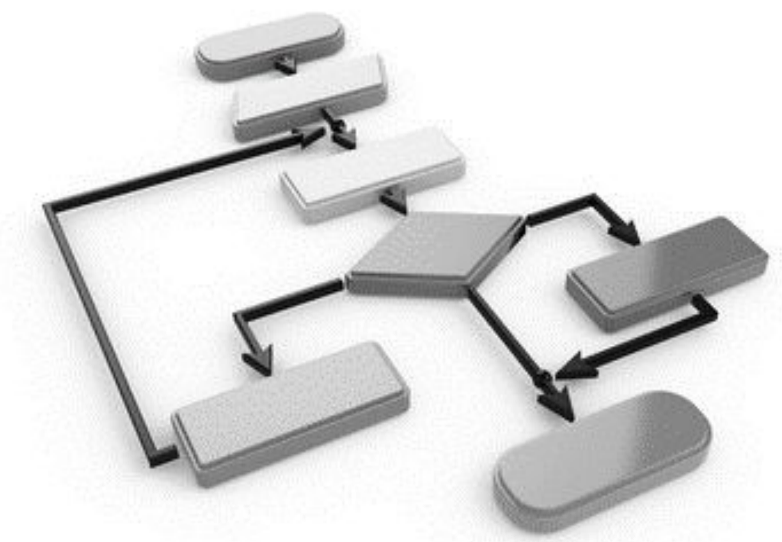


Dynamic Programming

CSE 301: Combinatorial Optimization

Longest Common Subsequence (LCS)

Given two sequences

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$
$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

find a maximum length common subsequence (LCS) of X and Y

Application: comparison of two DNA strings

Example

$X = \langle A, B, C, B, D, A, B \rangle$ $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$ $Y = \langle B, D, C, A, B, A \rangle$

Both $\langle B, C, B, A \rangle$ and $\langle B, D, A, B \rangle$ are longest common subsequences of X and Y (length = 4)

$\langle B, C, A \rangle$ is a common subsequence of X and Y, however it is not a LCS of X and Y

Brute-Force Solution

For every subsequence of X, check whether it's a subsequence of Y

There are 2^m subsequences of X to check

Each subsequence takes $\Theta(n)$ time to check

scan Y for first letter, from there scan for second, and so on

Running time: $\Theta(n2^m)$

LCS Recursive Solution

First we'll find the length of LCS. Later we'll modify the algorithm to find LCS itself.

Define X_i, Y_j to be the prefixes of X and Y of length *i* and *j* respectively

Define $c[i,j]$ to be the length of LCS of X_i and Y_j

Then the length of LCS of X and Y will be $c[m,n]$

LCS Recursive Solution

We start with $i = j = 0$ (empty substrings of x and y)

Since X_0 and Y_0 are empty strings, their LCS is always empty (i.e. $c[0,0] = 0$)

LCS of empty string and any other string is empty, so for every i and j: $c[0, j] = c[i,0] = 0$

LCS Recursive Solution

When we calculate *c[i,j],* we consider two cases:

First case: *x[i]=y[j]*:

one more symbol in strings X and Y matches, so the length of LCS X_i and Y_j equals to the length of LCS of smaller strings X_{i-1} and Y_{i-1} , plus 1

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ & \end{cases}$$

LCS Recursive Solution

Second case: *$x[i] \neq y[j]$*

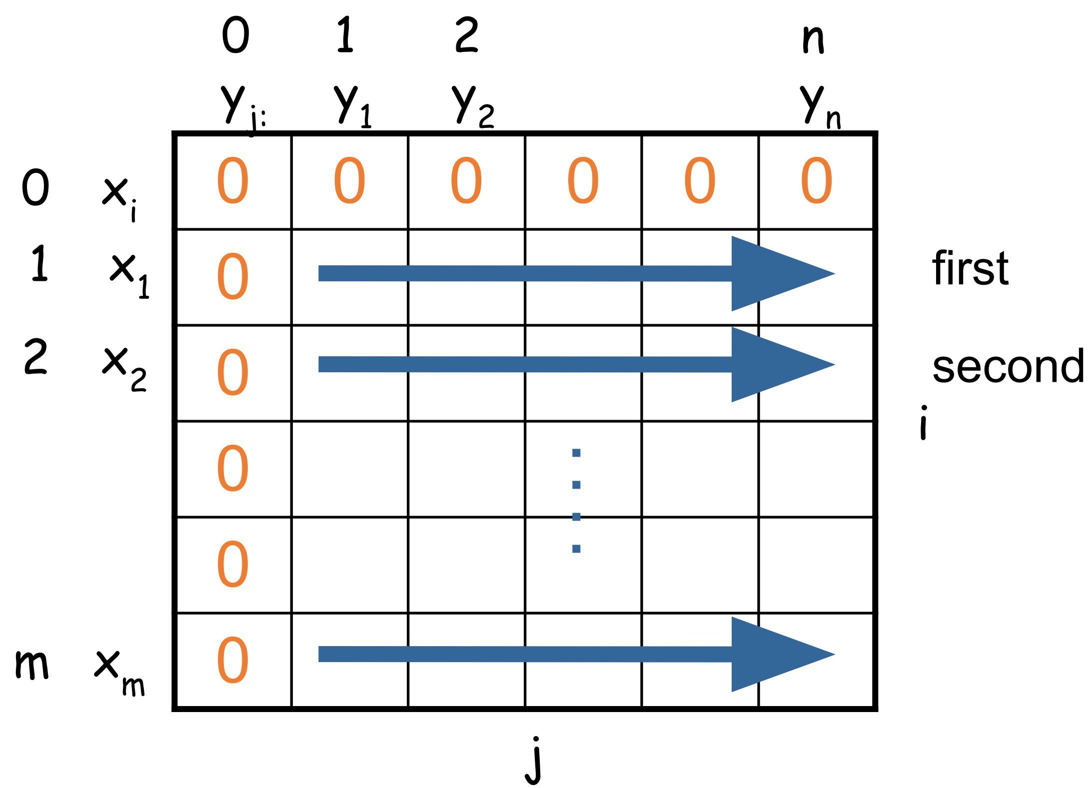
As symbols don't match, our solution is not improved, and the length of LCS(X_i , Y_j) is the same as before, *i.e.,* maximum of LCS(X_i, Y_{j-1}) and LCS(X_{i-1},Y_j)

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

Why not just take the length of LCS(X_{i-1}, Y_{j-1}) ?

Computing the Length of the LCS

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$



Additional Information

$$c[i, j] = \begin{cases} 0 & \text{if } i,j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$

b & c:

		0	1	2	3		n
		y_j:	A	C	D		F
0	x_i	0	0	0	0	0	0
1	A	0					
2	B	0			c[i-1,j]		
3	C	0		c[i,j-1]	←		
		0					
m	D	0					

i

j

A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$
  b[i, j] = " " ↖
- Else, if $c[i - 1, j] \geq c[i, j-1]$
  b[i, j] = " ↑ "
  else
  b[i, j] = " ← "

LCS-LENGTH(X, Y, m, n)

```
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
        do if xi = yj
            then c[i, j] ← c[i - 1, j - 1] + 1
                b[i, j ] ← "    "
            else if c[i - 1, j] ≥ c[i, j - 1]
                then c[i, j] ← c[i - 1, j]
                    b[i, j] ← "↑"
                else c[i, j] ← c[i, j - 1]
                    b[i, j] ← "←"
return c and b
```

The length of the LCS if one of the sequences is empty is zero

Case 1: $x_i = y_j$

Case 2: $x_i \neq y_j$

Running time: Θ (mn)

Example

$X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$

If $x_i = y_j$

$b[i, j] =$ " ↖ "

Else if $c[i-1, j] \geq c[i, j-1]$

$b[i, j] =$ " ↑ "

else

$b[i, j] =$ " ← "

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

4. Constructing a LCS

Start at b[m, n] and follow the arrows

When we encounter a "↖" in b[i, j] $\Rightarrow x_i = y_j$ is an element of the LCS

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

PRINT-LCS(b, X, i, j)

```
if i = 0 or j = 0
   then return
if b[i, j] = "↖"
     then PRINT-LCS(b, X, i - 1, j - 1)
              print x_i
elseif b[i, j] = "↑"
              then PRINT-LCS(b, X, i - 1, j)
              else PRINT-LCS(b, X, i, j - 1)
```

Running time: $\Theta(m + n)$

Initial call: PRINT-LCS(b, X, *length*[X], *length*[Y])

Compute Edit (Levenshtein) Distance : ED

Given two strings: X and Y, how can you convert X to Y via the <u>minimum</u> number of *edit operations* in X where an edit operation is*:* insert, substitute, or delete.

E.g. X = "**heater**", Y = "**speak**"

Minimum sequence of edits required to convert X to Y:

- substitute **h** by **s**: **<u>h</u>eater** -> **<u>s</u>eater**
- insert **p** after **s**: **seater** -> **s<u>p</u>eater**

(skip next two positions of X+Y, i.e., **e** and **a**, since they match)

- substitute **t** by **k**: **spea<u>t</u>er** -> **spea<u>k</u>er**
- delete **e**: **speak<u>e</u>r** -> **speakr**
- delete **r**: **speak<u>r</u>** -> **speak**

Total 5 edit operations are needed; so ED = 5

ED Recursive Solution

Define X_i, Y_j to be the prefixes of X and Y of length *i* and *j* respectively

Define $c[i,j]$ to be the edit distance between X_i and Y_j

Let $|X| = m$ and $|Y| = n$.

Then the ED of X and Y will be $c[m,n]$

ED Recursive Solution

We start with $i = j = 0$ (empty substrings of x and y).

Since X_0 and Y_0 are both empty strings, their ED is zero (i.e. $c[0,0] = 0$)

ED of any i-length string X_i and the empty string (""), is i because we need i deletions to convert X_i to ""; so $c[i,0] = i$

ED of "" and any j-length string Y_j, is j because we need j insertions to convert "" to Y_j; so $c[0, j] = j$

ED Recursive Solution

When we calculate *c[i,j],* we consider two cases:

First case: *x[i]=y[j]*:

one more symbol in strings X and Y matches, so the ED of X_i and Y_j equals to the ED of smaller strings X_{i-1} and Y_{j-1}

$$c[i,j] = \begin{cases} c[i-1, j-1] & \text{if } x[i] = y[j], \\ & \end{cases}$$

ED Recursive Solution

Second case: *x[i] ≠ y[j]*

As symbols don't match, we have to either (i) substitute *x[i]* by *y[j],* (ii) delete *x[i],* or (iii) insert *y[j]*. Among these 3 operations, we will apply that operation which yield minimum value of *c[i][j]*.

Cost of operation:

(i) Substitute: *c[i][j] = c[i-1][j-1]+1*

E.g. ED(hea$\underset{i}{\underline{\text{t}}}$, spea$\underset{j}{\underline{\text{k}}}$) = ED(he$\underset{i-1}{\underline{\text{a}}}$, spe$\underset{j-1}{\underline{\text{a}}}$) + 1 = 2+1 = 3

(ii) Delete *x[i]*: *c[i][j] = c[i-1][j]+1*

E.g. ED(breath$\underset{i}{\underline{\text{e}}}$, breadt$\underset{j}{\underline{\text{h}}}$) = ED(breat$\underset{i-1}{\underline{\text{h}}}$, breadt$\underset{j}{\underline{\text{h}}}$)+1 = 1+1 = 2

(iii) Insert *y[j]*: *c[i][j] = c[i][j-1]+1*

E.g. ED(po$\underset{i}{\underline{\text{t}}}$, yok$\underset{j}{\underline{\text{e}}}$) = ED(po$\underset{i}{\underline{\text{t}}}$, yo$\underset{j-1}{\underline{\text{k}}}$)+1 = 2+1 = 3

ED Recursive Solution

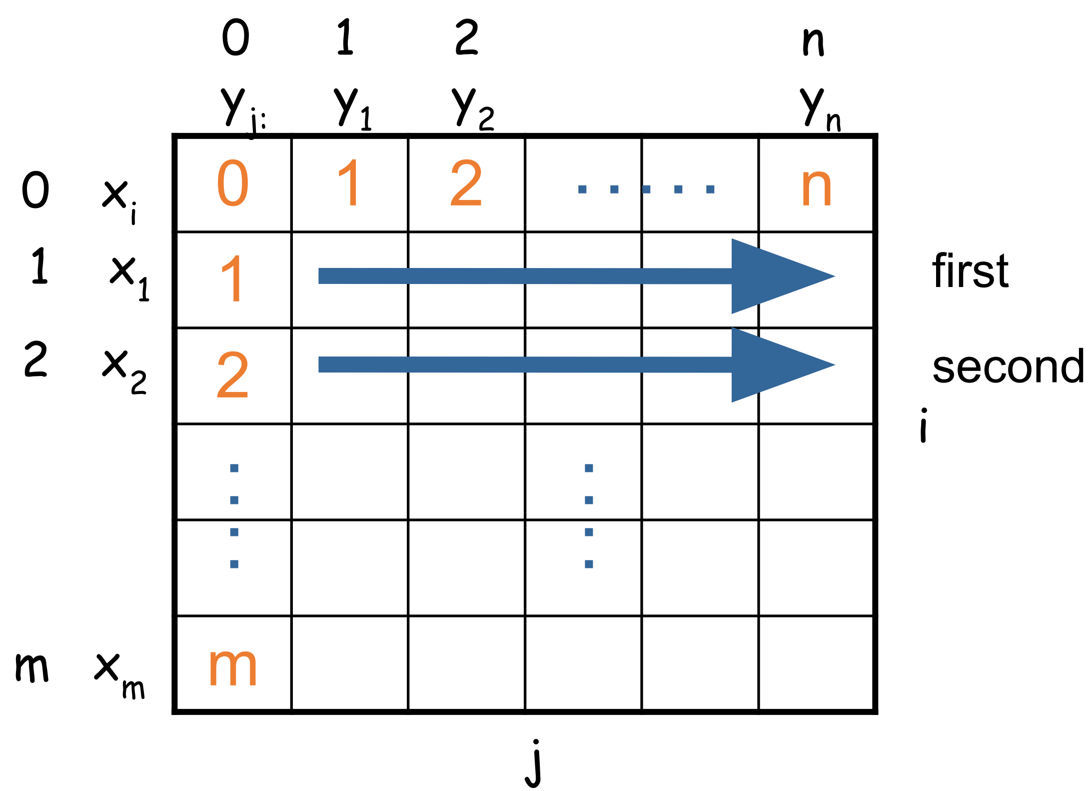
Second case: *x[i] ≠ y[j]*

As symbols don't match, we have to either (i) substitute x[i] by y[j], (ii) delete x[i], or (iii) insert y[j]. Among these 3 operations, we will apply that operation which yield minimum value of *c[i][j]*.

$$c[i,j] = \begin{cases} c[i-1,j-1] & \text{if } x[i] = y[j], \\ \min(c[i-1][j-1], c[i-1,j], c[i,j-1]) + 1 & \text{otherwise} \end{cases}$$

Computing ED

$$c[i, j] = \begin{cases} i, & \text{if } j = 0 \\ j & \text{if } i = 0 \\ c[i-1, j-1] & \text{if } x_i = y_j \\ \min(c[i-1,j-1], c[i-1, j], c[i][j-1])+1, & \text{if } x_i \neq y_j \end{cases}$$



Simulation

$$c[i, j] = \begin{cases} i, & \text{if } j = 0 \\ j, & \text{if } i = 0 \\ c[i-1, j-1], & \text{if } x_i = y_j \\ \min(c[i-1,j-1], c[i-1, j], c[i][j-1])+1, & \text{if } x_i \neq y_j \end{cases}$$

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1					
2	e	↑ 2					
3	a	↑ 3					
4	t	↑ 4					
5	e	↑ 5					
6	r	↑ 6					

Legends:

← Insert y_j

↑ Delete x_i

↖ Substitute x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Simulation

$$c[i, j] = \begin{cases} i, & \text{if } j = 0 \\ j, & \text{if } i = 0 \\ c[i-1, j-1], & \text{if } x_i = y_j \\ \min(c[i-1, j-1], c[i-1, j], c[i][j-1]) + 1, & \text{if } x_i \neq y_j \end{cases}$$

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1				
2	e	↑ 2					
3	a	↑ 3					
4	t	↑ 4					
5	e	↑ 5					
6	r	↑ 6					

Legends:

← Insert y_j

↑ Delete x_i

↖ Substitute x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Simulation

$$c[i, j] = \begin{cases} i, & \text{if } j = 0 \\ j, & \text{if } i = 0 \\ c[i-1, j-1], & \text{if } x_i = y_j \\ \min(c[i-1,j-1], c[i-1, j], c[i][j-1])+1, & \text{if } x_i \neq y_j \end{cases}$$

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2					
3	a	↑ 3					
4	t	↑ 4					
5	e	↑ 5					
6	r	↑ 6					

Legends:

← Insert y_j

↑ Delete x_i

↖ Substitute x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Simulation

$$c[i, j] = \begin{cases} i, & \text{if } j = 0 \\ j, & \text{if } i = 0 \\ c[i-1, j-1], & \text{if } x_i = y_j \\ \min(c[i-1, j-1], c[i-1, j], c[i][j-1]) + 1, & \text{if } x_i \neq y_j \end{cases}$$

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3					
4	t	↑ 4					
5	e	↑ 5					
6	r	↑ 6					

Legends:

← Insert y_j

↑ Delete x_i

↖ Substitute x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Simulation

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	← 3	← 4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	← 3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Sequence of edit operations needed to convert “heater” to “speak”:

1. Insert ‘s’: _heater -> <u>s</u>heater

Simulation

		0 y_j	1 s	2 p	3 e	4 a	5 k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	←3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Sequence of edit operations needed to convert “heater” to “speak”:

1. Insert ‘s’: _heater -> sheater
2. Replace ‘h’ by ‘p’: sheater -> speater

Simulation

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	←3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Sequence of edit operations needed to convert “heater” to “speak”:

1. Insert ‘s’: _heater -> <u>s</u>heater
2. Replace ‘h’ by ‘p’: s<u>h</u>eater -> s<u>p</u>eater

Simulation

		0 y_j	1 s	2 p	3 e	4 a	5 k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	←3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when x_i == y_j)

Sequence of edit operations needed to convert "heater" to "speak":

1. Insert 's': _heater -> <u>s</u>heater
2. Replace 'h' by 'p': s<u>h</u>eater -> s<u>p</u>eater
3. Delete 't': spea<u>t</u>er -> speaer

Simulation

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	←3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when x_i == y_j)

Sequence of edit operations needed to convert “heater” to “speak”:

1. Insert ‘s’: _heater -> sheater
2. Replace ‘h’ by ‘p’: sheater -> speater
3. Delete ‘t’: speater -> speaer
4. Delete ‘e’:speaer -> spear

Simulation

		0	1	2	3	4	5
		y_j	s	p	e	a	k
0	x_i	0	← 1	← 2	← 3	← 4	← 5
1	h	↑ 1	↖ 1	↖ 2	↖ 3	↖ 4	↖ 5
2	e	↑ 2	↖ 2	↖ 2	⇖ 2	←3	←4
3	a	↑ 3	↖ 3	↖ 3	↖ 3	⇖ 2	←3
4	t	↑ 4	↖ 4	↖ 4	↖ 4	↑ 3	↖ 3
5	e	↑ 5	↖ 5	↖ 5	⇖ 4	↑ 4	↖ 4
6	r	↑ 6	↖ 6	↖ 6	↑ 5	↖ 5	↖ 5

Legends:

← Insert y_j in X at position i

↑ Delete x_i from X

↖ Replace x_i by y_j

⇖ no edit operation (done when $x_i == y_j$)

Sequence of edit operations needed to convert "heater" to "speak":

1. Insert 's': _heater -> <u>s</u>heater
2. Replace 'h' by 'p': s<u>h</u>eater -> s<u>p</u>eater
3. Delete 't': spea<u>t</u>er -> speaer
4. Delete 'e':spea<u>e</u>r -> spear
5. Replace 'r' by 'k': spea<u>r</u> -> spea<u>k</u>