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 = \( \frac{\text{possible}}{\text{inputs}} \) \( \times \) 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 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:

$s_1 < s_2 < \ldots < s_k \quad \text{(subsequence)}$

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

of maximum length $k$.

will convert to longest path on DAG
\[\text{O}(n^2) \text{ bottom-up approach:}\]

Suppose you build your solution \( S_1, S_2, \ldots, S_m \) from left to right.

Walking a path: \( (S_i, A_k) \rightarrow (S_j, A_l) \rightarrow \ldots \)

on the DAG \( (i, A_i) \rightarrow (j, A_j) \quad \Leftrightarrow \quad i < j \quad \& \quad A_i < A_j \)

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

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

\[ = \max_{j < i} f_A(j) + 1 \quad \text{if } A_j < A_i \]

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

OR (for poly time)

\[ f(A) = \text{LIS of } A. \]

- let \( i = \arg \max A_i \)
- \( \text{LIS either contains } i \text{ or not} \)

\[ f(A) = \max \left( f(A \setminus A_i), f(A_i, A_i), f(A_i, A_i, A_i) + 1 \right) \]

How many possible inputs?

Bar to LL

\[ = n^2 \text{ options} \]

+ \( n \) time per option

\[ = n^3 \]
O(n log n) Version

Given two possible starts on A, i.e., A_m:

\[ s_i, \quad s_j \]

\[ s_i, \quad s_j \]

When is one clearly superior?
When are they equivalent?

Answer: only will care about
length & last value

\[ F_m(K) := \text{minimal last value} \]
\[ \text{of length-} K \text{ subsequence} \]
\[ 1, \ldots, M \]

\[ F_{m+1}(K) = \min \{ F_m(K) < \text{don't use} A_{m+1}, \]
\[ A_{m+1}, \text{if } F_m(K-1) < A_{m+1} \} \]

\[ F_m: [Z_1, Z_2, \ldots, Z_t, Z_{t+1}, \ldots, Z_M] \]

\[ + A_{m+1} \in (Z_t, Z_{t+1}) : \]
\[ F_{m+1}: [Z_1, Z_2, \ldots, Z_t, A_{m+1}, Z_{t+1}, \ldots, Z_M] \]

\[ = \text{update is } O(\log n) \text{ binary search} \]
\[ = \text{ } O(n \log n) \text{ time} \]
Interval Scheduling:

Want to compute $\text{Sched}(I)$ where $I = \text{set of } (s_i, f_i, w_i)$ pairs

\[
\text{maximize } \sum_{i \in S} w_i
\]

for $S \subseteq I$ non-overlapping.

Naive recursion:

$\text{Sched}(I)$

Let $i = \text{first elt of } I$

Return min of

not chosen: $\text{Sched}(I \setminus i)$,

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

Problem: $2^n$ possible inputs.

Solution: If I sorted by $f_i$, then

only $n+1$ inputs ever happen:

(suffix or I sorted by $s_i$)

\[
\Rightarrow \text{ memoized time is } n \cdot (\text{time per input})
\]

$n^2$ naively

$n$ more carefully,

$\text{Sched (index in } I, f \text{ or last chosen)}$
DAG: sort by $f_i$.

1 $\rightarrow$ j if $f_i \leq S_j$, or weight $w_i$.

$S \rightarrow$ everything weight 0

everything $\rightarrow$ $+$ weight $w_i$

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 till

- $f_{i_1} \leq S < f_{i_2} \leq S < \ldots$

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

What is the DAG?

Nodes = value

$x \rightarrow x + s_i$ for $i, x$, of weight 1

Answer = shortest $0 \rightarrow C$ path.

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

$0$ $x$ $s_2$ $s_3$ $s_n$ $C$