**DYNAMIC PROGRAMMING!**

**What is Dynamic Programming (DP)?**

**Dynamic Programming (DP)** is a method used in mathematics and computer science to solve *complex problems* by breaking them down into *simpler subproblems.* By solving each subproblem *only once* and *storing the results*, it avoids redundant computations, leading to more efficient solutions for a wide range of problems.

**How Does Dynamic Programming (DP) Work?**

* **Identify Subproblems:** *Divide* the main problem into smaller, independent subproblems.
* **Store Solutions:***Solve each subproblem* and *store the solution in a table or array.*
* **Build Up Solutions:** Use the *stored solutions to build up the solution* to the main problem.
* **Avoid Redundancy:** By storing solutions, DP ensures that each subproblem is solved only once, reducing computation time.

**When to Use Dynamic Programming (DP)?**

Dynamic programming is an *optimization technique* used when solving problems that consists of the following characteristics:

**1. Optimal Substructure:**

Optimal substructure means that we *combine the optimal results of* ***subproblems*** to achieve the optimal result of the bigger problem.

**Example:**

*Consider the problem of finding the****minimum cost****path in a weighted graph from a****source****node to a****destination****node. We can break this problem down into smaller subproblems:*

* *Find the****minimum******cost****path from the****source****node to each****intermediate****node.*
* *Find the****minimum******cost****path from each****intermediate****node to the****destination****node.*

*The solution to the larger problem (finding the minimum cost path from the source node to the destination node) can be constructed from the solutions to these smaller subproblems.*

**2. Overlapping Subproblems:**

The same *subproblems are solved repeatedly* in different parts of the problem.

**Example:**

*Consider the problem of computing the****Fibonacci series****. To compute the Fibonacci number at index****n****, we need to compute the Fibonacci numbers at indices****n-1****and****n-2****. This means that the subproblem of computing the Fibonacci number at index****n-1****is used twice in the solution to the larger problem of computing the Fibonacci number at index****n****.*

***KINDA QUESTIONS FROM DIFFERENT APPROACH:***

***1. Optimal Substructure Problems***

*“These problems can be broken down into smaller, independent subproblems, and the solution to the original problem can be derived from the solutions of these subproblems.”*

* ***Longest Common Subsequence (LCS)***
  + *Problem: Find the length of the longest subsequence common to two sequences.*
* ***Knapsack Problem***
  + *Problem: Determine the maximum value that can be obtained by selecting a subset of items with a given weight capacity.*
* ***Rod Cutting Problem***
  + *Problem: Maximize the profit by cutting a rod into pieces with given prices for different lengths.*
* ***Matrix Chain Multiplication***
  + *Problem: Find the most efficient way to multiply a sequence of matrices.*
* ***Longest Increasing Subsequence (LIS)***
  + *Problem: Find the length of the longest subsequence that is strictly increasing.*
* ***Palindrome Partitioning***
  + *Problem: Partition a string into the minimum number of palindromic substrings.*
* ***Coin Change Problem***
  + *Problem: Find the minimum number of coins needed to make a given amount.*
* ***Subset Sum Problem***
  + *Problem: Determine if a subset with a given sum exists within a set of integers.*
* ***House Robber Problem***
  + *Problem: Maximize the amount of money that can be robbed from a list of houses without robbing adjacent houses.*

***2. Overlapping Subproblems***

*“These problems involve solving the same subproblems multiple times. DP optimizes this by storing the results of these subproblems and reusing them.”*

* ***Fibonacci Sequence***
  + *Problem: Find the nth Fibonacci number.*
* ***Edit Distance (Levenshtein Distance)***
  + *Problem: Find the minimum number of operations required to transform one string into another.*
* ***Longest Common Subsequence (LCS)***
  + *Problem: Find the length of the longest subsequence common to two sequences (also overlaps with optimal substructure).*
* ***Rod Cutting Problem***
  + *Problem: Maximize the profit by cutting a rod into pieces with given prices for different lengths (also overlaps with optimal substructure).*
* ***Coin Change Problem***
  + *Problem: Find the minimum number of coins needed to make a given amount (also overlaps with optimal substructure).*
* ***Minimum Path Sum in a Grid***
  + *Problem: Find the path from the top-left corner to the bottom-right corner of a grid that minimizes the sum of the numbers along the path.*
* ***Maximum Subarray Sum (Kadane's Algorithm)***
  + *Problem: Find the contiguous subarray with the maximum sum.*
* ***Palindromic Subsequence***
  + *Problem: Find the length of the longest palindromic subsequence in a given string (also overlaps with optimal substructure).*
* ***Word Break Problem***
  + *Problem: Determine if a string can be segmented into a sequence of valid dictionary words.*
* ***Jump Game***
  + *Problem: Determine if you can reach the last index in an array given a maximum number of steps that can be jumped from each position.*

***3. Problems Exhibiting Both Optimal Substructure and Overlapping Subproblems***

*Most dynamic programming problems actually exhibit both properties. Below are examples that strongly show both:*

* ***Longest Common Subsequence (LCS)***
* ***Knapsack Problem***
* ***Edit Distance***
* ***Rod Cutting Problem***
* ***Coin Change Problem***
* ***Matrix Chain Multiplication***
* ***Longest Increasing Subsequence (LIS)***
* ***Subset Sum Problem***
* ***Palindrome Partitioning***
* ***Minimum Path Sum in a Grid***
* ***House Robber Problem***
* ***Jump Game***

**Approaches of Dynamic Programming (DP)!!**

Dynamic programming can be achieved using two approaches:

**1. Top-Down Approach (Memoization):**

In the top-down approach, also known as **memoization**, we *start with the final solution* and recursively break it down into smaller subproblems. To avoid redundant calculations, we store the results of solved subproblems in a memoization table.

Let’s breakdown Top down approach:

* Starts with the final solution and recursively breaks it down into smaller subproblems.
* Stores the solutions to subproblems in a table to avoid redundant calculations.
* **Suitable** when the number of *subproblems* is **large and many of them are reused.**

**2. Bottom-Up Approach (Tabulation):**

In the bottom-up approach, also known as **tabulation**, we start with the *smallest subproblems* and gradually build up to the final solution. We store the results of solved subproblems in a table to avoid redundant calculations.

Let’s breakdown Bottom-up approach:

* Starts with the smallest subproblems and gradually builds up to the final solution.
* Fills a table with solutions to subproblems in a bottom-up manner.
* Suitable when the number of *subproblems* is **small and the optimal solution can be directly computed from the solutions to smaller subproblems.**

**Dynamic Programming (DP) Algorithm**

Dynamic programming is a algorithmic technique that solves complex problems by breaking them down into smaller subproblems and storing their solutions for future use. It is particularly effective for problems that contains **overlapping subproblems**and **optimal substructure.**

**Common Algorithms that Use Dynamic Programming:**

* **Longest Common Subsequence (LCS):**Finds the longest common subsequence between two strings.
* **Shortest Path in a Graph:** Finds the shortest path between two nodes in a graph.
* **Knapsack Problem:**Determines the maximum value of items that can be placed in a knapsack with a given capacity.
* **Matrix Chain Multiplication:**Optimizes the order of matrix multiplication to minimize the number of operations.
* **Fibonacci Sequence:**Calculates the nth Fibonacci number.

<https://www.geeksforgeeks.org/dynamic-programming/>