Dynamic programming (DP) is a powerful technique used to solve problems by breaking them down into smaller subproblems and storing the results of those subproblems to avoid redundant computations. Here are some fundamental concepts and techniques for solving dynamic programming problems on platforms like LeetCode:

**Identify Optimal Substructure**

Dynamic programming problems typically involve finding the optimal solution by breaking down the problem into smaller subproblems. Identify the optimal substructure, meaning that the optimal solution to the overall problem can be constructed from the optimal solutions of its subproblems.

**Overlapping Subproblems**

Many dynamic programming problems have overlapping subproblems, meaning that the same subproblem is solved multiple times. DP techniques involve solving each subproblem only once and storing the results for future use.

**Memoization and Tabulation**

There are two common approaches to implement DP algorithms: memoization and tabulation.

Memoization involves storing the results of solved subproblems in a data structure (like a dictionary or an array) so that they can be reused when needed.

Tabulation involves building a table (usually a 2D array) and filling it iteratively from the bottom up, starting with the base cases and moving towards the solution.

**Top-Down vs. Bottom-Up Approach**

DP problems can be solved using either a top-down (recursive with memoization) or bottom-up (iterative with tabulation) approach. Choose the approach that best fits the problem and leads to efficient solutions.

**State Representation**

Define the state of the problem, which represents the variables needed to compute the solution. The state should be concise and capture all necessary information to solve the problem.

**State Transition**

Determine how to transition between states. Define the recurrence relation or the transition function that relates the current state to its neighboring states.

**Base Cases**

Identify the base cases or smallest subproblems that can be directly solved without further recursion or iteration.

**Optimization Techniques**

Sometimes, dynamic programming solutions can be optimized further by reducing the space complexity or by optimizing the transition function. Look for opportunities to optimize the solution without sacrificing correctness.

**Greedy vs. Dynamic Programming**

Understand the difference between greedy algorithms and dynamic programming. Greedy algorithms make locally optimal choices at each step, while dynamic programming considers all possible choices and optimizes globally.

**Practice, Practice, Practice**

Dynamic programming can be challenging to master, so practice solving a variety of DP problems on platforms like LeetCode. Start with easier problems and gradually move on to more complex ones as you become more comfortable with the concepts and techniques.

By mastering these fundamental concepts and techniques, you'll be better equipped to tackle dynamic programming problems effectively on platforms like LeetCode.

There are several widely used algorithms and techniques for solving dynamic programming problems on LeetCode. Here are some of them:

**Fibonacci Sequence using Memoization or Tabulation**

The Fibonacci sequence is a classic example of a problem that can be solved using dynamic programming. You can use either memoization (top-down) or tabulation (bottom-up) to efficiently compute the Fibonacci numbers.

**Longest Common Subsequence (LCS)**

LCS is a common problem in dynamic programming. Given two sequences, find the longest subsequence present in both of them. This problem can be solved using dynamic programming with memoization or tabulation.

**Longest Increasing Subsequence (LIS)**

LIS is another classic problem that involves finding the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. This problem can also be solved using dynamic programming.

**Knapsack Problem**

The 0/1 Knapsack Problem and the Unbounded Knapsack Problem are both classic optimization problems that can be solved using dynamic programming. These problems involve maximizing the value of items selected while staying within a given weight constraint.

**Coin Change Problem**

Given a set of coin denominations and a target amount, find the minimum number of coins needed to make up that amount. This problem can be solved using dynamic programming with either memoization or tabulation.

**Matrix Chain Multiplication**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. This problem can be solved using dynamic programming to minimize the number of scalar multiplications needed.

**Edit Distance**

Edit Distance is a measure of similarity between two strings, defined as the minimum number of operations required to convert one string into another. This problem can be solved using dynamic programming to find the minimum number of insertions, deletions, and substitutions needed.

**Maximum Subarray**

Given an array of integers, find the contiguous subarray with the largest sum. This problem can be solved using dynamic programming with Kadane's algorithm, which efficiently finds the maximum subarray sum in linear time.

**Partition Equal Subset Sum**

Given a non-empty array containing only positive integers, determine whether it can be partitioned into two subsets with equal sum. This problem can be solved using dynamic programming to find whether a subset sum is possible.

**Unique Paths**

Given a grid of m x n size, find the number of unique paths from the top-left corner to the bottom-right corner, moving only down or right at each step. This problem can be solved using dynamic programming to calculate the number of paths for each cell.

These are some of the widely used algorithms and techniques for solving dynamic programming problems on LeetCode and other similar platforms. Mastering these algorithms will provide a strong foundation for tackling a wide range of dynamic programming problems.