Dynamic Programming

Dynamic programming (DP) is a powerful algorithmic technique used to solve optimization problems by breaking them down into simpler subproblems and efficiently solving those subproblems using a combination of recursion and memoization or tabulation. Here's a note detailing the approach to dynamic programming algorithms:

1. Problem Identification:

Recognize problems that exhibit optimal substructure and overlapping subproblems. Optimal substructure implies that the optimal solution to the problem can be constructed from optimal solutions to its subproblems. Overlapping subproblems imply that the same subproblems are encountered multiple times.

2. Define Subproblems:

Decompose the original problem into smaller, simpler subproblems. Each subproblem should represent a part of the original problem and be of the same form, but with reduced input size.

3. Formulate Recurrence Relation:

Express the solution to each subproblem in terms of solutions to its smaller subproblems. This is often done using a recursive formula, which defines how solutions to larger problems can be computed from solutions to smaller problems.

4. Choose an Approach:

There are two common approaches to solving dynamic programming problems:

Memoization (Top-Down): Start solving the original problem and recursively solve smaller subproblems, storing the solutions in a table or cache (memoization) to avoid redundant computations.

Tabulation (Bottom-Up): Solve the smallest subproblems first and store their solutions in a table. Then, use these solutions to solve larger subproblems iteratively until reaching the original problem.

5. Implement the Algorithm:

Depending on the chosen approach (memoization or tabulation), implement the dynamic programming algorithm using appropriate data structures (arrays, matrices, hash tables) to store intermediate solutions and avoid redundant computations.

6. Optimize Space and Time Complexity:

Analyse the time and space complexity of the algorithm and look for opportunities to optimize it further. Techniques such as space optimization (reducing memory usage) or time optimization (eliminating unnecessary computations) can be applied.

7. Build Solution to Original Problem:

Once all subproblems have been solved, construct the solution to the original problem using the solutions stored in the table or cache.

What is dynamic programming advantage?

Efficiency: Reduces time complexity by avoiding redundant calculations.

Optimization: Ideal for finding the best solution among feasible options.

Memoization: Stores results of costly function calls to save time on repeated inputs.

Simplicity: Often provides a more intuitive solution once the approach is understood.

Generalization: Solutions can be adapted to solve a range of similar problems.

Structured Approach: Offers a systematic method for tackling complex problems.

Overlapping Subproblems: Solves each subproblem once, preventing repetitive computations.

Bottom-up Computation: Starts with the smallest subproblems, ensuring all required data is available.

Space Efficiency: Uses extra memory for significant time savings.

Versatility: Applicable to a broad range of algorithmic and real-world problems.

What are dynamic programming limitations?

Memory Usage: Can require significant memory to store solutions of all subproblems.

Optimal Substructure: Not all problems have the property where the optimal solution can be constructed from optimal solutions of its subproblems.

Initialization Overhead: Setting up tables or matrices can add extra initialization time.

Complexity: Implementing dynamic programming solutions can be more complex than simpler recursive solutions.

Not Always Efficient: For some problems, greedy algorithms can be more efficient.

Overhead of Recursion: Recursive dynamic programming can lead to stack overflow for deep recursions