Admin:
- PS 6 due tonight (1 grace day max.)
- Final Exam, Monday 16 May, 1:30-4:30 pm, Johnson Track
- Problem Solving Session - led by instructors
  - Thursday, 12 May, 7:00-9:00 pm, 26-100
  - Dinner food available

Today:
- Two more dynamic programming problem scenarios
- Both involve string comparisons
- Useful in multiple application areas
  - Spell checking/correction
  - Difference finding (cliff, merge documents, ...)
  - Plagiarism detection
  - Genomics (protein, RNA, DNA sequence comparison)
Longest Common Subsequence

Given: Two sequences
\[ X = [x_1, x_2, \ldots, x_m] \]
\[ Y = [y_1, y_2, \ldots, y_n] \]

Find: A sequence
\[ Z = [z_1, z_2, \ldots, z_k] \]
that is a maximum-length common subsequence of \( X \) and \( Y \)

(\( Z \) is a subsequence of \( X \) if all of the elements of \( Z \) appear in order within \( X \), although possibly interspersed with other elements of \( X \))

Example:

\( X: \) Ala Lys Val Glu Asp Phe
\( Y: \) Lys Val Cys Glu Phe Tyr

\( Z: \) Lys Val Glu Phe is a longest common subsequence of \( X \) and \( Y \).
Step 0: Enumeration/Optimization

All subsequences of \( X \) (or \( Y \)) can be constructed. There are \( 2^m \) (or \( 2^n \)) of them, because each sequence element can be included or excluded, combinatorially. Each subsequence can of \( X \) can be checked to see if it is also a subsequence of \( Y \) (or vice versa).

Total time = \( \Theta(\min(m \cdot 2^n, n \cdot 2^m)) \)

Step 1: Optimal subproblem structure/recursion with base case

Consider shrinking problem from end:

\[
\begin{align*}
(X_{1\ldots m}, Y_{1\ldots n}) &= (X_{1\ldots(m-1)}, Y_{1\ldots n}) & (X_{1\ldots m}, Y_{1\ldots(n-1)}) & (X_{1\ldots(m-1)}, Y_{1\ldots(n-1)}) \\
n\text{no match} &\Rightarrow n\text{ no match} & \text{if } x_m = y_n \text{ match } +1 & \text{if } x_m \neq y_n \text{ no match} + 0
\end{align*}
\]
Let $c[i,j] = \text{length of LCS for } X_1..i, Y_1..j$

$$c[i,j] = \begin{cases} 
0 & \text{if } i=0 \text{ or } j=0 \quad \text{base case} \\
(c[i-1,j-1]) + 1 & \text{if } i,j>0 \text{ and } x_i=y_j \quad \text{match} \\
\max(c[i-1,j], c[i,j-1]) & \text{if } i,j>0 \text{ and } x_i\neq y_j \quad \text{the 3rd is a combination of these} \quad \rightarrow \text{2 cases of mismatch} 
\end{cases}$$

The optimal substructure property can be shown by a cut-and-paste argument in the theorem 15.1 proof in CLRS.

Step 2: Subproblem dependence graph a DAG?
Trivially yes, because each subproblem depends on smaller subsubproblems, so there can’t be any cycles.

If assign zero weight to $\bullet (x_i\neq y_j)$, $\uparrow$, $\downarrow$, and unit weight to $\bullet (x_i=y_j)$, then seek longest path in a DAG.
Step 3: Memorize/Store for reuse? \( c[i,j] \) \( 0 \leq i \leq M \)

Step 4: Write algorithm. See CLRS Chpt 15.4

Running Time Analysis

Because a constant amount of work is done in each recursive step (not including time spent in recursive calls), running time proportional to \( \Theta(V+E) \) in subproblem graph.

\[ \Rightarrow \Theta(m \cdot n) \]

**Edit Distance**

Given: (a) Two sequences \( X = [x_1, x_2, \ldots, x_m] \)
\[ Y = [y_1, y_2, \ldots, y_n] \]

(b) Catalog of edit functions and their associated cost

insert: +1
delete: +1
substitute: +1

Find: Minimum edit cost for converting one string into the other. (Cost is the same for the current catalog and costs for \( X \rightarrow Y \) and \( Y \rightarrow X \))
Example: Given \( \text{X: EXPONENTIAL \quad Y: POLYNOMIAL} \)

Imagine an algorithm that identifies each alignment, like that below, that corresponds to

\[
\begin{align*}
\text{X: EXPONENTIAL} \\
\text{Y: POLYNOMIAL}
\end{align*}
\]

\[x \rightarrow y: \text{del del } - \text{ sub sub } \quad \text{ins sub } - - -
\]

\[
\text{cost: } +1 \quad +1 \quad +1 \quad +1 \quad +1 \quad \rightarrow \text{cost = 6}
\]

Step 0: Enumeration/optimization
- All possible edit combinations sound daunting
- We can more simply imagine all possible alignments and assign a cost to each.
- Here the cost is the number of mismatches.

Step 1: Subproblems/recurrence with basecase/opt. subst.
- Similar to LCS, smaller versions obtained by removing one character from the end of one string, the other, or both could be good subproblems. Rightmost column of alignment must be one of: \( [X_i \mid \text{or} \mid Y_i \mid \text{or} \mid X_i] \)

Let \( c[i,j] = \text{cost of minimum edit distance} \)

\[
c[i,j] = \begin{cases} 
0 & \text{if } X_i = Y_j \\
\min \{ c[i-1,j-1], c[i-1,j], c[i,j-1] \} + 1 & \text{otherwise}
\end{cases}
\]

for \( X_1 \ldots i \rightarrow Y_1 \ldots j \)
Base case

→ \( c[i, 0] \) corresponds to \( X_{1..i} : EXP..i \)

so \( c[i, 0] = i \)

→ \( c[0, j] = j \) because \( X_0 : \)

\( Y_{1..j} : P O L \ldots j \)

\( \# \) mismatches = \( j \)

Step 2: We know the subproblem dependence graph is a DAG because each subproblem only depends on smaller subproblems.

Step 3: Memoize/Store \( c[i, j] \) for reuse

- Can fill this two dimensional table in any order that solves all subsubproblems before they are needed for any particular subproblem.
  - Examples: Filling rows top-to-bottom and each row left-to-right; or filling columns left-to-right and each column top-to-bottom.
Step 4: Algorithm (bottom-up)

for $i \leftarrow 0$ to $m$
    $c[i, 0] \leftarrow i$

for $j \leftarrow 0$ to $n$
    $c[0, j] \leftarrow j$

for $i \leftarrow 1$ to $m$
    for $j \leftarrow 1$ to $n$

    if $x[i] = y[j]$
        $c[i, j] \leftarrow c[i-1, j-1]$
    else
        $c[i, j] \leftarrow 1 + \min(c[i-1, j], c[i, j-1], c[i-1, j-1])$

return $c[m, n]$

Below is the final array for $c[i, j]$

<table>
<thead>
<tr>
<th>\text{POLYNOMIAL}</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{E}</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>\text{X}</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>\text{P}</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>\text{O}</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>\text{N}</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>\text{T}</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>\text{E}</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>\text{N}</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>\text{T}</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>\text{A}</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>\text{L}</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>\text{L}</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

The arrows point along subproblem solutions that are min-cost.