A Dynamic Programming Algorithm: Edit Distance

How can we measure how different two strings $s$ and $t$ are? One way is to find the minimum number of single-character edit operations that turn $s$ into $t$. Call this the edit distance, $d(s,t)$. We could allow three operations:

- insert a character;
- delete a character;
- change a character.

Since “stop” can be changed into “ropes” with no fewer than four edits, the distance between these two strings is 4: Change the “s” to “r,” delete “t,” insert “e,” and insert “s.” Other sequences of four edits can also do it. The distance from “ropes” to “stop” is also 4, and in general $d(s,t) = d(t,s)$: make the inserts into deletes and the deletes into inserts, and reverse the character changes.

We use the notation $w_i$ to mean the $i$th character of $w$, and $w_{1..i}$ to mean a string consisting of the first $i$ characters of $w$; so $w_{1..0}$ is the empty string. Let $m$ and $n$ be the lengths of $s$ and $t$, so $s = s_{1..m}$ and $t = t_{1..n}$. Now we can find $d(s,t)$ recursively:

- If both strings are empty ($m = n = 0$), they are the same, so no edits are needed to change one into the other, so $d(s,t) = 0$.
- If $m > 0$, a minimum sequence of edits needed to turn $s_{1..m-1}$ into $t$ can be followed by deletion of the final character $s_m$, so $d(s,t)$ is at most $d(s_{1..m-1},t) + 1$.
- Similarly, if $n > 0$, a minimum sequence of edits needed to turn $s$ into $t_{1..n-1}$ can be followed by insertion of the final character $t_n$, so $d(s,t)$ is at most $d(s,t_{1..n-1}) + 1$, and possibly also bounded by the item mentioned above.
- If $m > 0$ and $n > 0$ and the final characters are the same ($s_m = t_n$) then a sequence of edits that turns $s_{1..m-1}$ into $t_{1..n-1}$ will also turn $s$ into $t$, so in this case $d(s,t) = d(s_{1..m-1},t_{1..n-1})$. If the final characters differ ($s_m \neq t_n$), the sequence can be followed by one more edit to change the final character, so $d(s,t)$ is at most $d(s_{1..m-1},t_{1..n-1}) + 1$, and also bounded by the two items above.

Although I do not prove it, the smallest number of edits to change $s$ into $t$ can always be constructed in one of the above ways. So if we have a table of $d(s_{1..i},t_{1..j})$, with rows for values of $i$ from 0 to $m$, and columns for values of $j$ from 0 to $n$, then we can fill in $d(s_{1..i},t_{1..j})$ with the smallest of the three values $d(s_{1..i-1},t_{1..j}) + 1$; and $d(s_{1..i},t_{1..j-1}) + 1$; and $d(s_{1..i-1},t_{1..j-1}) + c$ where $c$ is 0 if $s_i = t_j$ and 1 otherwise. If $i$ or $j$ is zero, one or more of these three is not defined, and so omitted from the minimum. Thus, each cell depends on the one above, the one to the left, and the one diagonally up and left, with at least one of them (plus 0 or 1) supplying the minimum. The final answer $d(s,t) = d(s_{1..m},t_{1..n})$ is at the bottom right:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>st</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>sto</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Here, I boldface one possible trail of minima leading back from the final answer. This trail gives one possible minimal sequence of edits that transforms “stop” into “ropes”: the one mentioned at the beginning of this sheet.

We can fill in the upper left corner cell trivially. Then, we can proceed from there to fill in the topmost row and the leftmost column. After that, we can fill in the rest row by row. This algorithm runs in $\Theta(mn)$ time.

If we did not use a table (an array) to remember partial results, but instead used recursive function calls, many results at every level would be calculated more than once. For example, $(i,j)$ would calculate $(i-1,j)$, $(i,j-1)$ and $(i-1,j-1)$, but $(i-1,j)$ and $(i,j-1)$ would each also calculate $(i-1,j-1)$, and so on at every level of recursion. This algorithm would run in $O(3^{\max(m,n)})$ time, much slower than $\Theta(mn)$. 