Announcements

1. Problem Set 1 is now graded.
2. Recitations: some TAs will be providing optional recitations solving sample problems.
   Stay tuned for announcement on Ed and recitation calendar on website.

RECAP. Bellman-Ford algorithm takes
- directed graph $G = (V, E)$ with $n$ vertices, $m$ edges
- source $s \in V$, destination $t \in V$
- cost function $\text{Cost} : E \rightarrow \mathbb{R}$ (including possibly negative costs)
and outputs the minimum cost walk from $s$ to $t$, if one exists.

If $G$ has no negative cost cycles that can occur in a walk from $s$ to $t$,
$\text{Cost} = O(mn)$ time.
Reducing Problem A to Problem B.

- Using an alg that solves B as a subroutine to solve A.
- Most common type calls to subroutine only once. "Karp reduction"

Input to A → Translate → Input to B

1. Running time analysis should account for all 3 steps.
2. Proof must show input to B is properly formatted.
3. Solve using pre-existing alg for B
4. Output of B

Output of A → Translate → Output of B

Proof of correctness assumes alg for B is correct and only needs to show correct output for A assuming correct output for B.
Illustrations...

1. **Longest path in a directed acyclic graph.**

   Given directed $G = (V, E)$ with no cycles.

   - **source** $s$, **destination** $t$.
   - **length**: $E \to \mathbb{R}$

   Find path from $s$ to $t$ with maximum length.

   Solution: set $cost(e) = -length(e)$ and run Bellman-Ford.

   - **Longest path**
   - **negative length** $O(m)$
   - **shortest path**
   - Bellman Ford $O(mn)$

   $O(m) + O(mn) + O(1)$

   $= O(mn)$. 

   Identity $O(1)$
2. Weighted interval scheduling.

\[ A = \text{weight int. sched} \]

\[ B = \text{longest path in DAG.} \]

\[ \{4, 8\} \]

Aside from the 5 edges labeled with numbers, all others have length 0.

This works because

\[ \{\text{set paths in } G\} \leftrightarrow \{\text{conflict-free sets of intervals}\} \]

via a one-to-one correspondence that transforms path length to weight of an interval set.
Running time: graph has $2n+2$ vertices. Reduction creates $3n$ edges plus any additional "purple" edges between compatible pairs, at most $(\binom{n}{2})$ such pairs.

DAG has $N = 2n+2$ vertices, $M = O(n^3)$ edges.

Running time $O(MN) = O(n^6)$.

Remember to translate running time analysis in terms of size of input to $B$ back into an expression that relates it to the input size of problem $A$.

Exercise: Try translating Friday’s knapsack algorithm into a reduction from knapsack to shortest path (or longest path in a DAG).
RNA secondary structure prediction (§6.5)

RNA is a single-stranded molecule made of a sequence of "bases" in the set \{A, C, G, U\}. These can pair with one another: A pairs with U, C pairs with G, causing the molecule to fold on itself!

A simplified form of the rules for this self-pairing:

1. No sharp turns: if base i pairs with base j, then |i - j| > 4.
2. A pairs with U, C with G.
3. Matching property: each base pairs with at most one other.
4. Non-crossing: if i < j < k < l then
Subject to these rules, the molecule tends to form as many pairs as possible.

The last base in the sequence (base $n$) is either paired with $j < n-1$ or unpaired.

- If unpaired, the best we can do is compute the optimal pairing of bases $1, \ldots, n-1$.

- If paired with $j$
Non-crossing $\Rightarrow$ all remaining pairs are contained within the purple segment of the brown segment.

\[
\text{OPT}(a,b) = \max \left\{ \begin{array}{c} \text{OPT}(a, b-1) \\ \max_{j \in P(n)} \left\{ 1 + \text{OPT}(a_{j-1}) + \text{OPT}(j+1, b-1) \right\} \end{array} \right\}
\]

\[P(n) = \begin{cases} \{j \mid j < b-1 \text{ and } \} & \\
\{b \neq A \text{ and } j = \text{"U"} \} & \\
\{b \neq C \text{ and } j = \text{"G"} \} & \\
\text{etc.} & \\
\end{cases}\]
for \( l = 0, \ldots, n-1 \)
  for \( a = 1, \ldots, n-l \)
    if \( l = 0 \)
      \( \text{OPT}[a, a+l] = 0 \)
    else
      compute \( \text{OPT}[a, a+l] \) using
    endif
  endfor
endfor

Output \( \text{OPT}(1, n) \).