Dynamic Programming I

Recursion:
To solve a problem for one input,
solve on other inputs
and combine results.

Benefits: often easy to find
Problem: usually exponential time.

How can we make it faster?

Memoization: whenever you compute
\( f(x) \) in some recursive call,
store the answer. Next time
you call \( f(x) \) just return it.

Memoization: time = \( \text{number of inputs} \times \text{time per call} \)
Recursion: time = \( \text{number of paths from input to base case} \)
Can think of inputs as a DAG. 
A → B if B’s recursion uses A.

Fibonacci:

Recursion time = # paths = $F_n \approx 1.6^n$
Memorization time = (# inputs) * (time per input) 
= n * n = $n^2$ bit operations

Bottom-up DP: fill from left to right, looking at each node.

Can either **pull** or **push** along edges.

**Pull**: compute a node’s value when you visit it, based on previous values

**Push**: when you visit a computed node, update future node values that use it.

(In Fibonacci: add your value to theirs)

Generally equivalent, sometimes only have easy access to out- or in-edges.
Interval Scheduling:

Want to compute $\text{Sched}(I)$
where $I =$ set of $(s_i, f_i, w_i)$ pairs
maximize $\sum_{i \in S} w_i$
for $S \subseteq I$ non-overlapping.

Naive recursion:
$\text{Sched}(I)$
Let $i =$ first elt of $I$
Return min of
not chosen: $\text{Sched}(I \setminus i)$,
chosen: $\text{Sched}(I \setminus i \setminus i$ or anything conflicting with $i$) + $w_i$.

Problem: $2^n$ possible inputs.
Solution: If I sorted by $f_i$, then
only $n+1$ inputs ever happen:
(suffix of I sorted by $s_i$)

$\Rightarrow$ memoized time is $n \cdot (\text{time per input})$
$\quad n^2$ naively
$\quad n$ more carefully
$\quad \text{Sched(index in I, f or last chosen})$
DAG: sort by $f_i$.

$i \rightarrow j$ if $f_i \leq S_j$. or weight $w_i$.

S -> everything weight 0

everything -> + weight $w_j$

Answer = max weight $s \rightarrow +$ path.

Path $s \rightarrow i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_k \rightarrow +$

- Weight = $0 + w_{i_1} + w_{i_2} + \ldots + w_{i_k}$
  = total value of scheduling then

- $f_{i_1} \leq S_{i_1} \leq f_{i_2} \leq S_{i_2} \leq \ldots$

  => is valid schedule.

Many DP problems have this form:

Convert to a DAG

Find max-weight (or min-weight)

$s \rightarrow +$ path.

Time = # edges in DAG.
Stamps
values \( S_1, \ldots, S_n \) w/ finest stamps

What is the DAG?

nodes = value
\( x \rightarrow x + S_i \) \( \forall i, x \) of weight 1

Answer = shortest \( 0 \rightarrow C \) path.
Time = \# edges = \( C \cdot N \).