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?

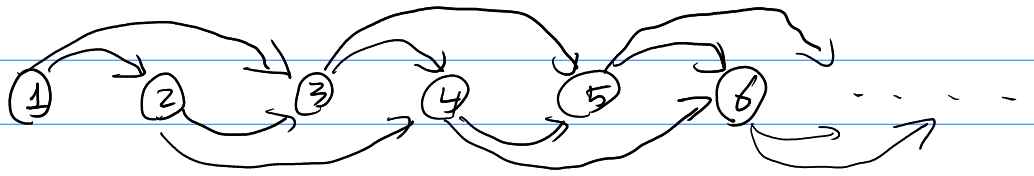
Memoization: whenever you compute $f(x)$ in some recursive call, store the answer. Next time you call $f(x)$, just return it.

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

Recursion: time = # paths from input to base case

Can think of inputs as a DAG.
A → 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 <u>Pull</u> or <u>Push</u> along edges.

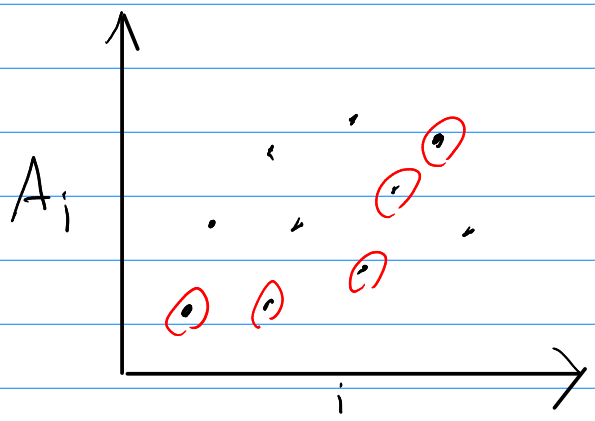
Pull: compute a nodes value when you visit it, based on previous values

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

(In Fibonacci: add your value to the 1's)

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)

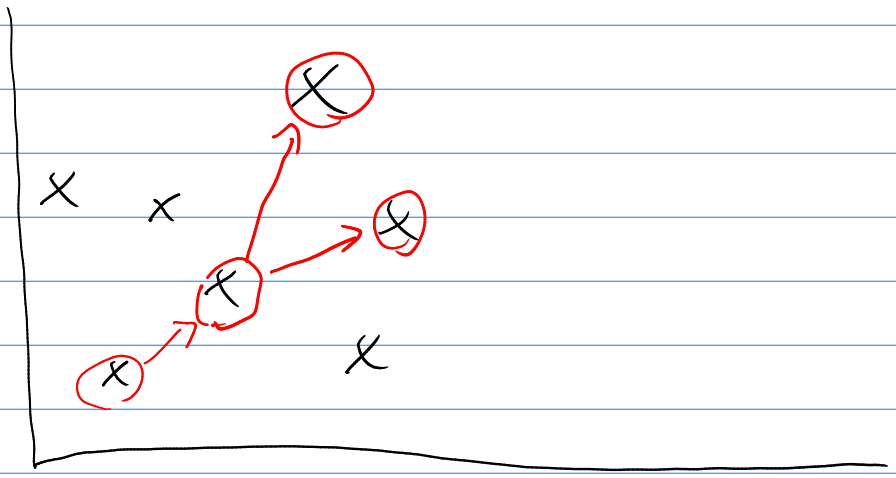
of maximum length k.

---

will convert to longest path on DAG

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

Suppose you build your solution $S_1, \ldots, S_n$ 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 (∞, ∞)

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

$$f_A(i) := \text{LIS ending at } A_i$$

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

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

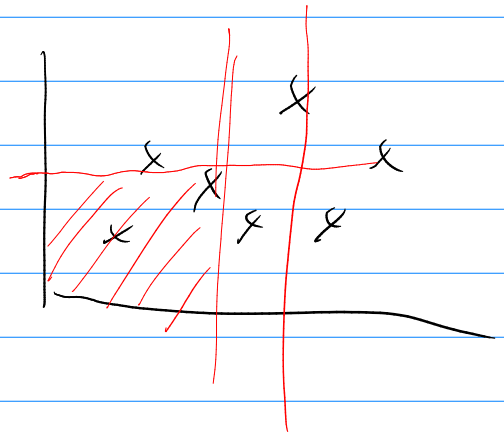
OR (for poly time)

if $f(A) = \text{LIS of } A$.

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

$$f(A) = \max\left(f(A \setminus A_i),\ f(A_{1,\dots,} A'_{i-1}) + 1 \right)$$

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 $A_1, \ldots, A_n$:

$s_1, \ldots, s_i$

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

When is one clearly superior?
When are they equivalent?

Answer: only will care about length & last value

$f_n(K) :=$ minimal last value of length-K subsequence on $1, \ldots, n$

$f_{n+1}(K) = \min(f_n(K),$ ← don't use A_{n+1}
A_{n+1} if $f_n(K-1) < A_{n+1})$

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

if $A_{n+1} \in (z_t, z_{t+1})$:

f_{n+1}: $[z_1, z_2, \ldots, z_t, A_{n+1}, z_{t+2}, \ldots, z_\ell]$

$\Rightarrow$ 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.

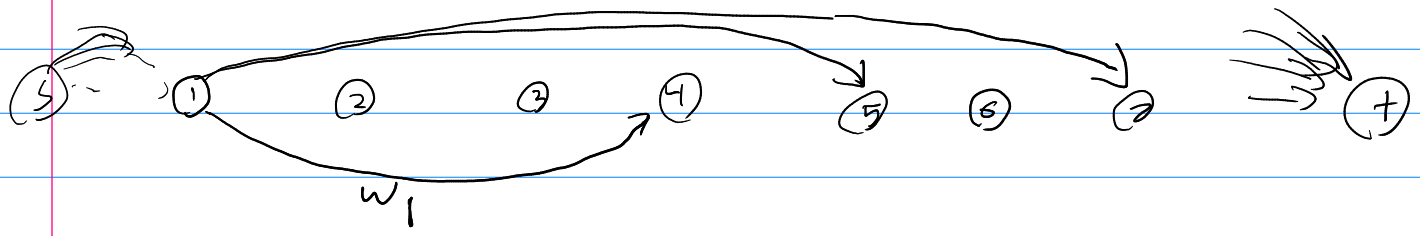
Naive recursion:

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 $s \to i_1 \to i_2 \to \cdots \to i_k \to t$

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

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

$\Rightarrow$ is valid schedule.

Many DP problems have this form:

Convert to a DAG
Find max-weight (or min-weight)
$s \to t$ path.
Time = # edges in DAG.

Stamps

values $s_1, \ldots, s_n$

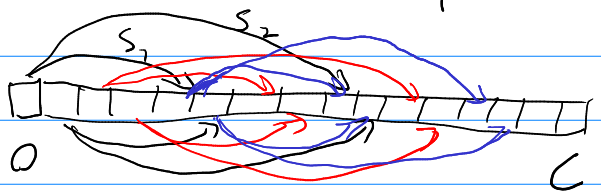
want collection of value C.

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 = $C \cdot n$.