**Dafuq is dynamic programming?**

Basically the divide and conquer method, except the results of solved subproblems are stored and used to check if other subproblems have already been resolved instead of computing duplicate subproblems.

**Optimal solutions:**

When solving a problem, there exist multiple solutions. Each solution is represented by some value (basically, the output). The most optimal solutions are the ones the lead to either the greatest or lowest value possible. Whether you should aim for min or max is contextual. For example, if you need to find a solution that gives you a score, you generally want the highest score possible, while if the solution gives some measurement of speed, say for a car, you generally want the lowest time possible.

To make an algorithm that finds the most optimal solution, the following steps serve as great guidelines:

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution, typically in a bottom-up fashion.
4. Construct an optimal solution from computed information.

The first 3 steps serve as the core for algorithms designed to find optimal solutions, but on their own can only help find the value of it. If you include step 4 you can also make the algorithm actually define the solution, however this may require that you maintain extra information during step 3.

**Rod cutting example problem.**

You have a company that sells rods. They charge based on length. Longer rods = greater price per inch. Find the optimal solution.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| length i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Price Pi | 1 | 5 | 8 | 9 | 10 | 17 | 17 | 20 | 24 | 30 |

As can be seen in the table above, the price increases erratically with the length of the rod. Granted, even if it did not, that would just mean the optimal solution would be different, despite the problem technically being the same. It should be noted that a rod of length I can be cut up in different ways.

An algorithm looking for the most optimal price of a rod will take the rods length as input, and then recursively divide the rod. For each length given, the algorithm will divide it in every way possible (divide into subproblems), and compare the different ratios, selecting the best one available. For example, say you have a rod of length 4, the algorithm will store the value of the rod, then divide it with the ratios 1:3, 2:2, and 3:1. It will then sum the returned value of the divisions, 1+8, 5+5, and 8+1. These values are then compared with the 4-rods own value, to find the optimal solution. In this case, dividing the rod 2:2 would be best, as that gives a value of 10 instead of 9. It should also be noted that for each division, the algorithm is recursively running on both lengths, so for 1:3 the algorithm will run with 1 first and then with 3 second, while with 2:2 it will then run with 2 twice. You’re also supposed to do some wizardry bullshit where you just tell the program the answer to subproblems that have already been solved instead of having it solve the exact same scenario like 500 times. Hashmaps could be your friend and savior for this. In fact, doing this is referred to as top-down with memoization by the book. The bottom-up version would have the algorithms sort the division by size (smallest first). The trick with this one is that you build up the memory with the smallest, most basic subproblems. When you then encounter a big subproblem, you will already have the solution to all the sub-subproblems, making it quick to solve the bigger subproblem. This is usually the preferred way, as it has steady progression or some shit.

**Do exercise 2 on moodle on lecture 9**