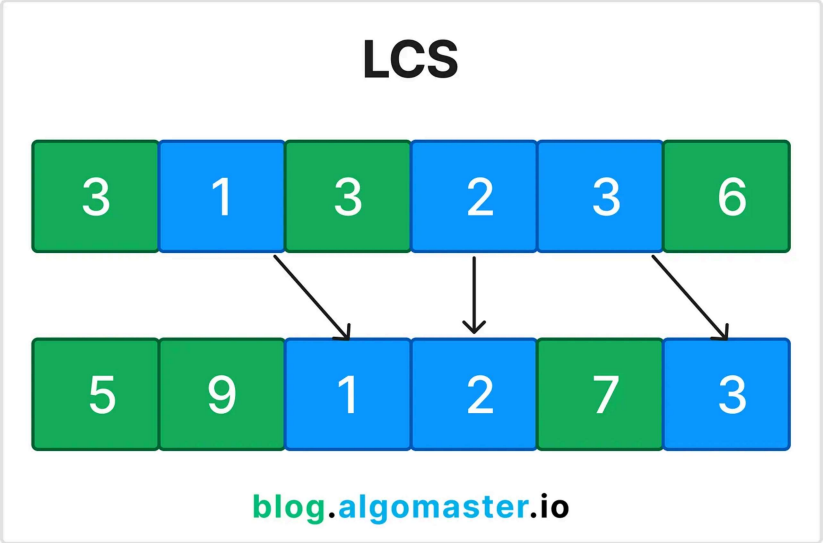
The **Unbounded Knapsack** pattern is useful when:

1. You have a set of items, each with a weight and a value.
2. You need to select items to maximize total value.
3. There's a constraint on the total weight (or some other resource) you can use.
4. You can select each item multiple times (unlike 0/1 Knapsack where each item can be chosen only once).
5. The supply of each item is considered infinite.

LeetCode Problems:

- LeetCode 322: Coin Change
- LeetCode 518: Coin Change 2
- LeetCode 279. Perfect Squares

5. Longest Common Subsequence (LCS)



The [Longest Common Subsequence] pattern is useful when you are given two sequences and need to find a subsequence that appears in the same order in both the given sequences.

LeetCode Problems:

- LeetCode 1143: Longest Common Subsequence
- LeetCode 583: Delete Operation for Two Strings
- LeetCode 1092: Shortest Common Supersequence

6. Longest Increasing Subsequence (LIS)

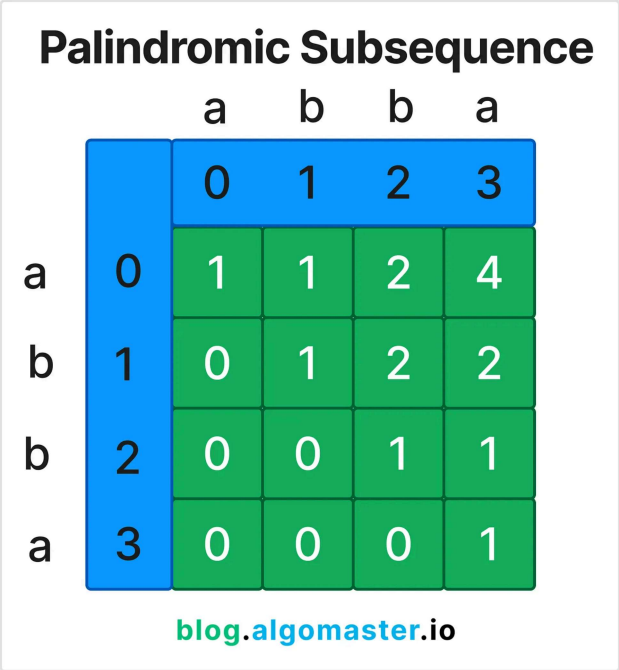
The [Longest Increasing Subsequence] pattern is used to solve problems that involve finding the longest subsequence of elements in a sequence where the elements are in

increasing order.

LeetCode Problems:

- LeetCode 300: Longest Increasing Subsequence
- LeetCode 673: Number of Longest Increasing Subsequence
- leetCode 354: Russian Doll Envelopes

7. Palindromic Subsequence



The Palindromic Subsequence pattern is used when solving problems that involve finding a subsequence within a sequence (usually a string) that reads the same forwards and backwards.

LeetCode Problems:

- LeetCode 516: Longest Palindromic Subsequence
- LeetCode 647: Palindromic Substrings
- LeetCode 1312: Minimum Insertion Steps to Make a String Palindrome

8. Edit Distance

The **Edit Distance** pattern is used to solve problems that involve transforming one sequence (usually a string) into another sequence using a minimum number of operations.

The allowed operations typically include insertion, deletion, and substitution.

LeetCode Problems:

- LeetCode 72: Edit Distance
- LeetCode 583: Delete Operation for Two Strings
- LeetCode 712: Minimum ASCII Delete Sum for Two Strings

9. Subset Sum

The **Subset Sum** pattern is used to solve problems where you need to determine whether a subset of elements from a given set can sum up to a specific target value.

LeetCode Problems:

- LeetCode 416: Partition Equal Subset Sum
- LeetCode 494: Target Sum
- LeetCode 698: Partition to K Equal Sum Subsets

10. String Partition

The **String Partition** pattern is used to solve problems that involve partitioning a string into smaller substrings that satisfy certain conditions.

It's useful when:

- You're working with problems involving strings or sequences.
- The problem requires splitting the string into substrings or subsequences.
- You need to optimize some property over these partitions (e.g., minimize cost, maximize value).
- The solution to the overall problem can be built from solutions to subproblems on smaller substrings.
- There's a need to consider different ways of partitioning the string.

LeetCode Problems:

- LeetCode 139: Word Break
- LeetCode 132. Palindrome Partitioning II
- LeetCode 472: Concatenated Words

11. Catalan Numbers

The **Catalan Number** pattern is used to solve combinatorial problems that can be decomposed into smaller, similar subproblems.

Some of the use-cases of this pattern include:

- Counting the number of **valid parentheses** expressions of a given length
- Counting the number of distinct **binary search trees** that can be formed with `n` nodes.
- Counting the number of ways to triangulate a polygon with `n+2` sides.

LeetCode Problems:

- LeetCode 96: Unique Binary Search Trees
- LeetCode 22: Generate Parentheses

12. Matrix Chain Multiplication

This pattern is used to solve problems that involve determining the optimal order of operations to minimize the cost of performing a series of operations.

It is based on the popular optimization problem: **Matrix Chain Multiplication**.

It's useful when:

1. You're dealing with a sequence of elements that can be combined pairwise.
2. The cost of combining elements depends on the order of combination.
3. You need to find the optimal way to combine the elements.
4. The problem involves minimizing (or maximizing) the cost of operations of combining the elements.

LeetCode Problems:

- LeetCode 1039: Minimum Score Triangulation of Polygon
- LeetCode 312: Burst Balloons
- LeetCode 1000: Minimum Cost to Merge Stones

13. Count Distinct Ways

This **pattern** is useful when:

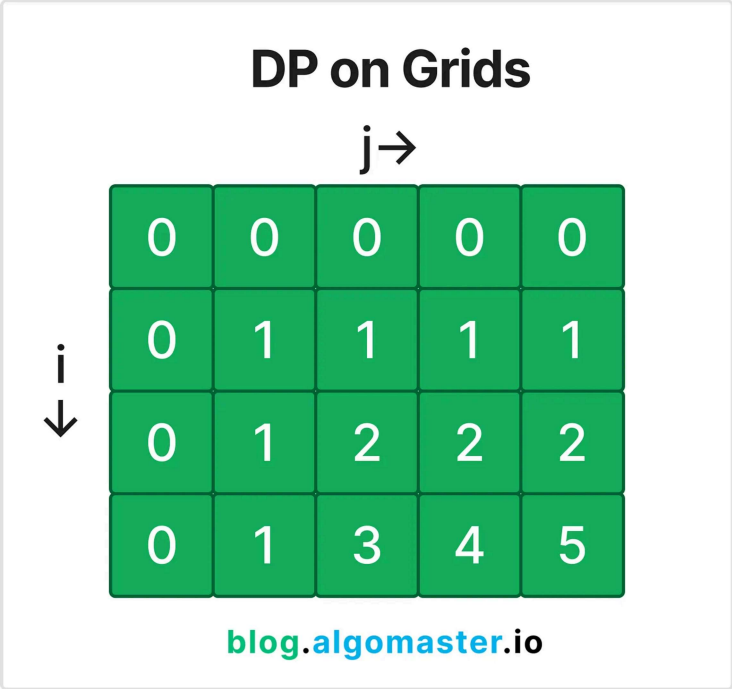
1. You need to **count** the number of different ways to achieve a certain goal or reach a particular state.
2. The problem involves making a series of choices or steps to reach a target.
3. There are multiple valid paths or combinations to reach the solution.
4. The problem can be broken down into smaller subproblems with overlapping solutions.
5. You're dealing with combinatorial problems that ask "**in how many ways**" can something be done.

LeetCode Problems:

- LeetCode 91: Decode Ways

- LeetCode 2266. Count Number of Texts

14. DP on Grids



The DP on Grids pattern is used to solve problems that involve navigating or optimizing paths within a grid (2D array).

For these problems, you need to consider multiple directions of movement (e.g., right, down, diagonal) and solution at each cell depends on the solutions of neighboring cells.

LeetCode Problems:

- LeetCode 62: Unique Paths
- LeetCode 64: Minimum Path Sum
- LeetCode 329. Longest Increasing Path in a Matrix

15. DP on Trees

The DP on Trees pattern is useful when:

1. You're working with tree-structured data represented by nodes and edges.

2. The problem can be broken down into solutions of subproblems that are themselves tree problems.
3. The problem requires making decisions at each node that affect its children or parent.
4. You need to compute values for nodes based on their children or ancestors.

LeetCode Problems:

- LeetCode 337: House Robber III
- LeetCode 124: Binary Tree Maximum Path Sum
- LeetCode 968: Binary Tree Cameras

16. DP on Graphs

The **DP on Graphs** pattern is useful when:

1. You're dealing with problems involving graph structures.
2. The problem requires finding optimal paths, longest paths, cycles, or other optimized properties on graphs.
3. You need to compute values for nodes or edges based on their neighbors or connected components.
4. The problem involves traversing a graph while maintaining some state.

LeetCode Problems:

- LeetCode 787: Cheapest Flights Within K Stops
- LeetCode 1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance

17. Digit DP

The **Digit DP Pattern** is useful when:

1. You're dealing with problems involving counting or summing over a range of numbers.

2. The problem requires considering the digits of numbers individually.
3. You need to find patterns or properties related to the digits of numbers within a range.
4. The range of numbers is large (often up to 10^18 or more), making brute force approaches infeasible.
5. The problem involves constraints on the digits.

LeetCode Problems:

- LeetCode 357: Count Numbers with Unique Digits
- LeetCode 233: Number of Digit One
- LeetCode 902. Numbers At Most N Given Digit Set

18. Bitmasking DP

The **Bitmasking DP pattern** is used to solve problems that involve a large number of **states** or **combinations**, where each state can be efficiently represented using **bits** in an integer.

It’s particularly useful when:

1. You're dealing with problems involving **subsets** or **combinations** of elements.
2. The total number of elements is relatively small (typically <= 20-30).
3. You need to efficiently represent and manipulate sets of elements.
4. The problem involves making decisions for each element (include/exclude) or tracking visited/unvisited states.
5. You want to optimize space usage in DP solutions.

LeetCode Problems:

- LeetCode 1986: Minimum Number of Work Sessions to Finish the Tasks
- LeetCode 2305. Fair Distribution of Cookies
- LeetCode 847: Shortest Path Visiting All Nodes

19. Probability DP

This pattern is useful when:

1. You're dealing with problems involving probability calculations.
2. The probability of a state depends on the probabilities of previous states.
3. You need to calculate the expected value of an outcome.
4. The problem involves random processes or games of chance.

LeetCode Problems:

- LeetCode 688: Knight Probability in Chessboard
- LeetCode 808: Soup Servings
- LeetCode 837. New 21 Game

20. State Machine DP

The **State Machine DP Pattern** is useful when when:

1. The problem can be modeled as a series of states and transitions between these states.
2. There are clear rules for moving from one state to another.
3. The optimal solution depends on making the best sequence of state transitions.
4. The problem involves making decisions that affect future states.
5. There's a finite number of possible states, and each state can be clearly defined.

LeetCode Problems:

- LeetCode 309: Best Time to Buy and Sell Stock with Cooldown
- LeetCode 123: Best Time to Buy and Sell Stock III

Thank you so much for reading.

If you found it valuable, hit a like ❤️ and consider subscribing for more such content every week.

If you have any questions or suggestions, leave a comment.

This post is public so feel free to share it.

Subscribe for free to receive new articles every week.

Type your email... Subscribe

Checkout my Youtube channel for more in-depth content.

Follow me on LinkedIn and X to stay updated.

Checkout my GitHub repositories for free interview preparation resources.

I hope you have a lovely day!

See you soon,

Ashish

235 Likes · 19 Restacks

← Previous Next →

Discussion about this post

Comments Restacks

Write a comment...

Verna Jul 28 Liked by Ashish Pratap Singh

Thanks, love it 😍

LIKE (2) REPLY SHARE

2 replies

selfboot Aug 5

Here is a visual representation of dynamic programming, which can help understand the idea of dynamic programming

https://gallery.selfboot.cn/en/algorithms/dpcoin

LIKE REPLY SHARE

3 more comments...