Properties of Longest Common Subsequences

Notation and Definitions

We begin by providing a brief summary of notation introduced previously, as well as introducing some new notation for convenience:

Σ	an arbitrary alphabet
Σ^n	the set of all strings of length n over Σ
$\Sigma^\star$	the set of all strings over Σ
$\lvert s\rvert$	the length of the string s
$s[k]$	the kth character of the string s
$s[i, j)$	the string of characters in s from index i up to (exclusive) index j
$s[i\rangle$	the string of characters in s from index i to the end of the string
$s_1 s_2$	the concatenation of the two strings s_1 and s_2
$s \setminus s[k]$	the string s with the kth character removed
$s \sqsubseteq s_1$	the string s is a subsequence of the string s_1
$\mathrm{LCS}(s_1, s_2)$	the set of the longest common subsequences of two strings
$\mathcal{L}(s_1, s_2)$	the length of the longest common subsequence of two strings

Edit Operation Properties

An edit operation is a mapping that transforms one string into another. The four most common edit operations are insertion, deletion, substitution, and permutation. Given two strings, we aim to define the effects of performing one of these operations on one of the strings with respects to their longest common subsequence.

For the purpose of this section, we will only consider the LCS of two strings. Namely, we will consider the strings x and y, of lengths n and m respectively, over the arbitrary alphabet Σ. Suppose also that $\mathcal{L}(x, y)$ is already known. The properties derived in this section can be extended to the longest common subsequence of any number of strings, but this is beyond the scope of this section.

We start by defining important characteristics of subsequences.

Definition 1. Let w be a subsequence of string x. We say that a character C in w is ***critical*** if removing C from x would make w no longer a subsequence of x.

Definition 2. An ***affix*** is a character or string that is added to another string to construct a new string. If the affix is added to the beginning of the string, it is called a ***prefix***. If the affix is added to the end of the string, it is called a ***suffix***. If the affix is added at any other position in the string, it is called an ***infix***. Additionally, if part of the affix is added to the beginning of the string and the remainder is added to the end of the string, it is called a ***circumfix***.

Insertions

The insert operation affixes a character to a given string at a specified index. Let z be a string that is the result of affixing a character C to y at some index k.

$$z = y[0, k)Cy[k\rangle$$

We want to know $\mathcal{L}(x, z)$. We know that the LCS of x and z must be at least as long as the LCS of x and y, since z contains all of y. Therefore, any LCS of x and y will also be a subsequence of x and z. However, we have to take into account the possibility that the extra character added to z causes the length of the LCS to increase. This will happen if C is critical with respect to the LCS of x and z.

C is critical when we can affix C to an LCS of x and y to get a subsequence of x and z. Let x and y have an LCS w that allows for this. In this scenario, x and z have a common subsequence one longer than w, since the subsequence is w with C inserted to it. We cannot remove C from z, as we know that affixing C causes the subsequence to be longer then the LCS of x and y, and removing C from z gives us y. Thus, C is critical.

This common subsequence must also be an LCS, as any characters other than C that are part of the LCS would already be part of w, since those characters already existed in both x and y. Therefore, when C is critical with respected to the LCS of x and z, inserting C increases the LCS length by one. We formalize this relation in Property 1.

Property 1. The Character Insertion Property

For all $x \in \Sigma^n$, $y \in \Sigma^m$, and $C \in \Sigma$,

$$\mathcal{L}(x, z) = \mathcal{L}(x, y) + 1$$

if there exists $u \sqsubseteq y[0, k)$, $v \sqsubseteq y[k\rangle$ such that $uCv \sqsubseteq x, z$, $uv \in \text{LCS}(x, y)$.
If this is not the case, then

$$\mathcal{L}(x, z) = \mathcal{L}(x, y)$$

Deletions

The delete operation removes a character from a given string at a specified index. Let z be a string that is the result of removing a character from the string y at some index k.

$$z = y \setminus y[k]$$

We want to know $\mathcal{L}(x, z)$. Let C denote the character $y[k]$. We know that the LCS of x and z must be at most as long as the LCS of x and y, which is true when C is not a part of at least one LCS of x and y (in which case that LCS is also an LCS for x and z). However, this is not always the case. If C is critical for every LCS of x and y, then the length of the LCS will decrease. This relation is formally defined in Property 2.

Property 2. The Character Deletion Property

For all $x \in \Sigma^n$ and $y \in \Sigma^m$,

$$\mathcal{L}(x, z) = \mathcal{L}(x, y)$$

if there is some $w \in \text{LCS}(x, y)$ such that $w \sqsubseteq y \setminus y[k]$.
If this is not the case, then

$$\mathcal{L}(x, z) = \mathcal{L}(x, y) - 1$$

Substitutions

The substitute operation replaces the kth character of a given string with a different character C. Substitution is equivalent to deleting the kth character of the string and inserting C in the same position. Therefore, we can determine the effect of a substitution by applying Properties 2 and 1, in that order. Let z be a string that is the result of substitution the kth character of y with C.

$$z = y[0, k)Cy[k + 1\rangle$$

We consider all possible combinations of Properties 2 and 1.

1. There exists an LCS of x and y for which $y[k]$ is not critical, and inserting C to that LCS forms a subsequence of x and z.

   Under Property 2, deleting $y[k]$ does not change the LCS length. Under Property 1, inserting C increases the LCS length. Overall, the LCS increases in length by one.

2. There exists at least one LCS of x and y for which $y[k]$ is not critical, but for all of these LCS, inserting C does not form a subsequence of x and z.

   Under Property 2, deleting $y[k]$ does not change the LCS length. Under Property 1, inserting C also does not change the LCS length, so the overall length stays the same.

3. $y[k]$ is critical for all LCS of x and y, but inserting C into at least one of those LCS forms a subsequence of x and z.

   Under Property 2, deleting $y[k]$ decreases the LCS length. Under Property 1, inserting C increases the LCS length. Overall, the LCS length stays the same.

4. $y[k]$ is critical for all LCS of x and y, and for all of these LCS, inserting C does not form a subsequence of x and z.

   Under Property 2, deleting $y[k]$ decreases the LCS length. Under Property 1, inserting C does not change the LCS length. Overall, the LCS length decreases by one.

We summarize these cases in Property 3.

Property 3. The Character Substitution Property
For all $x \in \Sigma^n$, $y \in \Sigma^m$, and $C \in \Sigma$,

$$\mathcal{L}(x, z) = \mathcal{L}(x, y) + 1$$

if both of the following conditions are satisfied for some $w \in \text{LCS}(x, y)$:

1. $w \sqsubseteq y \setminus y[k]$
2. $uCv \sqsubseteq x, z$ where $u \sqsubseteq y[0, k)$, $v \sqsubseteq y[k\rangle$, and $w = uv$

If only one of the conditions are satisfied, then

$$\mathcal{L}(x, z) = \mathcal{L}(x, y)$$

If neither condition is satisfied, then

$$\mathcal{L}(x, z) = \mathcal{L}(x, y) - 1$$

Transpositions and Permutations

The transpose operation swaps two characters within a string. Let z be a string that is result of transposing the jth and kth characters of y.

$$z = y[0, j)y[k]y[j + 1, k)y[j]y[k + 1\rangle$$

Using Property 3, we can calculate the exact change to the LCS length for a specific j and k. More generally, we note that a substitution changes the LCS length by at most one, and thus two substitutions changes the LCS length by at most two.

Property 4. The Transposition Property

For all $x \in \Sigma^n$ and $y \in \Sigma^m$,

$$\mathcal{L}(x, z) = \mathcal{L}(x, y) + d$$

where d is an integer in the range $[-2, 2]$

Definition 3. A ***permutation*** of a sequence is a rearrangement of its elements. For example, the sequence $\left(a\,,b\,,c\right)$ has six permutations:

$$\left(a\,,b\,,c\right) \qquad \left(a\,,c\,,b\right) \qquad \left(b\,,a\,,c\right) \qquad \left(b\,,c\,,a\right) \qquad \left(c\,,a\,,b\right) \qquad \left(c\,,b\,,a\right)$$

Let z be a permutation of the characters in y obtained by arbitrarily reordering the characters of y. This permutation is equivalent to substituting each character that has changed in y with their permuted value. Since a single substitution can only increase the LCS length by 1, decrease it by 1, or have no effect, a permutation that changes the position of k characters changes the LCS by an integer in the range $[-k, k]$.

Property 5. The Permutation Property

For all strings x, y, their LCS will change by at most k if y has k of its characters permuted.

Derived Properties

Here we define properties obtained by applying earlier properties to special types of strings.

Concatenation and Slicing Let $\text{LCS}(x, y)$ be given for strings x, y. Let H be some other string that we concatenate to the front of both x and y. It is trivial to see that the LCS of Hx and Hy will be Hz for any LCS $z \in \text{LCS}(x, y)$. Conceptually, think of a process where the characters in Hz are read in one at a time, and erased from their first occurrence in both x and y. Clearly, every character of H will be erased, leaving x, y, and z. The same idea applies to concatenation at the end of the strings and to both the beginning and end of strings simultaneously. Formally,

Property 6. The Concatenation Property

For all $x \in \Sigma^n$, $y \in \Sigma^m$, and $H, T \in \Sigma^\star$,

$$\mathrm{LCS}(HxT, HyT) = \Big\{ HzT \mid z \in \mathrm{LCS}(x, y) \Big\}$$

From this, we can derive a property for slicing. Let u and v be two strings that share a common prefix H. Suppose we slice u and v in such a way that the prefix H is removed. Let these new strings be x and y. We can now prefix x and y with a different string P, whose length is the same as H. We can see that the length of the LCS of u and v is the same as that of Px and Py. Conceptually, think of a process where the characters in H are read in one at a time and replaced with the corresponding character in P. Clearly, the length of the LCS does not change since at all times, x and y are prefixed the same string. The same idea applies to suffixes and, by extension, circumfixes. Formally,

Property 7. The Slice-Concatenation Length Property

For all $HxT \in \Sigma^n$, $HyT \in \Sigma^m$, $P \in \Sigma^{|H|}$, and $S \in \Sigma^{|T|}$,

$$\mathcal{L}(HxT, HyT) = \mathcal{L}(PxS, PyS)$$

Complementations

Definition 4. For an arbitrary alphabet, define a relationship between pairs of symbols. Then each symbol in a pair is a ***complement*** of the other symbol in the pair. For alphabets with an odd number of characters, one symbol is paired with itself. For example, in the binary alphabet, 0 and 1 are complements of each other. The complement of a string is a string of equal length, where the character at index k in the original string is its complement in the complemented string.

We denote the complement of both individual characters and entire strings using the symbol $^{\mathsf{c}}$.

Let $\mathrm{LCS}(x, y)$ be given for strings x, y. From this, we also know $\mathcal{L}(x, y)$. We are interested in $\mathcal{L}(x^{\mathsf{c}}, y^{\mathsf{c}})$. This can be thought of as a series of substitutions, replacing each character in x and y with its complement. However, more precise bounds on the LCS length can be obtained by noting that complementing both strings is the same as swapping the labels for each symbol with the complement's symbol. Then, all of the LCS also swap labels, and so the complemented string's LCS are the complements of the original string's LCS.

Property 8. The Complementation Property

For all $x \in \Sigma^n$ and $y \in \Sigma^m$,

$$\mathrm{LCS}(x^{\mathsf{c}}, y^{\mathsf{c}}) = \Big\{ z^{\mathsf{c}} \mid z \in \mathrm{LCS}(x, y) \Big\}$$

Reversals

Definition 5. Given a string $x = \{c_1, c_2, \ldots, c_n\}$, we define the ***reverse*** of x to be

$$x^{\mathsf{R}} = \{c_n, c_{n-1}, \ldots, c_1\}$$

Let LCS(x, y) be given for strings x, y. From this, we also know $\mathcal{L}(x, y)$. We are interested in $\mathcal{L}(x^{\mathsf{R}}, y^{\mathsf{R}})$. This can be thought of as a series of transpositions, swapping each character in x and y with the character at the opposite end of the string. Again, more precise bounds can be obtained by recognizing that the characters in x^{R} and y^{R} have the same ordering as x and y when working from the end of the string to the front. Thus, the LCS of x^R and y^R, working from the end of the LCS to the front, is the same as the LCS of x and y. Since reading back to front is the same as the reverse, this means the LCS of x^R and y^R are the reverse strings of the LCS of x and y.

Property 9. The Reversal Property

For all $x \in \Sigma^n$ and $y \in \Sigma^m$,

$$\mathrm{LCS}(x^{\mathsf{R}}, y^{\mathsf{R}}) = \Big\{ z^{\mathsf{R}} \mid z \in \mathrm{LCS}(x, y) \Big\}$$

The Removed Character Lemma Let strings x, y be given, both with lengths at least n. We are interested in $\mathcal{L}(x \setminus x[k], y \setminus y[k]), k \leq n$. By Property 2, the length of the LCS can stay the same, decrease by one, or decrease by two. However, we will prove that if the length decreases by two, the kth character of x and y must be different.

Lemma 1. ***The Removed Character Lemma***
For all $x \in \Sigma^n$, $y \in \Sigma^m$, and $n, m \geq k$,

If $\mathcal{L}(x \setminus x[k], y \setminus y[k]) = \mathcal{L}(x, y) - 2$, then $x[k] \neq y[k]$.

Proof. Let x and y be strings of length at least k such that $\mathcal{L}(x\backslash x[k], y\backslash y[k]) = \mathcal{L}(x, y)-2$. Without loss of generality, delete $x[k]$ from x first. Since the LCS length can only decrease by one per deletion, deleting $x[k]$ must reduce the LCS length by one. Thus, by Property 2, $x[k]$ is critical for the LCS of x and y. For the same reason, $y[k]$ must be critical for the LCS of $x \setminus x[k]$ and y. But if $y[k]$ is critical in the LCS of $x \setminus x[k]$ and y, then it must exist in that LCS. Since $x[k]$ has already been deleted, $x[k] \neq y[k]$. □

Distribution Properties

Here we define properties that describe what is contained in the set of LCS for given strings.

Commutativity

Intuitively, the order of strings does not matter when finding an LCS. This is described formally in the following proof:

Property 10. The Commutative Property:

For all $x \in \Sigma^n$ and $y \in \Sigma^m$,

$$\text{LCS}(x, y) = \text{LCS}(y, x)$$

Proof. The position of characters within x and y does not change, so the set of possible subsequences for x and y stay the same. The set of LCS is a subset of the set of possible subsequences for both x and y, so it will also stay the same. □

Substring Existence

Every possible string up to length n exists as an LCS for at least one pair of strings of length n.

Property 11. The Substring Property:

For any string z of length k ($0 \leq k \leq n$), there exists $x, y \in \Sigma^n$ such that

$$z \in \text{LCS}(x, y)$$

Proof. Let z be a string of length at most n. Construct strings $x = z0\dots0$ and $y = z1\dots1$, where 0 and 1 are appended such that $|x| = |y| = n$. Since exactly $|z|$ characters in x and y are the same and any character in an LCS must be in both x and y, all LCS must be at most $|z|$ characters long. By construction, z is an LCS of x and y of length $|z|$, so $z \in \mathrm{LCS}(x, y)$. □

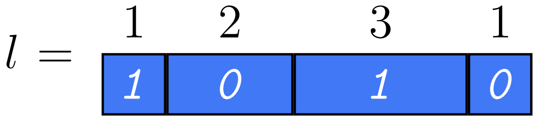
The $n - 1$ Distribution Property

Property 12. The $n - 1$ Distribution Property: Every binary string of length $n - 1$ appears as an LCS of a pair of binary strings of length n exactly $n(n + 1)$ times.

Note that this property only applies to pairs of binary strings. We will later expand this property to the more general case of any number of strings with an arbitrary alphabet. We will prove the binary property first, which requires new notation.

k-Block Formulation

To facilitate this proof, we introduce the k-block representation of a string. For any binary string, we can represent it as alternating blocks of contiguous 1s and 0s. For example, the string 1001110 is represented as follows:



We say that a binary string s of length $n - 1$ is represented by k_0 blocks of 0s and k_1 blocks of 1s. Furthermore, we denote the length of each block of 0s by $l^0_1, \dots, l^0_{k_0}$, and the length of each block of 1s by $l^1_1, \dots, l^1_{k_1}$. We denote the number of 0 characters as $L_0 = \sum_{i=1}^{k_0} l^0_i$, and the number of 1 characters as $L_1 = n - 1 - L_0 = \sum_{i=1}^{k_1} l^1_i$. Finally, we define p and q as follows:

$$p = \left\{ \begin{array}{ll} 0, & \text{if } s \text{ begins and ends with } 1 \\ 1, & \text{if } s \text{ begins and ends with opposite symbols} \\ 2, & \text{if } s \text{ begins and ends with } 0 \end{array} \right\} = 2 - q$$

This representation will make it easier to reason about and demonstrate several useful properties.

Placing a Character in a Single String

We are interested in finding how many times a string s of length $n-1$ appears as an LCS in all possible pairs of two strings a and b of length n. To do this, we work in reverse.

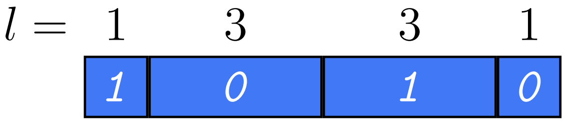
Let $s = t = u$. Clearly, s is an LCS of t and u. Our original problem is equivalent to counting the number of unique string pairs made by inserting a character to t and a character to u such that s is still an LCS of both strings. This will necessarily count all possible pairs of strings of length n whose longest common subsequence is s.

We note that, starting with two copies of s, adding a single character to only one of the strings does not immediately affect the LCS length. Thus, to count the number of unique pairs with s as an LCS, we can count the number of *unique* ways to add a character to one string, then look at the number of *unique* ways to add a character to the second string such that the LCS length does not change.

Lemma 2. *There are exactly $k_0 + L_1 - k_1 + p$ unique ways of inserting a 0 in s.*

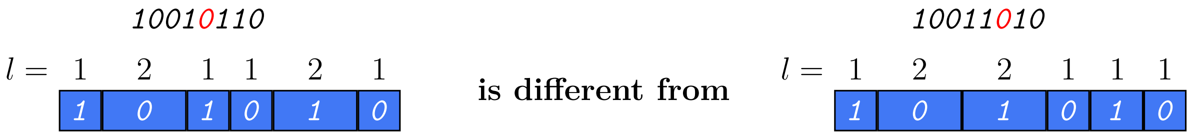
Proof. Firstly, note that where you place a 0 inside or next to any given 0-block does *not* matter. That is, there is only one unique way of placing a 0 inside or next to a given 0-block. For instance, the following placements inside 1001110 are identical:

10001110 10001110 10001110



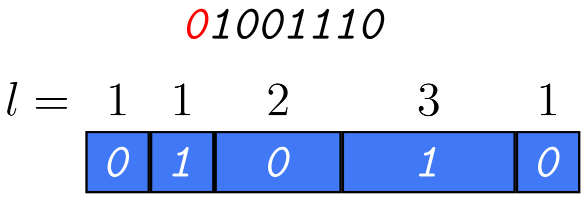
However, it does matter *which* 0-block you choose to place a 0 inside of or next to. Thus, there are exactly k_0 unique ways of placing a 0 inside or next to the 0-blocks in s.

Unlike with 0-blocks, it *does* matter where a 0 is placed inside a 1-block. For instance, these are unique placements within the same 1-block in 1001110:



For a 1-block of length l_i^1, there are $l_i^1 - 1$ ways of placing a 0 inside of it. Again, it also matters which 1-block you place a 0 inside of. Thus, there are exactly $\sum_{i=1}^{k_1}(l_i^1 - 1) = L_1 - k_1$ unique ways of placing a 0 inside the 1-blocks in s.

Since we have considered placing a 0 next to or inside of each 0-block, and all 1-blocks are surrounded by 0-blocks, we need not consider placing 0's next to 1-blocks, with one exception. If a 1-block occurs at the start of the string, we must consider placing a 0 to the left of it as a unique possibility, and if a 1-block occurs at the end of the string, we must consider placing a 0 to the right of it as a unique possibility.



The number of additional ways of placing a 0 because of this is exactly equal to q, by construction.

Combining all of these possible placements, there are exactly $k_0 + L_1 - k_1 + q$ unique ways of placing a 0 in s. □

We can use the same argument for the number of placements for a 1 in s to yield the following Lemma:

Lemma 3. *There are exactly $k_1 + L_0 - k_0 + p$ unique ways of inserting a 1 in s.*

We can combine these two Lemmas to count all possible ways to insert a character into s:

Lemma 4. *There are exactly $n + 1$ ways of placing a 0 or 1 in s.*

Proof. By Lemmas 2 and 3, there are $k_0 + L_1 - k_1 + q$ unique ways of placing a 0 in s and $k_1 + L_0 - k_0 + p$ unique ways of placing a 1 in s. Summing these yields

$$\begin{aligned} k_0 + L_1 - k_1 + q + k_1 + L_0 - k_0 + p &= L_1 + q + L_0 + p \\ &= L_1 + L_0 + 2 \\ &= (n-1) + 2 \\ &= n + 1 \end{aligned}$$

□

Inserting a Second Character

Let $s = t = u$. Without loss of generality, add a character to t. We now ask how many ways we can add a character to u such that $s \in \mathrm{LCS}(t, u)$.

Lemma 5. *There are exactly $n(n+1)$ ways of placing a 0 or a 1 in t and a 0 or a 1 in u such that their longest common subsequence is s.*

Proof. By Lemma 4, there are exactly $n+1$ unique strings that can be created by inserting a character into t. Since the LCS of two strings of length n can only be length n if the two strings are equal, any insertion into u will not affect the LCS length except for the insertion that results in the same string as t. Thus, there are $n+1-1=n$ ways of inserting the second character that result in a different string from t, so there are $n(n+1)$ ways in total to add a character to both strings without modifying the LCS length. $\square$

Generalization

We now generalize this distribution property for any number of strings with an arbitrary alphabet:

Lemma 6. *Across all possible d-tuples of strings of length n and alphabet size $\sigma \geq 2$, every possible string of length $n-1$ over that same alphabet each appears as an LCS exactly $(\sigma n - n + 1)^d - (\sigma n - n + 1)$ times in total.*

Proof. First, let us consider the case where we have only two strings but an alphabet Σ of size $\sigma \geq 2$. We use the same definitions as in the binary case, as well as one new definition. In the binary case, a *0*-block can only be next to *1*-blocks or the ends of the string, meaning we could count all insertions at the edges of a *1*-block as part of our count for an insertion inside of a *0*-block. In the general case, this is no longer true, as we could, for example, insert a *0* in between a *1*-block and a *2*-block. To account for this difference, we define a ***seam*** as a position between two blocks or at the ends of the string.

Let a character $a \in \Sigma$ be given. As before, there is exactly one way of placing a inside any given a-block, so there are k_a unique ways of placing a inside all a-blocks. Further, a can be placed inside the blocks of some other character b in

$$\sum_{i=1}^{k_b}(l_i^b - 1) = L_b - k_b$$

ways. Thus, a can be placed inside the blocks of all other characters $c \in \Sigma$ in

$$\sum_{c \in \Sigma \setminus \{a\}} \left(\sum_{i=1}^{k_c} l_i^c - 1 \right) = \sum_{c \in \Sigma \setminus \{a\}} (L_c - k_c)$$

ways.

However, so far we have considered only placing inside blocks, but not next to them at their seams. There are

$$\sum_{c \in \Sigma}(k_c) + 1$$

seams in total (number of blocks + 1).

Placing a at a seam is a unique placement if the seam does not border an a-block. So the total ways of placing a character a at seams is

$$\sum_{c \in \Sigma}(k_c) + 1 - 2k_a$$

Each block borders exactly two seams, and since no blocks of the same character may border each other, no overcounting occurs. Thus, for placing specifically a, there are

$$\begin{aligned}
k_a + \sum_{c \in \Sigma \setminus \{a\}} (L_c - k_c) + \sum_{c \in \Sigma} (k_c) + 1 - 2k_a &= k_a + \sum_{c \in \Sigma \setminus \{a\}} (L_c - k_c) + \sum_{c \in \Sigma \setminus \{a\}} (k_c) + 1 - k_a \\
&= k_a + \sum_{c \in \Sigma \setminus \{a\}} (L_c) + 1 - k_a \\
&= \sum_{c \in \Sigma \setminus \{a\}} (L_c) + 1 \\
&= n - L_a
\end{aligned}$$

ways. So, for placing *any* symbol, there are

$$\begin{aligned}
\sum_{c \in \Sigma} (n - L_c) &= \sum_{c \in \Sigma} (n) - \sum_{c \in \Sigma} (L_c) \\
&= \sum_{c \in \Sigma} (n) - (n - 1) \\
&= \sigma n - n + 1
\end{aligned}$$

ways.

As with the binary case, the second character inserted can be added anywhere so long as the insertion results in a different string from the first insertion. So there are

$$(\sigma n - n + 1)(\sigma n - n)$$

ways of placing a symbol each in two strings such that the LCS length remains the same.

To generalize this to $d \geq 2$ strings, the idea remains the same. The placement in the first string does not matter. The only restriction is that the placements in the following $d - 1$ strings cannot all be the same as the placement in the first string. Thus, there are

$$(\sigma n - n + 1)((\sigma n - n + 1)^{d-1} - 1) = (\sigma n - n + 1)^d - (\sigma n - n + 1) \tag{1}$$

ways of doing these placements in total.

For each of the $(\sigma n - n + 1)$ unique strings created from inserting into the first string, there are $(\sigma n - n + 1)^{d-1}$ unique combinations for the other $d - 1$ strings obtained by inserting a character into each of them. Since we do not want to count the combination where every resultant string is identical, we subtract that possibility from the count, resulting in Equation 1. □