Dynamic Programming

Dynamic Programming Defined
Dynamic programming amounts to breaking down an optimization problem into simpler sub-problems, and storing the solution to each sub-problem so that each sub-problem is only solved once. Dynamic Programming is mainly used when solutions of same sub-problems are needed again and again(overlapping sub-problem).

The term **Programming** is not related to coding but it is from literature, and means **filling tables**

Sub-problems are smaller versions of the original problem. In fact, sub-problems often look like a reworded version of the original problem. If formulated correctly, sub-problems build on each other in order to obtain the solution to the original problem.

DP is a useful technique for optimization problems, those problems that seek the maximum or minimum solution given certain constraints, because it looks through all possible sub-problems and never recomputes the solution to any sub-problem. This guarantees correctness and efficiency, which we cannot say of most techniques used to solve or approximate algorithms.

```
General explanation:
Dynamic Programming = Recursion + memoization (caching subproblem solutions)
```

Two main properties of a problem that suggest that the given problem can be solved using Dynamic programming:

1. Overlapping Subproblems
2. Optimal Substructure

There are two main techniques of storing solution of sub-problems which are *not mutually exclusive*:

- Tabulation (Bottom Up)
- Memoization (Top down)

Bottom Up: If we start our transition from our base state i.e dp[0] and follow our state transition relation to reach our destination state dp[n], we call it Bottom Up approach. Reverse is true for Top down approach.

Example:
fibonacci(5) - with general recursive code

```
                  F(5)
                /      \
               /        \
              /          \
           F(4)          F(3)
        /       \        /   \
      F(3)     F(2)     F(2)  F(1)
     /   \     /  \     /   \
   F(2) F(1) F(1) F(0) F(1) F(0)
   /  \
 F(1) F(0)
```

The tree above represents each computation that must be made in order to find the Fibonacci value for n = 5. Notice how the sub-problem for n = 2 is solved thrice. For a relatively small example (n = 5), that’s a lot of repeated , and wasted, computation!

What if, instead of calculating the Fibonacci value for n = 2 three times, we created an algorithm that calculates it once, stores its value, and accesses the stored Fibonacci value for every subsequent occurrence of n = 2? That’s exactly what memoization does.

Popular Dynamic Programming questions

- Fibonacci Sequence ✅
- 0-1 Knapsack problem ✅
- Longest common subsequence problem ✅
- Egg Dropping Puzzle
- String Edit Distance ✅
- Cutting a Rod
- Longest Increasing Subsequence

How to solve a Dynamic Programming Problem

Step 1: Identify the sub-problem in words.

```
If you can identify a sub-problem that builds upon previous sub-problems to solve the problem at hand, then y
```

Step 2: Write out the sub-problem as a recurring mathematical decision.

```
If it is difficult to encode your sub-problem from Step 1 in math, then it may be the wrong sub-problem!

There are two questions that I ask myself every time I try to find a recurrence:
-  What decision do I make at every step?
-  If my algorithm is at step i, what information would it need to decide what to do in step i+1?
(And sometimes: If my algorithm is at step i, what information did it need to decide what to do in step i-1?)
```

Step 3: Solve the original problem using Steps 1 and 2.

```
Since the sub-problem we found in Step 1 is the maximum value schedule for punchcards i through n such that t
we can write out the solution to the original problem as the maximum value schedule for punchcards 1 through
Since Steps 1 and 2 go hand in hand, the original problem can also be written as OPT(1)
```

Step 4: Determine the dimensions of the memoization array and the direction in which it should be filled.
Step 5: Code it!

Performance analysis of dynamic algorithm

Generally, a dynamic program's runtime is composed of the following features:

```
Pre-processing
How many times the for loop runs
How much time it takes the recurrence to run in one for loop iteration
Post-processing
```

Overall, runtime takes the following form:

```
Pre-processing + Loop * Recurrence + Post-processing
```

online articles:

- Topcode Notes on DP
- Medium article on DP
- Top 20 DP questions