Today: Dynamic Programming I (of 4)
- memoization & subproblems; bottom up
- Fibonacci
- 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)

\[ \text{DP} \approx \text{careful brute force} \]
\[ \text{DP} \approx \text{recursion + "re-use"} \]

\[ \text{History: Richard E. Bellman (1920-1984)} \]

"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: \( F_1 = F_2 = 1; \) \( F_n = F_{n-1} + F_{n-2} \)

**Goal:** compute \( F_n \)

**Naive algorithm:** follow recursive definition

\[
fib(n): \begin{cases} 
  \text{if } n \leq 2: & f = 1 \\
  \text{else: } & f = fib(n-1) + fib(n-2) \\
\end{cases}
\]

\[
\Rightarrow T(n) = T(n-1) + T(n-2) + O(1) \geq F_n \approx \varphi^n \\
\geq 2T(n-2) + O(1) \geq 2^{n/2} \quad \text{EXponential - BAD!}
\]

**Memoized DP algorithm:** remember, remember!

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

\[
\Rightarrow fib(k) \text{ only recurses first time called, } \forall k \\
\Rightarrow \text{only } n \text{ nonmemoized calls: } k = n, n-1, \ldots, 1 \\
\text{- memoized calls free } (\Theta(1) \text{ time}) \\
\Rightarrow \Theta(1) \text{ time per call (ignoring recursion)} \quad \text{POLYNOMIAL - GOOD!}
\]
**[DP ≈ recursion + memoization]
- memoize (remember) & re-use solutions to subproblems that help solve problem
- in Fibonacci, subproblems are F₁, F₂, ..., Fₙ

⇒ time = # subproblems • time/subproblem

- Fibonacci: n
  - ✗ Θ(1) ≠ Θ(n)
  - ignore recursion!

**Bottom-up DP algorithm:**

```python
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
- practically faster: no recursion
- analysis more obvious
- can save space: just remember last 2 fibs

⇒ Θ(1)

[sidetext: there is also an O(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

\*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:
  \( S_k(s,v) = \min \{ S_{k-1}(s,u) + w(u,v) \mid (u,v) \in E \} \)
  \( S_0(s,v) = \infty \text{ for } s \neq v \) \( \{ \text{base case} \} \)
  \( S_k(s,s) = \emptyset \text{ for any } k \) \( \{ \text{if no neg. cycles} \} \)
- goal: \( S(s,v) = S_{|V|-1}(s,v) \)

- memoize
- time: \( \frac{|V|! \cdot |V| \to k}{\sum_{v \in V} \text{indegree}(v)} \cdot \frac{|V|!}{|V|} = O(V^3) \)
  
  \( \Rightarrow \) 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 $\Rightarrow$ optimal choices

* $\begin{align*}
- \text{key: small (polynomial) \# possible guesses per subproblem,}
- \text{typically this dominates time/subproblem}
\end{align*}$

$\star$ $\begin{align*}
\text{DP} \approx \text{recursion + memoization + guessing}
\end{align*}$

**DAG view:**

- like replicating graph to represent time
- converting shortest paths in graph
  $\Rightarrow$ shortest paths in DAG

$\star$ $\begin{align*}
\text{DP} \approx \text{shortest paths in some DAG}
\end{align*}$