Dynamic Programming

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.

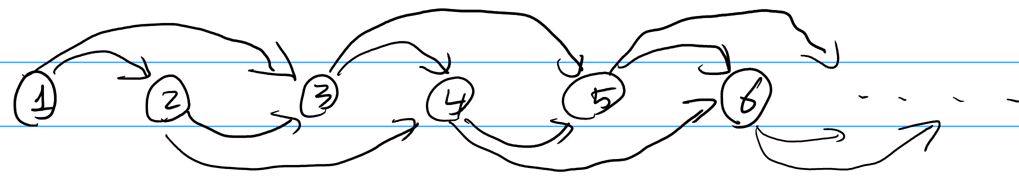
Memoization: time = (# possible inputs) · (time per call)

Recursion: time = # paths from input to base case

Can think of inputs as a DAG.

$A \to B$ if B's recursion uses A.

Fibonacci:



Recursion time = # paths = $F_n \approx 1.6^n$

Memoization time = (# inputs)$\cdot$(time per input)

$= n \cdot n = n^2$ bit operations

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

Can either <u>Pull</u> or <u>Push</u> 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 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.

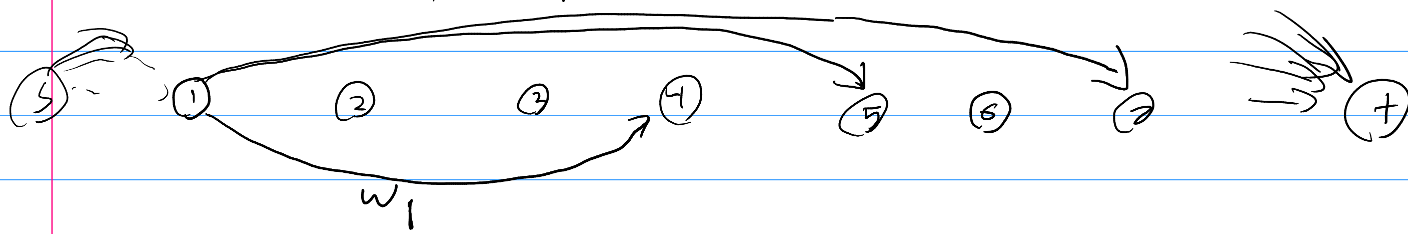
Naive recursion:

Sched (I)
Let i = first elt of I
Return min of
not chosen: Sched $(I \setminus i)$,
chosen: Sched $(I \setminus \{ i$ 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 $n \cdot$ (time per input)
n^2 naively
n more carefully.
Sched (index in I, f of last chosen)

DAG: sort by f_i.



$i \to j$ if $f_i \le s_j$. of weight w_i.

$s \to$ everything weight 0

everything $\to t$ weight w_j

Answer = max weight $s \to t$ path.

Path $s \to i_1 \to i_2 \to \dots \to i_k \to t$

- weight $= 0 + w_{i_1} + w_{i_2} + \dots + w_{i_k}$

  = total value of scheduling them.

- $f_{i_1} \le s_{i_2} < f_{i_2} \le s_{i_3} < \dots$

  $\Rightarrow$ is valid schedule.

Many DP problems have this form:

Convert to a DAG

Find max-weight (or min-weight) s–t path.

Time = # edges in DAG.

Stamps

values $s_1, \ldots, s_n$

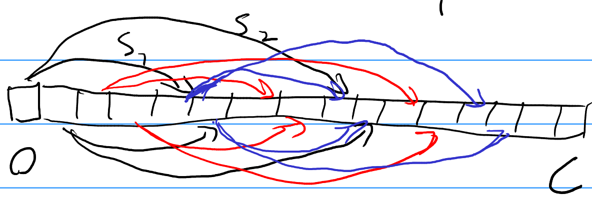
want collection of value C.

w/ fewest stamps

What is the DAG?

nodes = value

$x \to x + s_i$ $\forall i, x,$ of weight 1



Answer = shortest $0 \to C$ path.

Time = # edges = $C \cdot n$.