CS314 - More dynamic programming

Announcements

- Turn in HW1
- HW2 - posted tonight
Edit distance

The minimum number of deletions, insertions, and substitutions of letters to transform between two strings.

Ex: FOOD \rightarrow MOOD

MONED \rightarrow MOED

MONEY

\leq 4
First: why do we care?

Spell checking
Auto correcting
Strings in gameshows

Second: any ideas?

2D matrix to math
Column format:

\[
\begin{array}{c}
\text{ALGORITHM} \\
\text{ALTRUISTIC}
\end{array}
\]

Inserted letters:

[THM]

Deleted letters:

So edit distance is \( \leq 6 \).
Nice property: If you delete the last column of the previous representation, must still be optimal.

**PF:** (contradiction) Assume not

If better one in first $n-1$, use it.

Contradiction: above wasn't correct edit distance.
So- recursive formulation:

Consider \( A[1..m] \) and \( B[1..n] \).

What are the 3 possibilities for just the \( \text{last character} \)?

1. \( A[m] \)
2. \( A[m] \)
3. \( A[m] \)

\[ \downarrow \quad \downarrow \quad \downarrow \]

\[ B[n] \quad B[n] \]
So more formally:

Let \( \text{Edit}(A, B) = \text{the edit distance between } A[1..m] \text{ and } B[1..n] \).

\[
\text{Edit}(A[1..m], B[1..n]) = \min \left\{ \begin{array}{l}
\text{Edit}(A[1..m-1], B[1..n]) + 1 \\
\text{Edit}(A[1..m], B[1..n-1]) + 1 \\
\text{Edit}(A[1..m-1], B[1..n-1]) + [A[m] \neq B[n]] \\
\end{array} \right. 
\]

\[A \xrightarrow{A[m]} A[m] \]
\[B[n] \xrightarrow{B[n]} G \stackrel{G+1}{\downarrow} H\]
Base cases?

- empty
- \( n \) insertions

- \( m \) deletions

- empty string (empty)

- \( \text{Edit}(\varepsilon, B[1...n]) = n \)
- \( \text{Edit}(A[1...m], \varepsilon) = m \)
Now again, we have something like LIS & the recursive calls are always on prefixes!

So if we can somehow start with the "front" & move towards "end" we could get the value for \( \text{Edit}(A[1...n], B[1...m]) \).

Let's try to simplify & formalize this idea...
First, $\text{Dist}(A[1 \ldots i], B[1 \ldots j])$ is too long.

Shorten: $\text{Dist}(i, j)$. (Since always a prefix.)

Recurrence (rewritten):

$$
\text{Dist}(i, j) = \begin{cases} 
  i & \text{if } j = 0 \\
  j & \text{if } i = 0 \\
  \min \left\{ \begin{array}{l}
  \text{Dist}(i - 1, j) + 1, \\
  \text{Dist}(i, j - 1) + 1, \\
  \text{Dist}(i - 1, j - 1) + [A[i] \neq B[j]] 
\end{array} \right\} & \text{otherwise}
\end{cases}
$$
This does give a recursive algorithm.

Running time—probably ugly.

But—dynamic programming seems like an option.
The table:

$$Edit(i, j) = \begin{cases} 
  \left( \begin{array}{c} i \\ j \end{array} \right) & \text{if } j = 0 \\
  \min \left( \begin{array}{c} Edit(i - 1, j) + 1, \\
  Edit(i, j - 1) + 1, \\
  Edit(i - 1, j - 1) + [A[i] \neq B[j]] \end{array} \right) & \text{otherwise}
\end{cases}$$
Example:

Edit distance between algorithm and altimetric.

Note: Any path from top right to bottom left is an optimal set of substitutions.
Pseudo code:

\[ \sum_{i=1}^{m} \sum_{j=1}^{n} \]

\[ \text{EDITDISTANCE}(A[1..m], B[1..n]): \]
\[
\text{for } j \leftarrow 1 \text{ to } n \\
\quad \text{Edit}[0, j] \leftarrow j \\
\text{for } i \leftarrow 1 \text{ to } m \\
\quad \text{Edit}[i, 0] \leftarrow i \\
\text{for } j \leftarrow 1 \text{ to } n \\
\quad \text{if } A[i] = B[j] \\
\quad \quad \text{Edit}[i, j] \leftarrow \min \{\text{Edit}[i - 1, j] + 1, \text{Edit}[i, j - 1] + 1, \text{Edit}[i - 1, j - 1]\} \\
\quad \text{else} \\
\quad \quad \text{Edit}[i, j] \leftarrow \min \{\text{Edit}[i - 1, j] + 1, \text{Edit}[i, j - 1] + 1, \text{Edit}[i - 1, j - 1] + 1\} \\
\text{return Edit}[m, n] \]
Running time: $O(n^2)$ space $O(nm)$ same time
Dynamic Programming on Trees

Def: An independent set in a graph is a subset that have no edges between them.

First - why?

Graphs model everything.
Recursion

Goal: Compute largest independent set.

Consider a node \( v \).
The options?

- *include* \( v \)

  recurse on \( G - N(v) \)

- *not include* \( v \)

  recurse on \( G - v \)
\textbf{MaximumIndSetSize}(G):
\begin{itemize}
  \item if $G = \emptyset$
    \begin{itemize}
      \item return 0
    \end{itemize}
  \item else
    \begin{itemize}
      \item $v \leftarrow$ any node in $G$
      \item \texttt{withv} $\leftarrow 1 + \text{MaximumIndSetSize}(G \setminus N(v))$
      \item $\texttt{without}_v \leftarrow \text{MaximumIndSetSize}(G \setminus \{v\})$
      \item return $\max\{\texttt{withv}, \texttt{without}_v\}$.
    \end{itemize}
\end{itemize}

\text{Runtime: } T(n) = 2T(n-1) + \text{poly}(n)
\begin{equation}
  = O(\text{poly}(n) 2^n)
\end{equation}
depends on d.s.
Aside: Can actually do a bit better. How big will these recursive calls be?

(Continued next time... )