# 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?

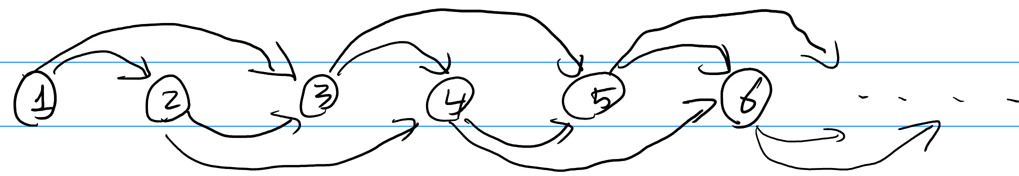
Memoization: whenever you compute $f(x)$ in some recursive call, store the answer. Next time you call $f(x)$, just return it.

Memoization: time = (# 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) · (time per input)
$= n \cdot n = n^2$ bit operations

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

Can either Pull or Push along edges.

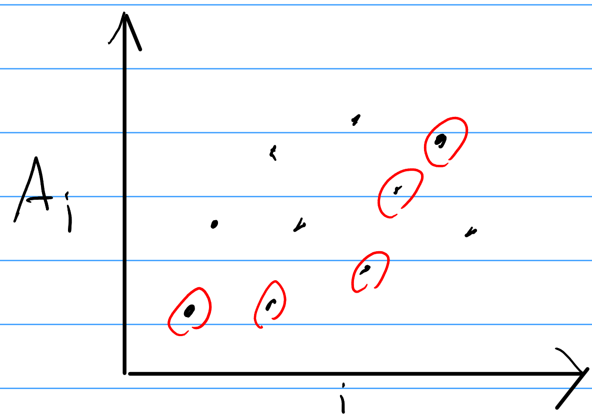
Pull: compute a node's value when you visit it, based on previous values

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.

Longest increasing Subsequence:




Given $n$ numbers $A_1, \ldots, A_n$

find increasing subsequence:

$s_1, \ldots, s_k \in [n]$

$s_1 < s_2 < \ldots < s_k$ (subsequence)

$A_{s_1} < A_{s_2} < \ldots < A_{s_k}$ (increasing)

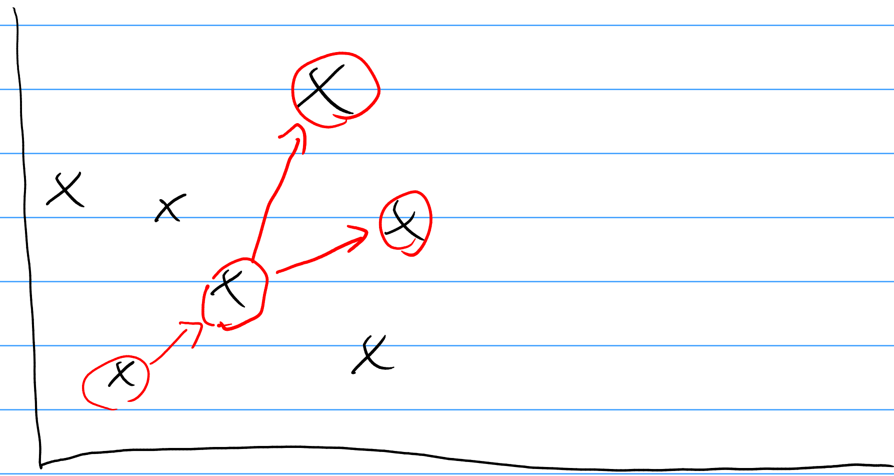
of maximum length $k$.

---

Will convert to longest path on DAG

$O(n^2)$ bottom-up approach:

Suppose you build your solution $S_1, \ldots, S_k$ from left to right



walking a path: $(S_1, A_{S_1}) \to (S_2, A_{S_2}) \to \ldots$

on the DAG $(i, A_i) \to (j, A_j) \iff i < j \ \& \ A_i < A_j$

start at $(-\infty, \infty)$ end at $(\infty, \infty)$

Answer = longest Path on DAG $(-1)$

because:

Any Path is an increasing sequence

any IS is a Path

$\Rightarrow$ longest Path = longest IS

## Memoized View

$f_A(i) :=$ LIS ending at $A_i$

$$= \max_{j<i,\ A_j<A_i} f_A(j)+1$$

$$\text{Answer} = \max_{i\in[n]} f_A(i)$$

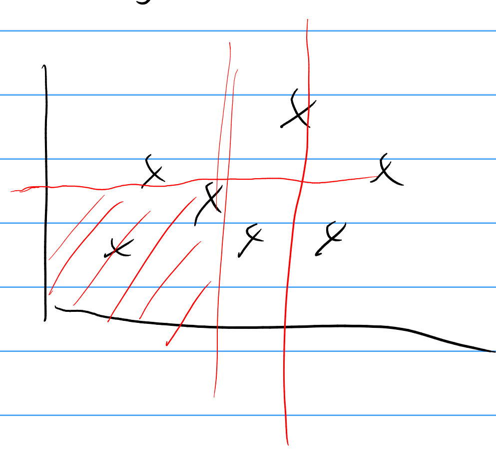
OR (for poly time)

if $f(A) =$ LIS of $A$.

- let $i = \operatorname{argmax} A_i$
- LIS either contains $i$ or not

$$f(A) = \max\begin{pmatrix} f(A \setminus A_i), \\ f(A_1, \ldots, A_{i-1}) + 1 \end{pmatrix}$$

How many possible inputs?



box to LL

$\Rightarrow n^2$ options

$+ n$ time per option

$= n^3$

# $O(n \log n)$ version

Given two possible starts, on $A_1, \ldots, A_m$:

$s_1, \ldots, s_i$

$s'_1, \ldots, s'_j$

When is one clearly superior?
When are they equivalent?

Answer: only will care about length & last value

$$f_m(k) := \text{minimal last value of length-}k\text{ subsequence on } 1, \ldots, m$$

$$f_{m+1}(k) = \min\big(f_m(k), \quad \Leftarrow \text{don't use } A_{m+1}$$
$$A_{m+1} \text{ if } f_m(k-1) < A_{m+1}\big)$$

$f_m$: $[z_1, z_2, \ldots, z_t, z_{t+1}, \ldots, z_\ell]$

if $A_{m+1} \in (z_t, z_{t+1})$:
$f_{m+1}$: $[z_1, z_2, \ldots, z_t, A_{m+1}, z_{t+2}, \ldots, z_\ell]$

$\Rightarrow$ update is $O(\log n)$ binary search
$\Rightarrow$ $O(n \log n)$ time

# 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 $s_i$)

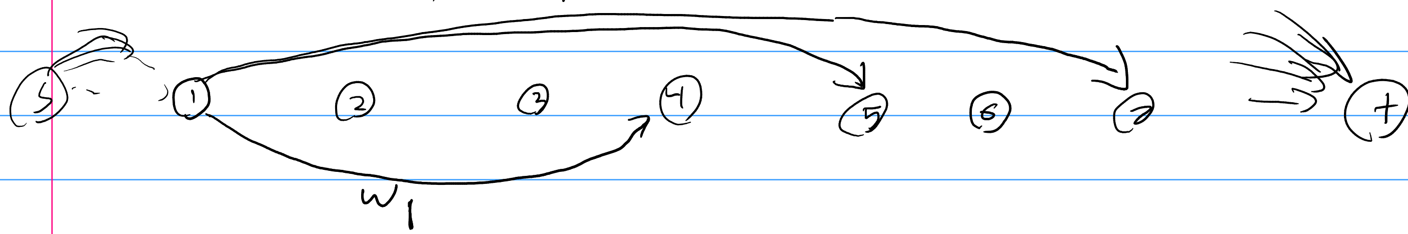
$\Rightarrow$ memoized time is $n \cdot$ (time per input)

$n^2$ 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 $s \to i_1 \to i_2 \to \ldots \to i_k \to t$

- weight $= 0 + w_{i_1} + w_{i_2} + \ldots + w_{i_k}$
$=$ total value of scheduling them.

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

$\Rightarrow$ 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 $s_1, \ldots, s_n$

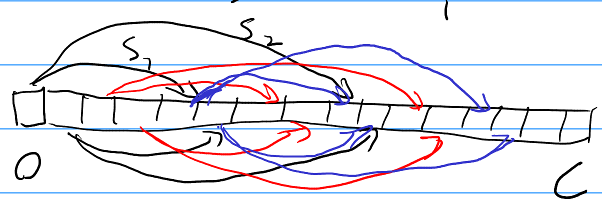
want collection of value $C$.

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 = $C \cdot n$.