R5 Dynamic Programming 1

4.4 https://github.com/GUMI-21/MIT6.006_note
Dynamic Programming often applies to optimization problems, where you are maximizing or minimizing a single scalar value, or counting problems, where you have to count all possibilities.

Solving a Problem Rcursively Framework(SRT BOT)

1.*Subproblem* definition subproblem x $\in$ X

- Describe the meaning of a subproblem in words, in terms of parameters
- Often subsets of input: prefixes, suffixes, contiguous subsequences
- Ofen record partial state: add subproblems by incrementing some auxiliary variables
  2.*Relate* subproblem solutions recursively x(i) = f(x(j),..) for one or more j < i
  3.*Topological order* to argue relation is acyclic and subproblems form a DAG
  4.*Base cases*
- State solutions for all (reachable) independent subproblems where relation doesn't apply/work
  5.*Original problem*
- Show how to compute solution to original problem from solutions to subproblems
- Possibly use parent pointers to recover actual solution, not just objective function
  6.*Time* analysis

- $\sum_{x \in X} \text{work}(x)$, or if $\text{work}(x) = O(W)$ for all $x \in X$, then $|X| \cdot O(W)$

- work(x) measures nonrecursive work in relation; treat recursions as taking O(1) time

Implementation

Once subproblems are chosen and a DAG of dependencies is found, there are two primary methods for solving the problem, which are functionally equivalent but are implemented differently.

- Top down
  approach evaluates the recursion starting from roots (vertices incident to no incoming edges). At the end of each recursive call the calculated solution to a subproblem is recorded into a memo, while at the start of each recursive call, the memo is checked to see if that subproblem has already been solved.

- Bottom up
  approach calculates each subproblem according to a topological sort order of the DAG of subproblem dependencies, also recording each subproblem solution in a memo so it can be used to solve later subproblems. Usually subproblems are constructed so that a topological sort order is obvious, especially when subproblems only depend on subproblems having smaller parameters, so performing a DFS to find this ordering is usually unnecessary.