Outline: Dynamic Programming I (of 4)
- Fibonacci warmup
- memoization & subproblems
- shortest paths
- Crazy Eights
- guessing viewpoint

Reading: CLRS 15

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)

* DP ≈ "controlled brute force"
* DP ≈ recursion + re-use
Fibonacci numbers: \( F_1 = F_2 = 1; \) \( F_n = F_{n-1} + F_{n-2} \)

- **naïve algorithm:** follow recursive definition

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

\[
T(n) = T(n-1) + T(n-2) + O(1) = 2^{n/2} \approx \phi^n
\]

**EXPOENTIAL - BAD!**

- **simple idea:** memoize

\[
\text{memo} = \emptyset
\]

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

\[
T(n) = T(n-1) + O(1) = O(n)
\]

[side note: there is also an \( O(\log n) \)-time algorithm for Fibonacci via different techniques]
* DP ≈ recursion + memoization
  - remember (memoize) previously solved "subproblems" that make up problem
  - in Fibonacci, subproblems are F₀, F₁, ..., Fₙ
  - if subproblem already solved, re-use solution
  - \( \Rightarrow \text{time} = \# \text{subproblems} \cdot \text{time/subproblem} \)

\[ \text{fib: } O(n) \cdot O(1) = O(n) \]

Shortest paths:
- recursive formulation:
  \[ S(s, t) = \min \{ w(s, v) + S(v, t) \mid (s, v) \in E \} \]
- does this work with memoization?
  \[ \Rightarrow \text{no. cycles} \Rightarrow \text{infinite loops} \]
- in some sense necessary for neg.-weight cycles
- works for directed acyclic graphs in \( O(V + E) \)
  (recursion effectively DFS/topological sort)
- trick for shortest paths: removing cyclic dep.
  - \( S_k(s, t) = \text{shortest path using } \leq k \text{ edges} \)
  \[ = \min \{ S_{k-1}(s, t), \min_{v \in V} \{ w(s, v) + S_{k-1}(v, t) \mid (s, v) \in E \} \} \]
  \[ \text{... except } S_k(t, t) = 0, S_k(s, t) = \infty \text{ if } s \neq t \]
- \( S(s, t) = S_{n-1}(s, t) \) assuming no neg. cycles
  \[ \Rightarrow \text{time} = \# \text{subproblems} \cdot \text{time/subproblem} \]
  \[ = O(n^3) \text{ for } s, t, k \ldots \text{really } O(n^2) \]
  \[ = O(n) \ldots \text{really } \deg(v) \]
  \[ = O(V \cdot \sum \deg(v)) = O(VE) \]

* Subproblem dependency should be acyclic
Crazy Eights puzzle:
- given a sequence of cards \( c[0], c[1], \ldots, c[n-1] \)
  - e.g., 7♥️, 6♥️, 7♦️, 3♣️, 8♣️, J♠️
- find longest left-to-right "trick" (subsequence) \( c[i_1], c[i_2], \ldots, c[i_k] \) \((i_1 < i_2 < \ldots < i_k)\)
  - where \( c[i_j] \& c[i_{j+1}] \) "match" for all \( j \):
  - have same suit or rank or one has rank 8
- recursive formulation:
  \[
  \text{trick}(i) = \text{length of best trick starting at } c[i] = 1 + \max \left( \text{trick}(j) \text{ for } j \text{ in } \text{range}(i+1, n) \right) \\
  \text{if } \text{match}(c[i], c[j])
  \]
  - best = \( \max(\text{trick}(i) \text{ for } i \text{ in } \text{range}(n)) \)
- memoize: \( \text{trick}(i) \) depends only on \( \text{trick}(>i) \)
  \[
  \Rightarrow \text{time} = \frac{\# \text{subproblems}}{\text{time/subproblem}} = O(n^2) \quad \text{(to find actual trick, trace through max's)}
  \]

"Guessing" viewpoint:
- what is the first card in best trick? guess!
  - i.e. try all possibilities & take best result
  - only \( O(n) \) choices
- what is next card in best trick from \( i \)? guess!
  - if you pretend you knew, solution becomes easy (using other subproblems)
  - actually pay factor of \( O(n) \) to try all
  - use only small # choices/guesses per subproblem

\(*\)