Basic Dynamic Programming

What is Dynamic Programming?

Name isn’t helpful.
Programming doesn’t actually refer to computer programming but more so planning.

DP is a method of solving problems where you break the problem into smaller subproblems such that you can use the results of the earlier subproblems to more efficiently compute the result of the full problem.

The main thing is you calculate values in an order such that you can use the previous computed ones to compute new ones efficiently.

Example problem

https://dmoj.ca/problem/ccc00s4

Another example

https://dmoj.ca/problem/ioi94p1

More generally

In DP problems you always have a “state” and a “transition”.

State is whatever data you actually store/compute.

Transition is the function that says how to compute the values for each of the states based on previous states.

ccc00s4

State: dp[i] is min # of strokes to get to position i

Transition: dp[i] = min(dp[i-arr[j]]) + 1

ioi94p1

State: dp[i][j] is max sum of a path going from root to row i, position j

Transition: dp[i][j] =

arr[i][j] + dp[i-1][j]	if j == 0
arr[i][j] + dp[i-1][j-1]	if j == i
arr[i][j] + max(dp[i-1][j-1], dp[i-1][j])	if 0 < j < i

Compute recursively

You can also compute dp values recursively. In this case, you make a recursive function that calculates the value of a given state. We also store the result of the function, and any time in the future we ask it to compute the same state, it just does a lookup in an array/map. This way, we don’t compute the value multiple times. Most DP problems in general can be done recursively; often both iterative and recursive approaches work.

Recursion can be enforced if not all states need to have their values computed and it’s not easy to figure out which ones do. Problems like this are not common.

Iterative can be enforced if the problem has a tight time limit (iterative is generally faster, in some cases much faster), or a low enough memory limit that you can only optimize memory with iterative (see next slide).

Memory optimizations

Sometimes you don’t need to store all previous states.

Example:
https://dmoj.ca/problem/dpd

Interval DP

Have a DP state that corresponds to an interval. E.g dp[i][j] is some function of the interval [i, j).

Example:
https://dmoj.ca/problem/dpn

Useful to iterate through intervals in increasing order of length.

DP further comments

Hardest part of a DP problem is often coming up with the state.

You'll definitely need to practice solving a few problems to see what kinds of states there are that are useful.

Problems

Problems that appeared in earlier slides:
https://dmoj.ca/problem/ccc00s4
https://dmoj.ca/problem/ioi94p1
https://dmoj.ca/problem/dpd
https://dmoj.ca/problem/dpn

DP on prefix:
https://dmoj.ca/problem/dpb
https://dmoj.ca/problem/ccc02s4
https://dmoj.ca/problem/dpc
https://dmoj.ca/problem/ccc07j5

2D state:
https://dmoj.ca/problem/ccc15j5
https://dmoj.ca/problem/dpe (not quite the same as dpd, which we did on an earlier slide. Constraints are different)

Enforces iterative:
https://dmoj.ca/problem/ioi07p4 (need to optimize memory)

Enforces recursive:
https://dmoj.ca/problem/ccc18s4 (proving exact complexity is hard, not necessary for solving)

Interval DP:
https://dmoj.ca/problem/dpl
https://dmoj.ca/problem/ccc16s4