Admin:

Reading: CLRS 15.1-15.4

Outline: Dynamic Programming (2/4)
- Review DP concepts
- Longest path in a DAG
- Longest common subsequence
- Picking up pennies
Review Dyn. Prog. concepts

• Typically, an optimization problem that can be solved recursively.
• Recursive calls generate similar subproblems.
• Subproblem dependence graph. (DAG: no cycles!)
  \[ V = \text{set of subproblems} \]
  \[ E = \{ x \to y : \text{solving } x \text{ involves calling for } y \text{'s solution} \} \]

• Two approaches:
  1. "Top-down" (recursive) implementation, but
     "memoizing" solutions to subproblems as they are solved
     \[ \text{DFS} \]
  2. "Bottom-up": sort \( V \) in reverse topological order,
     solve subproblems in that order, save solutions as
     subproblems are solved.

• Solution time is typically \( \Theta(V + E) \), assuming that
  solving a subproblem is linear in number of recursive
  calls it makes (i.e., in \# of outgoing edges it has).
Longest Path in a DAG

- Find longest \( S \rightarrow T \) path in given DAG \( G=(V,E) \)

**Diagram:**

- Subproblem for each vertex \( u \):
  - What is length of longest \( u \rightarrow T \) path, and
  - What is first vertex (after \( u \)) on this path?

  ("single destination" rather than "single source"—we've turned things around for convenience, no big deal.)

- **Equation:**
  \[
  d[u] = \max_{(u,v) \in E} \left( w(u,v) + d[v] \right) \quad \text{(but } d[t] = 0) \]

- Do DFS on \( G \), starting at \( S \),
  - Save ("memoize") \( d[u] \) when \( u \) "finished".
  - (Illustrate)
  - Time is \( \Theta(V+E) \)

- Can find "critical path" in project ("PERT chart") similarly
  \[
  d[u] = w(u) + \max_{(u,v) \in E} \left[ d[v] \right] \quad \text{(but } d[t] = w(t))
  \]

where \( w(u) = \text{time to complete task } u \) on its own.
"Picking up pennies"

Given a DAG $G = (V,E)$, where some edges have pennies on them & some don't, find path from given start vertex with most pennies.

This is just longest path problem, where length of edge = # of pennies on it!
Longest Common Subsequence (LCS)

- **Apps:** "edit distance", "diff", spell-checking, DNA comparison, document distance (new def.), plagiarism detection, etc.

- **Given two strings/sequences** $x$ and $y$, **find a longest common subsequence** (elements in some order but not nec. continuous)

- **Example:** $x = \text{CATG}$ or $x = \text{CATG}$
  
  $y = \text{ACGT}$ or $y = \text{ACGT}$
  
  $z = \text{LCS}(x,y) = \text{AC}$ or $z = \text{LCS}(x,y) = \text{CG}$

  (may be more than one LCS)

- **Naive brute force:** try all $2^{|x|}$ subsequences of $x$
  
  see if it occurs in $y$
  
  time = $\Theta (2^{|x|} \cdot |y|)$

- **Consider "chopping game" - try to maximize # points**
  
  Given two strings, do a sequence of moves until nothing is left.

  **Move** = drop 1st char of $x$ (0 pts)
  
  $\text{CATG}$ or $\text{CATG}$

  or drop 1st char of $y$ (0 pts)
  
  $\text{ACGT}$ or $\text{ACGT}$

  or drop 1st char of $x$&$y$ (1 pt if same char dropped else 0 pts)

- **Claim:** max score achievable = length of LCS

  (Just take letters corresponding to points achieved.)
• State of game = subproblem
  = pair of suffixes \((x[i:], y[j:])\)
  after having dropped \(i\) from \(x\) & \(j\) from \(y\)

• Draw subproblem dependence graph

\[
\begin{array}{cccccc}
X & C & A & T & G \\
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
Y & A & C & G \\
0 & 1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\rightarrow &=& \text{drop \(x\)} & \Rightarrow \text{all edges have 0 weight} \\
\downarrow &=& \text{drop \(y\)} & \Rightarrow \text{except for \(\downarrow\) with \(x[i] = y[j]\)} \\
\rightarrow &=& \text{drop both} & \Rightarrow \text{want longest path (!)} \\
\end{array}
\]

Time = \(\Theta(V+E) = \Theta(|x|\cdot|y|)\)

Space = \(\Theta(|x|\cdot|y|) \Rightarrow \Theta(\min(|x|,|y|))\)
  just to find length
do by rows, or columns