Introduction to Algorithms and Data Structures

Lecture 20: Edit distance (via Dynamic Programming)

Mary Cryan

School of Informatics
University of Edinburgh

Edit distance

Our setting is strings over some input alphabet. We want to measure the edit distance between two given strings s, t over that alphabet.

We have three operations on strings - insertion, deletion, and substitution.

Examples:

- DNA or RNA strings over their 4-character alphabet: for example, ”AATCCGCTAG” versus ”AAACCCTTAG”.
- Words from a natural language - for example, “kitten” versus “sitting”.

k	i	t	t	**e**	n	-
s	i	t	t	**i**	n	**g**

(3 operations: 2 “substitutions” and 1 “insertion”)

Sequence Alignment

We often talk about possible alignments of two (or more) sequences. For example, here are two competing alignments for a given pair of DNA sequences:

A	C	C	**G**	**G**	T	**A**	T	**C**	**C**	T	A	G	G	A	C
A	C	C	**T**	**A**	T	**C**	T	-	-	T	A	G	G	A	C

A	C	C	**G**	**G**	T	A	T	C	**C**	T	A	G	G	A	C
A	C	C	-	-	T	A	T	C	**T**	T	A	G	G	A	C

An alignment of two sequences $s \in \Sigma^m, t \in \Sigma^n$ is any padding (with some $-$ insertions) s' of s, and t' of t such that

$$
\begin{aligned}
&|s'| = |t'| \\
&(s'_i \neq -) \vee (t'_i \neq -) \quad \text{for all } 1 \leq i \leq |s'|
\end{aligned}
$$

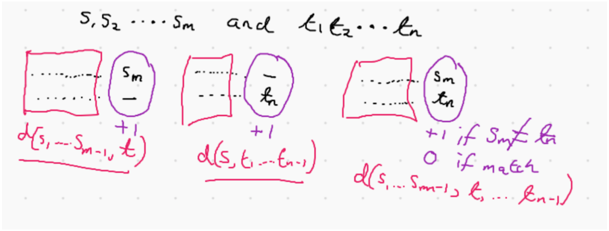
The score of an aligment is the total number of insertions ($s'_i \in \Sigma$ with $t'_i = -$), deletions ($s'_i = -$ with $t'_i \in \Sigma$) and substitutions ($s'_i \neq t'_i, s'_i \in \Sigma, t'_i \in \Sigma$).

Edit distance

The edit distance $d(s,t)$ between two strings $s, t \in \Sigma^*$ is the minimum number of operations possible for an alignment of those strings.

We start with strings over the alphabet Σ.
How to align these? We don't know.

But we do know there are only 3 ways the final column can be arranged!



$s_1 s_2 \cdots s_m$ and $t_1 t_2 \cdots t_n$
s_m
$-$
$+1$
$d(s_1 \ldots s_{m-1}, t)$
$-$
t_n
$+1$
$d(s, t_1 \ldots t_{n-1})$
s_m
t_n
$+1$ if $s_m \neq t_n$
0 if match
$d(s_1 \ldots s_{m-1}, t_1 \ldots t_{n-1})$

And the "best possible" for each of these 3 possibilities is another "edit distance" problem for an input that is slightly smaller.

Edit distance

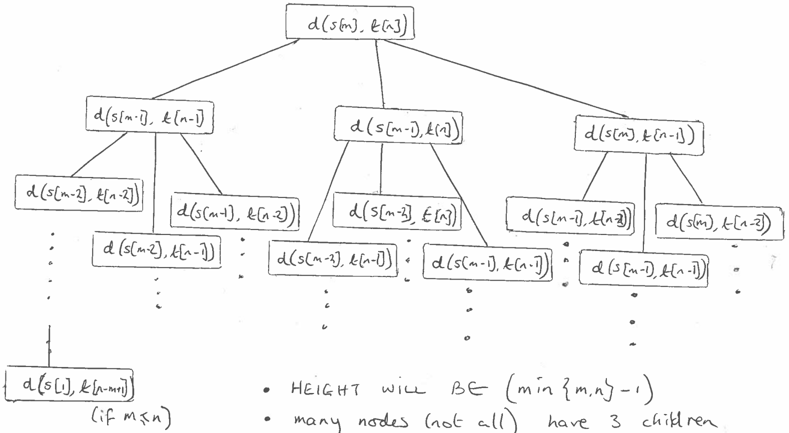
We get a natural recurrence for the edit distance for $s = s[1\ldots m], t = t[1\ldots n]$:

$$d(s[1\ldots m], t[1\ldots n]) = \begin{cases} m & \text{if } n = 0 \\ n & \text{if } m = 0 \\ d(s[1\ldots m-1], t[1\ldots n-1]) & \text{if } s_m = t_n \\ \begin{array}{c} 1 + \min\{d(s[1\ldots m-1], t[1\ldots n-1]) \\ d(s[1\ldots m-1], t[1\ldots n]) \\ d(s[1\ldots m], t[1\ldots n-1])\} \end{array} & \text{if } s_m \neq t_n \end{cases}$$

Justification?

Whatever the best alignment is, its right column must *either* be a substitution, *or* a deletion, *or* an insertion.

A recursive implementation?



- HEIGHT WILL BE $(\min\{m,n\} - 1)$
- many nodes (not all) have 3 children
- #-of-leaves grows similar to $3^{\min\{m,n\}-1}$!

Recursion tree is exponential in size ... however there are at most $m \cdot n$ sub-problems that can arise! So we are in a situation where DP can be exploited

Dynamic programming implementation

We will need a table/array of size $(m+1) \cdot (n+1)$, let the table be d.

- Entry $d[i, j]$ is intended to store the value of $d(s[1 \ldots i]), t[1 \ldots j])$ (when we have computed it).
- We need to fill the table in a careful order - need to be sure that $d[i-1, j-1], d[i-1, j]$ and $d[i, j-1]$ have *already* been computed before we exploit the recurrence to compute $d[i, j]$.

We will also keep a table/array called a which will store values $0, 1, 2, 3$ to mark whether the optimum for $s[1 \ldots i], t[1 \ldots j]$ ended in a match (0), a substitution (1), an insertion (2) or a deletion (3).

The a table will help us reconstruct the actual (best) alignment that achieves the edit distance.

(the 0/1/2/3 are just quaternary “flags” and their values are not significant)

Dynamic programming implementation

Algorithm Edit-Distance($s[1 \dots m], t[1 \dots n]$)

```
1.  for $i \leftarrow 0$ to $m$
2.      $d[i,0] \leftarrow i$, $a[i,0] \leftarrow 3$
3.  for $j \leftarrow 0$ to $n$
4.      $d[0,j] \leftarrow j$, $a[0,j] \leftarrow 2$
5.  for $i \leftarrow 1$ to $m$ do
6.      for $j \leftarrow 1$ to $n$ do
7.          if $s_i = t_j$ then
8.              $d[i,j] \leftarrow d[i-1,j-1]$
9.              $a[i,j] \leftarrow 0$
10.         else
11.             $d[i,j] \leftarrow 1 + \min\{d[i,j-1], d[i-1,j], d[i-1,j-1]\}$
12.             if $d[i,j] = d[i-1,j-1] + 1$ then $a[i,j] \leftarrow 1$
13.             else if $d[i,j] = d[i,j-1] + 1$ then $a[i,j] \leftarrow 2$
14.             else $a[i,j] \leftarrow 3$
```

Reconstructing the best alignment

We use the information in the a table to fill two arrays b, c.

- b will hold the padded version of s (the s'), in reverse
- c will hold the padded version of t (the t'), in reverse
- we will build b, c by "working-back" through the table a (having started at $a[m, n]$ ($i \leftarrow m, j \leftarrow n$).
- At each step, we will check whether $a[i, j]$ is either
  - 0/1 In this case we insert character s_i into b, and character t_j into c, then decrement both i and j
  - 2 In this case we insert character '$-$' into b, and character t_j into c, then decrement j (but not i)
  - 3 In this case we insert character s_i into b, and character '$-$' into c, then decrement i (but not j)
- At some point either i or j will hit 0, then we need to "finish off" b and c with a "run of insertions" or a "run of deletions".
- This results with the exact alignment stored in b and c, in reverse order. We then can print out in reverse.

Running time

It is not too hard to show that the running time for Edit-Distance is the same as the space of its primary tables, ie, $\Theta(mn)$.

Reading Materials

Edit Distance: None of our texts cover edit distance in exactly the way we have done - however, each of them has a section on sequence alignment:

[KT] Chapter 6.6

[CLRS] 14.4

[Roughgarden] 17.1