Dynamic Programming

Dynamic Programming's main notion is to prevent repeating labor by remembering partial results, and this concept is used in a variety of real-world situations.

A very good example

If we have fish tank and we are assign to count the fishes

Now we count the no. of fishes as 10

So we answer 10.

Now if someone puts one more fish in the tank

We quickly answer 11. Without counting again since we remember the last answer "10"

Dynamic programming is the same concept remembering the answer to the past question

Core Idea : A problem is solved by identifying a set of subproblems and addressing them one by one, smallest to largest, until all of them are solved. (bottom up approach)

General problems where we apply DP

This method of solving generally applied in the problems that can be restructured(greedy substructure property) after taking the first optimal step but we don't know the first optimal step that would lead to optimal solution(there is no way to find).

Naïve method would suggest to try with every possible first step at each step(restructure). This would result in exponential algorithm. Applying DP methods helps us to reduce to time to polynomial inmost cases.

General Strategy to solve the DP problems

1. We find a way to divide the problem into subproblem
2. We solve these subproblems in order such that we have all the information to solve current problem using the solutions of past subproblems

Look at the Shortest path in DAG problem at this point then come-back here and continue

Most dynamic programming problems can be abstracted away as implicit dags. Where nodes of the DP are the subproblems

1. We find a way to represent sub-problem of question as a DAG.
2. We perform a topological sort.
3. We start from source, solve and memorize the solution in linearized/topological order of DAG.

Doing this helps in finding the order we should solve the subproblems also helps in finding time complexity.

General method to get Time complexity

The time it takes to run a dynamic programming algorithm can be easily obtained from the dag of subproblems: in many cases, it's simply the total number of edges in the dag. All we're doing is visiting nodes in linearized order, inspecting each node's in-edges, and, in most cases, performing a constant amount of work per edge. Each edge of the dag has been checked at least once by the end.