DAILY DSA | DAY-24 | Dynamic programming –Brief|
-GOPALKRISHNA A

Dynamic programming: Technique for improving the runtime performance of problems by saving previously calculated values that are repeated frequently, eliminating the need for any re-computation.

In general, DP uses **caching to ensure that each subproblem is solved at most once**.

Note: As a rule of thumb, any problem that can be solved with DP can also be solved recursively/Divide and conquer approach, but DP is most helpful when the problem has **overlapping subproblems.**

When to use dynamic programming (DP)
The main two properties of DP are:

Optimal substructure: We say that a problem has an optimal substructure if we can break the problem into sub/recursive

For example: We want to make a big Lego tower using Lego bricks. We can do that by sticking the small Lego towers on top of each other. Each logo tower is a substructure, and one bigger tower is the bigger structure.

In the Optimal substructure, we combine the optimal results of subproblems to achieve the optimal result of the bigger problem.

Overlapping subproblems: We say that a problem has overlapping subproblems when the problem is broken down into recursive subproblems — some subproblems are solved more than once.
DP can be used when solutions to the same/repeated subproblems are needed again and again.

Example: Consider a function Fibonacci(n) that outputs the n-th Fibonacci number (F_n). In the series calculations happen more than once - in total, we actually compute Fibonacci (1) three times in the tree!
In DP, we use caching to optimize this solution by ensuring that we never have to repeat the calculation

How does dynamic programming work?

1. Break down the broader or complex problem into several smaller subproblems
2. It computes a solution to each subproblem
3. After calculating the result, it remembers the solution to each sub-problem (Memorization)

4. Reutilize stored result if given subproblem recur
5. Integrate solutions of subproblems to formulate a broader problem's solution

In this solution-building approach, we are utilizing memory to store the results. Hence, the **space complexity will be increased**. But, owing to the same utilization of space, the **time complexity will be decreased** significantly.

Approaches of Dynamic Programming:

Top-down approach: Top-down approach follows the memorization technique. It consists of two distinct events: recursion and caching.
Memoization = Recursion + Caching

Recursion represents the process of computation by calling functions repeatedly, whereas 'caching' represents the storing of intermediate results.

Advantages:

- Easy to understand and implement
- Solves the subproblem only if the solution is not memorized
- Debugging is easier

Disadvantages:

- Uses recursion, which takes up more memory space in the call stack, degrading the overall performance
- Possibility of a stack overflow error

Bottom-up approach: This approach uses the tabulation technique to implement the dynamic programming solution. It addresses the same problem as before but without recursion. The recursion is replaced with iteration. Hence there is no stack overflow error or overhead of recursive procedures.

Top down approach	Bottom up approach
Uses memorization technique	Uses tabulation technique
Recursive nature	Iterative nature
Approach uses decomposition to formulate a solution	Approach uses composition to develop a solution
A lookup table is maintained and checked before computation of any subproblem	The solution is built from a bottom-most case using iteration