Today: Dynamic Programming I (of 4)
- memoization & subproblems: bottom up
- Fibonacci examples
- shortest paths
- guessing & DAG view

Dynamic programming (DP) - big idea, hard, yet simple
- powerful algorithmic design technique
- large class of seemingly exponential problems have a polynomial solution ("only") via DP
- particularly for optimization problems (min/max) (e.g. shortest paths)

*p DP ≈ careful brute force
*p DP ≈ recursion + "re-use"

History: Richard E. Bellman (1920-1984)
"Bellman...explained that he invented the name 'dynamic programming' to hide the fact that he was doing mathematical research at RAND under a Secretary of Defense who 'had a pathological fear and hatred of the term, research.' He settled on the term 'dynamic programming' because it would be difficult to give a 'pejorative meaning' and because 'It was something not even a Congressman could object to.'" [John Rust 2006]
Fibonacci numbers: \( F_1 = F_2 = 1; \quad F_n = F_{n-1} + F_{n-2} \)

\textbf{- goal:} compute \( F_n \)

\textbf{Naive algorithm:} follow recursive definition

\begin{align*}
\text{fib}(n) : & \\
\text{if } n \leq 2: & f = 1 \\
\text{else: } & f = \text{fib}(n-1) + \text{fib}(n-2) \\
\text{return } f & \\
\end{align*}

\[ T(n) = T(n-1) + T(n-2) + O(1) \quad \Rightarrow F_n \approx \varphi^n \]

\[ \geq 2T(n-2) + O(1) \geq 2^{n/2} \quad \text{EXponential - Bad!} \]

\textbf{Memoized DP algorithm:} remember, remember!

\begin{align*}
\text{memo} &= \{ \} \\
\text{fib}(n) : & \\
\text{if } n \text{ in memo: } & \text{return memo}[n] \\
\text{if } n \leq 2: & f = 1 \\
\text{else: } & f = \text{fib}(n-1) + \text{fib}(n-2) \\
\text{memo}[n] &= f \\
\text{return } f & \\
\end{align*}

\[ \Rightarrow \text{fib}(k) \text{ only recurses first time called, } \forall k \]

\[ \Rightarrow \text{only } n \text{ nonmemoized calls: } k=n,n-1,\ldots,1 \]

\text{- memoized calls free (}\Theta(1)\text{ time)}

\[ \Rightarrow \Theta(1) \text{ time per call (ignoring recursion)} \quad \text{POLYNOMIAL - Good!} \]
**DP** ≈ recursion + memoization
- memoize (remember) & re-use solutions to subproblems that help solve problem
  - in Fibonacci, subproblems are \( F_1, F_2, \ldots, F_n \)

\[ \Rightarrow \text{time} = \# \text{subproblems} \cdot \frac{\text{time/ subproblem}}{\text{}} \]
- Fibonacci: \( n \)  
  \[ \Rightarrow \Theta(1) = \Theta(n) \]
  ignore recursion!

**Bottom-up DP algorithm:**
```
fib = {??}
for k in [1, 2, ..., n]:
    if k <= 2:  f = 1
    else:  f = fib[k-1] + fib[k-2]
    fib[k] = f
return fib[n]
```

- exactly the same computation as memoized DP (recursion "unrolled")
- in general: topological sort of subproblem dependency DAG  
  \[ \Rightarrow \Theta(1) \]
- practically faster: no recursion
- analysis more obvious
- can save space: just remember last 2 fibs

[sidetext: there is also an \( \Theta(\log n) \)-time algorithm for Fibonacci via different techniques]
Shortest paths:
- recursive formulation:
  \[ S(s, v) = \min \{ S(s, u) + w(u, v) \mid (u, v) \in E \} \]
- memoized DP algorithm:
  takes infinite time if cycles! (kinda necessary to handle neg. cycles)
  - works for directed acyclic graphs in \( O(V+E) \)
    ~ effectively DFS/Topological sort + Bellman-Ford round rolled into a single recursion

\[ \star \]
Subproblem dependency should be acyclic

- more subproblems remove cyclic dependence:
  \[ S_k(s, v) = \text{shortest } s \to v \text{ path using } \leq k \text{ edges} \]
- recurrence:
  \[ S_k(s, v) = \min \{ S_{k-1}(s, u) + w(u, v) \mid (u, v) \in E \} \]
  \[ S_0(s, v) = \infty \text{ for } s \neq v \]
  \[ S_k(s, s) = \emptyset \text{ for any } k \] \{ base case \}
  \{ if no neg. cycles \}
- goal: \( S(s, v) = S_{|V|-1}(s, v) \)

- memoize
- time: \( \# \text{ subproblems} \cdot \frac{O(V)}{v \in V} = O(V^3) \)
  \( \Rightarrow \) actually \( \Theta(\text{indegree}(V)) \) for \( S_k(s, v) \)
  \( \Rightarrow \) time = \( \Theta \left( V \sum_{v \in V} \text{indegree}(V) \right) = \Theta(VE) \)

Bellman-Ford!
**Guessing:** how to design recurrence
- want shortest \( s \rightarrow v \) path \( s \rightarrow \cdots \rightarrow u \rightarrow v \)
- what is the last edge in path? dunno
- guess it's \((u,v)\)
  \(\Rightarrow\) path is shortest \( s \rightarrow u \) path + edge \((u,v)\)
  by optimal substructure
  \(\Rightarrow\) cost is \( \delta_{k-1}(s,u) + w(u,v) \)
  another subproblem
- to find best guess, try all & use best \( \Rightarrow \) LV choices

\* \[ - \text{key: small (polynomial) # possible guesses per subproblem} \]
  - typically this dominates time/subproblem

\* [DP \( \approx \) recursion + memoization + guessing

**DAG view:**
- like replicating graph to represent time
- converting shortest paths in graph
  \( \Rightarrow \) shortest paths in DAG

\* [DP often \( \approx \) shortest paths in some DAG
  \( \Rightarrow \) (but not always!)