CS-E3190 Principles of Algorithmic Techniques

03. Dynamic programming – Tutorial Exercise

1. **Revisited Edit Distance.** The EDIT DISTANCE problem is as follows: the inputs are two strings a and b of lengths n and m, respectively. There are three different *edits*:

   1. Remove one letter
   2. Insert one letter
   3. Substitute one letter

   The *edit distance* $D(a, b)$ between a and b is the minimum number of edits required to turn string a into b. Our goal is to evaluate the distance between any pair of strings a, b.

   Suppose a string s is indexed as $s = (s_1, s_2, \ldots, s_n)$. For a given pair a, b let $d(i, j)$ be the edit distance between $a[1 : i]$ and $b[1 : j]$, where $s[1 : i] = (s_1, s_2, \ldots, s_i)$, and $s[1 : 0]$ is defined as the empty string ε. We define $d(i, j), \forall i \in \{0, 1, \ldots, n\}$ and $\forall j \in \{0, 1, \ldots, m\}$. Pseudocode for the *recursive* algorithm is given below.

   Algorithm 1: $d(n, m)$

   ```
   if n = 0 then
       return m;
   if m = 0 then
       return n;
   d_rm ← d(n − 1, m) + 1                    /* remove character       */
   d_in ← d(n, m − 1) + 1                    /* insert character       */
   d_sb ← d(n − 1, m − 1) + 1_(a_n ≠ b_m)    /* substitute character   */
   return min{d_rm, d_in, d_sb}
   ```

   The *indicator function* $\mathbb{1}_{(x \neq y)}$ takes a pair of characters x, y and evaluates to 1 if $x \neq y$ and to 0 otherwise. Your are asked to implement this recursion as a *dynamic programming* algorithm in the programming assignment. The correctness proof of these algorithms are largely the same. The goal of this problem is to develop our intuition about the problem as we gradually prove correctness. The programming assignment will have you think about runtimes.

   (a) *Base cases*. Prove that $d(0, j) = j$ and $d(i, 0) = i, \forall j \in \{0, 1, \ldots, m\}$ and $\forall i \in \{0, 1, \ldots, n\}$.

   Solution. Consider $d(0, j), \forall j \in \{1, \ldots, m\}$. By definition $d(0, j) = D(\varepsilon, b[1 : j])$. The minimum number of edits is clearly a sequence of j additions. So $d(0, j) = j$.

   Consider $d(i, 0), \forall i \in \{1, \ldots n\}$. The shortest sequence of edits that turns $a[1 : i]$ into the empty string is i successive deletions, and so $d(i, 0) = i$.

   (b) Let's go over one run of the algorithm before proceeding with the proof. Consider a table of $d(i, j)$-values for the input strings $a = LOONEY$ and $b = MOONS$ below. The values $d(i, j)$ of the base cases are already filled in. Complete the table by running the algorithm on input (a, b) and recording for each cell (i, j)[1]:

      i. The value $d(i, j)$, and a mark (e.g. underline) whenever $a[i] = b[j]$.

      ii. An arrow pointing to cell (i, j) from one of the parent cells $(i - 1, j)$, $(i, j - 1)$, or $(i - 1, j - 1)$ that minimizes $d(i, j)$. If two or more give the same minimum value, choose one arbitrarily.

[1] The table is similar to that in Erickson, p. 155; $d(i, j)$ is the entry of the i^{th} column and j^{th} row.

$j\backslash i$	ε		L		O		O		N		E		Y
ε	0	$\rightarrow$	1	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4	$\rightarrow$	5	$\rightarrow$	6
	$\downarrow$												
M	1												
	$\downarrow$												
O	2												
	$\downarrow$												
O	3												
	$\downarrow$												
N	4												
	$\downarrow$												
S	5												

Solution. Ties among minimizing cells are broken by rule $(i-1,j) \succ (i,j-1) \succ (i-1,j-1)$, i.e. we chose a down-arrow if available, else a right-arrow, and used diagonals only when they are the sole minimizer. The table as follows,

	ε		L		O		O		N		E		Y
ε	0	$\rightarrow$	1	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4	$\rightarrow$	5	$\rightarrow$	6
	$\downarrow$	$\searrow$											
M	1		1	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4	$\rightarrow$	5	$\rightarrow$	6
	$\downarrow$		$\downarrow$	$\searrow$									
O	2		2		**1**	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4	$\rightarrow$	5
	$\downarrow$		$\downarrow$		$\downarrow$	$\searrow$							
O	3		3		2		**1**	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4
	$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$	$\searrow$					
N	4		4		3		2		**1**	$\rightarrow$	2	$\rightarrow$	3
	$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$	$\searrow$			
S	5		5		4		3		2		2	$\rightarrow$	3

(c) The arrows in the table can be used to recover the appropriate sequences of edits.

(i) Give the implied sequence of edits for $LOON \rightarrow MOO$.

(ii) Give the implied sequence of edits for $LOONEY \rightarrow MOONS$.

Solution.

(i) Begin in the cell corresponding to $(LOON, MOO)$ and trace back the arrows. The sequence of edits is:

$$LOON \rightarrow_{sb} MOON \rightarrow_{rm} MOO$$

(ii) Starting from the bottom-right cell and tracing back we get the sequence of edits:

$$LOONEY \rightarrow_{sb} MOONEY \rightarrow_{sb} MOONSY \rightarrow_{rm} MOONS$$

(d) We proceed with the proof by proving an *optimal substructure* property. Let δ_k denote an edit, e.g. "insert an R after the 8^{th} character". We call successive edits $\delta_1\delta_2\cdots\delta_K$ an *edit sequence*. The edit distance $D(a,b)$ is exactly the length of the shortest edit sequence that converts a to b.

Without loss of generality, we can assume that edit sequences are ordered so that they apply left-to-right. As an example, the three edits associated with the table above are, in order:

a	L	O	O	N	E	Y
b	M	O	O	N	S	
δ	δ_1				δ_2	δ_3

Where δ_1, δ_2, and δ_3 are, respectively, sub $L \to M$, sub $E \to S$, and remove Y.

Fix arbitrary strings a and b, and let $\delta_1\delta_2 \cdots \delta_K$ be the shortest edit sequence converting a to b. Prove that the subsequence resulting from dropping the right-most edit δ_K, i.e. $\delta_1\delta_2 \cdots \delta_{K-1}$, is the *shortest edit distance* for the following subproblems:

$$\begin{cases} a[1:n-1] \to b[1:m] & \text{if } \delta_K = rm \\ a[1:n] \to b[1:m-1] & \text{if } \delta_K = in \\ a[1:n-1] \to b[1:m-1] & \text{if } \delta_K = sb. \end{cases}$$

Solution. Let $\delta_1\delta_2 \cdots \delta_K$ be the shortest sequence of edits converting a to b. Let a', and b' represent any one of the string pairs given above. By our assumption on the ordering of edits the subsequence $\delta_1\delta_2 \cdots \delta_{K-1}$ converts a' to b'. The right most edit δ_K completes whatever work remains in one edit. In particular δ_K

$$\begin{aligned} \text{removes } a_n &\quad \text{if} \quad \delta_K = rm \\ \text{inserts } b_m &\quad \text{if} \quad \delta_K = in \\ \text{substitutes } a_m \text{ with } b_m &\quad \text{if} \quad \delta_K = sb. \end{aligned}$$

Suppose that the subsequence $\delta_1 \cdots \delta_{K-1}$ is *not* a minimum-length edit sequence for converting a' to b'. Let's show this gives a contradiction.

The assumption implies that there exists a strictly shorter sequence $\delta'_1 \cdots \delta'_q$ converting a' to b', where $q < K-1$. We can use this sequence to convert a to b.

First use $\delta'_1 \cdots \delta'_{K-1}$ to convert a' to b'. Then use the last edit δ_K to "fix" the remainder. In other words, the sequence $\delta'_1 \cdots \delta'_q\delta_K$ is a length $q+1 < K$ sequence that converts a to b. This contradicts the assumption that $\delta_1 \cdots \delta_K$ is the shortest edit sequence for converting a to b. This concludes the proof of our claim.

(e) *Induction*. Suppose distances $d(i-1,j)$, $d(i,j-1)$, and $d(i-1,j-1)$ are correct. Prove that the algorithm computes the correct edit distance for $d(i,j)$.

Solution. The entry $d(i,j)$ is the edit distance of $a[1:i]$ to $b[1:j]$.

Let $\delta_1\delta_2 \cdots, \delta_K$ be a minimum-length edit sequence for $a[1:i]$ to $b[1:j]$. The last edit δ_K must be either a remove, an insert, or a substitution.

There are exactly three possible forms the subsequence $\delta_1 \cdots \delta_{K-1}$ can take, depending on whether δ_K is remove, insert, or substitute:

$$\begin{aligned} a[1:i-1]+a_i &\xrightarrow[\delta_1\cdots\delta_{K-1}]{} b[1:j]+a_i &&\xrightarrow[\delta_K]{} b[1:j] \\ a[1:i] &\xrightarrow[\delta_1\cdots\delta_{K-1}]{} b[1:j-1] &&\xrightarrow[\delta_K]{} b[1:j-1]+b_j \\ a[1:i-1]+a_i &\xrightarrow[\delta_1\cdots\delta_{K-1}]{} b[1:j-1]+a_i &&\xrightarrow[\delta_K]{} b[1:j-1]+b_j \end{aligned}$$

Let the minimum lengths of these sequences be d_{rm}, d_{in}, and d_{sb} respectively. These are by assumption exactly equal to $d(i-1,j)$, $d(i,j-1)$, and $d(i-1,j-1)$. As such the algorithm considers three edit sequences for converting a to b; these are the shortest sequence ending in deletion, addition, and substitution, respectively.

It follows that $d(i,j) = \min\{d_{rm}, d_{in}, d_{sb}\}$. By definition the edit distance $d(i,j)$ satisfies

$$d(i,j) \leq \min\{d_{rm}, d_{in}, d_{sb}\}$$

so the only way the two values are not equal is if $d(i,j) < \min\{d_{rm}, d_{in}, d_{sb}\}$. This however would contradict the optimality of one of $d(i-1,j)$, $d(i,j-1)$, or $d(i-1,j-1)$, because the sequence associated with $d(i,j)$ necessarily ends in either deletion, addition, or substitution.

(f) *Runtime*. Prove that the runtime and memory complextiy when a and b are of length n and m are at most $O(nm)$.

Solution. There are $(n+1) \times (m+1) = O(nm)$ entries in the table.

Computing $d(i,j)$ is $O(1)$ work because it requires looking up at most 3 cells, one $(a_i = b_j)$ check, 3 additions, and a minimum over 3 values.