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 = \( \# \text{ inputs} \times \text{time per call} \)

Recursion: time = \# paths from input to base case
Can think of inputs as a DAG.  
A \rightarrow B  if  B's recursion uses A.

Fibonacci:

\[ \text{Recursion time} = \# \text{paths} = F_n \sim 1.6^n \]
\[ \text{Memoization time} = (\# \text{inputs}) \cdot (\text{time per input}) = n \cdot n = n^2 \text{ bit operations} \]

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

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 nodes 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: $\text{Sched}(I \setminus \{i\} \text{ or anything conflicting with } i \text{ \& } i)$ $+ w_i$.

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 $n \cdot (\text{time per input})$

$n^2$ naively
$n_2$ more carefully
$\text{Sched}(\text{index in } I, f \text{ or last chosen})$
Let $G$ be a directed acyclic graph (DAG) with vertices labeled by $f_i$. Sort the vertices by $f_i$, then:

1. If $f_i = f_j$, sort by $w_i$.
2. If $f_i < f_j$, sort by $f_j$.
3. If $f_i > f_j$, sort by $w_i$.

$S \rightarrow$ everything weight 0
everything $\rightarrow$+ weight $w_i$

Answer is $\max$ 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

If $f_{i_1} \leq f_{i_2} \leq f_{i_3} \leq \ldots$

$\Rightarrow$ 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 \( S_1, \ldots, S_n \)
want collection of value \( C \)
with fewest stamps

What is the DAG?

Nodes = value
\[ x \rightarrow x + S_i \quad \forall i, x \text{ of weight } 1 \]

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