

# Edit Distances

Yufei Tao

KAIST

May 6, 2013

Early search engines supported keyword search in an “exact” manner, namely, reporting websites that contain the query strings precisely. Not surprisingly, many queries in practice contain typing errors. Instead of returning websites containing mis-typed words (which is a bad idea in most cases), modern search engines are able to automatically correct many of these errors, and proceed with the corrected query strings.

In a nutshell, spelling correction works as follows. Given a (mis-spelled) query string  $q$ , we want to look up in a dictionary for the strings  $s$  that are similar to  $q$ . To carry out this approach, apparently, we must first find a good way to quantify the similarity of  $q$  and  $s$ .

In this lecture, we will discuss such a similarity metric, called the **edit distance**.

Given two strings  $s$  and  $t$ , the edit distance  $edit(s, t)$  is the **smallest** number of following **edit operations** to turn  $s$  into  $t$ :

- **Insertion**: adding a character
- **Deletion**: removing a character
- **Substitution**: replace a character with another one.

### Example

$edit(\text{abode}, \text{blog}) = 4$  because

- We can change abode into blog by 4 operations:
  - 1 delete a  $\Rightarrow$  bode
  - 2 insert l after b  $\Rightarrow$  blode
  - 3 delete d  $\Rightarrow$  bloe.
  - 4 substitute e with g  $\Rightarrow$  blog
- Impossible to do so with at most 3 operations.

## Lemma

$$\textit{edit}(s, t) = \textit{edit}(t, s).$$

Namely, the edit distance is **symmetric**.

Next, we discuss the algorithm for computing  $edit(s, t)$ .

Given a string  $\sigma$ , we denote by  $\sigma[i]$  the  $i$ -th character of  $\sigma$ , for each  $i \in [1, |\sigma|]$ . Also, given  $1 \leq i \leq |\sigma|$ , denote by  $\sigma[1..i]$  as the substring of  $\sigma$  starting from  $\sigma[1]$  and ending at  $\sigma[i]$ . Specially, define  $\sigma[1..0]$  to be the empty string.

Let  $m = |s|$  and  $n = |t|$ .

### Lemma

- If  $m = 0$ , then  $edit(s, t) = n$ .
- If  $n = 0$ , then  $edit(s, t) = m$ .
- If  $m > 0$ ,  $n > 0$ , and  $s[m] = t[n]$ , then  $edit(s, t)$  is

$$\min \begin{cases} 1 + edit(s, t[1..n-1]) \\ 1 + edit(s[1..m-1], t) \\ edit(s[1..m-1], t[1..n-1]) \end{cases}$$

- If  $m > 0$ ,  $n > 0$ , and  $s[m] \neq t[n]$ , then  $edit(s, t)$  is

$$\min \begin{cases} 1 + edit(s, t[1..n-1]) \\ 1 + edit(s[1..m-1], t) \\ 1 + edit(s[1..m-1], t[1..n-1]) \end{cases}$$

Equipped with the lemma of the previous slide, next we will learn a **dynamic programming** algorithm for computing  $edit(s, t)$ .

The goal of the algorithm is to fill up a two-dimensional array  $A$ . Specifically,  $A$  has  $m + 1$  rows and  $n + 1$  columns. Let us label the rows as  $0, \dots, m$ , and the columns as  $0, \dots, n$ . The cell  $A[i, j]$  at row  $i$  and column  $j$  equals:

$$A[i, j] = edit(s[1..i], t[1..j]).$$



## Example

The matrix  $A$  for  $s = \text{abode}$  and  $t = \text{blog}$ :

	0	1	2	3	4
0	0	1	2	3	4
1	1	1	2	3	4
2	2	1	2	3	4
3	3	2	2	2	3
4	4	3	3	3	3
5	5	4	4	4	4

We use the following strategy to fill up  $A$ :

- 1 First set the first row, and first column.
- 2 Then, fill the rest of  $A$  row by row, starting from row 2, and then row 3, 4, ... At each row, fill the cells from left to right.

By the lemma in Slide 7, we can fill in cell  $A[i, j]$  the value:

$$\min \begin{cases} 1 + A[i, j - 1] \\ 1 + A[i - 1, j] \\ A[i - 1, j - 1] \text{ if } s[i] = t[j], \text{ or } 1 + A[i - 1, j - 1] \text{ otherwise} \end{cases}$$

In any case, we need  $A[i, j - 1]$ ,  $A[i - 1, j]$ , and  $A[i - 1, j - 1]$ , all of which are already available.

## Lemma

The dynamic programming algorithm on the previous slide runs in  $O(mn)$  time.

Next, we extend the algorithm to compute also an **editing path**, namely, a sequence of edit operations to turn  $s$  into  $t$ .

### Definition

A cell  $A[i, j]$  is said to be **determined** by

- $A[i, j - 1]$ , if  $A[i, j] = 1 + A[i, j - 1]$
- $A[i - 1, j]$ , if  $A[i, j] = 1 + A[i - 1, j]$
- $A[i - 1, j - 1]$ , otherwise.

## Definition

Given strings  $s, t$ , a set  $T$  of integer pairs is called a **trace** if we can order the pairs of  $T$  as  $(a_1, b_1), (a_2, b_2), \dots, (a_z, b_z)$  for some  $z \geq 1$  such that  $1 \leq a_1 < a_2 < \dots < a_z \leq |s|$ , and  $1 \leq b_1 < b_2 < \dots < b_z \leq |t|$ .

## Example

Given  $s = \text{abode}$  and  $t = \text{blog}$ , then:

- $\{(2, 1), (3, 3)\}$  is a trace.
- $\{(2, 3), (3, 1)\}$  is **not** a trace.
- $\{(2, 3), (3, 5)\}$  is **not** a trace.

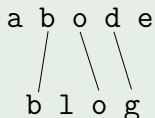
Given a trace  $T$ , we can imagine a line connecting  $s[a_i]$  to  $t[b_i]$ , for each  $i \in [1, z]$  where  $z = |T|$ .

If a character of  $s$  or  $t$  has a line, we say that it is **traced**; otherwise, it is **non-traced**.

Let  $t[x_1], t[x_2], \dots, t[x_z]$  be traced characters. We refer to each of  $t[1..x_1 - 1]$ ,  $t[x_1 + 1..x_2 - 1]$ ,  $\dots$ ,  $t[x_{z-1} + 1..x_z - 1]$ ,  $t[x_z + 1..|t|]$  as a **non-traced substring** of  $t$ .

### Example

Given a trace  $\{(2, 1), (3, 3), (4, 4)\}$  on  $s = \text{abode}$  and  $t = \text{blog}$ , then:



$t$  has 4 non-traced substrings:  $\emptyset$ ,  $l$ ,  $o$ , and  $g$ .

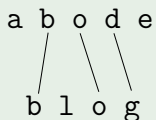
Given a trace  $T$  on  $s$  and  $t$ , we can convert  $s$  to  $t$  by the following steps:

- Copy  $s$  to  $\sigma$ .
- For each  $i \in [1, z]$  such that  $a_i \neq b_i$ , substitute  $\sigma[a_i]$  with  $t[b_i]$ .
- For each non-traced character in  $s$ , delete the corresponding character from  $\sigma$ .
- Insert the first non-traced substring of  $t$  at the beginning of  $\sigma$ .
- Append the last non-traced substring of  $t$  at the end of  $\sigma$ .
- For each non-traced  $\pi$  substring of  $t$  between traced characters  $c$  and  $c'$ , insert  $\pi$  in  $\sigma$  between the corresponding characters of  $c$  and  $c'$ .



## Example

a b o d e  
b l o g



We convert  $s$  to  $t$  by:

- Substitute  $d$  with  $g$ .
- Delete  $a$  and  $e$ .
- Insert  $l$ .

We can obtain an optimal **editing path** by finding a trace as follows

**algorithm** trace( $A$ )

/\* assume that  $A$  has been filled \*/

1. initialize a set  $T = \emptyset$
2.  $i = m, j = n$
3. **while**  $i \neq 0$  and  $j \neq 0$
4.     suppose that  $A[i, j]$  is determined by  $A[i', j']$
5.     **if**  $i' = i - 1$  and  $j' = j - 1$  **then**
6.         add  $(i, j)$  to  $T$
11.      $i = i', j = j'$
12. **return**  $T$

## Example

The matrix  $A$  for  $s = \text{abode}$  and  $t = \text{blog}$ :

	0	1	2	3	4
0	0	1	2	3	4
1	1	1	2	3	4
2	2	1	2	3	4
3	3	2	2	2	3
4	4	3	3	3	3
5	5	4	4	4	4

Thus,  $T = \{(4, 4), (3, 3), (2, 1)\}$

## Remark

Note that the editing path of  $T$  (see Slide 17) is different from the one in Slide 4. In general, there can be **multiple** optimal editing paths.