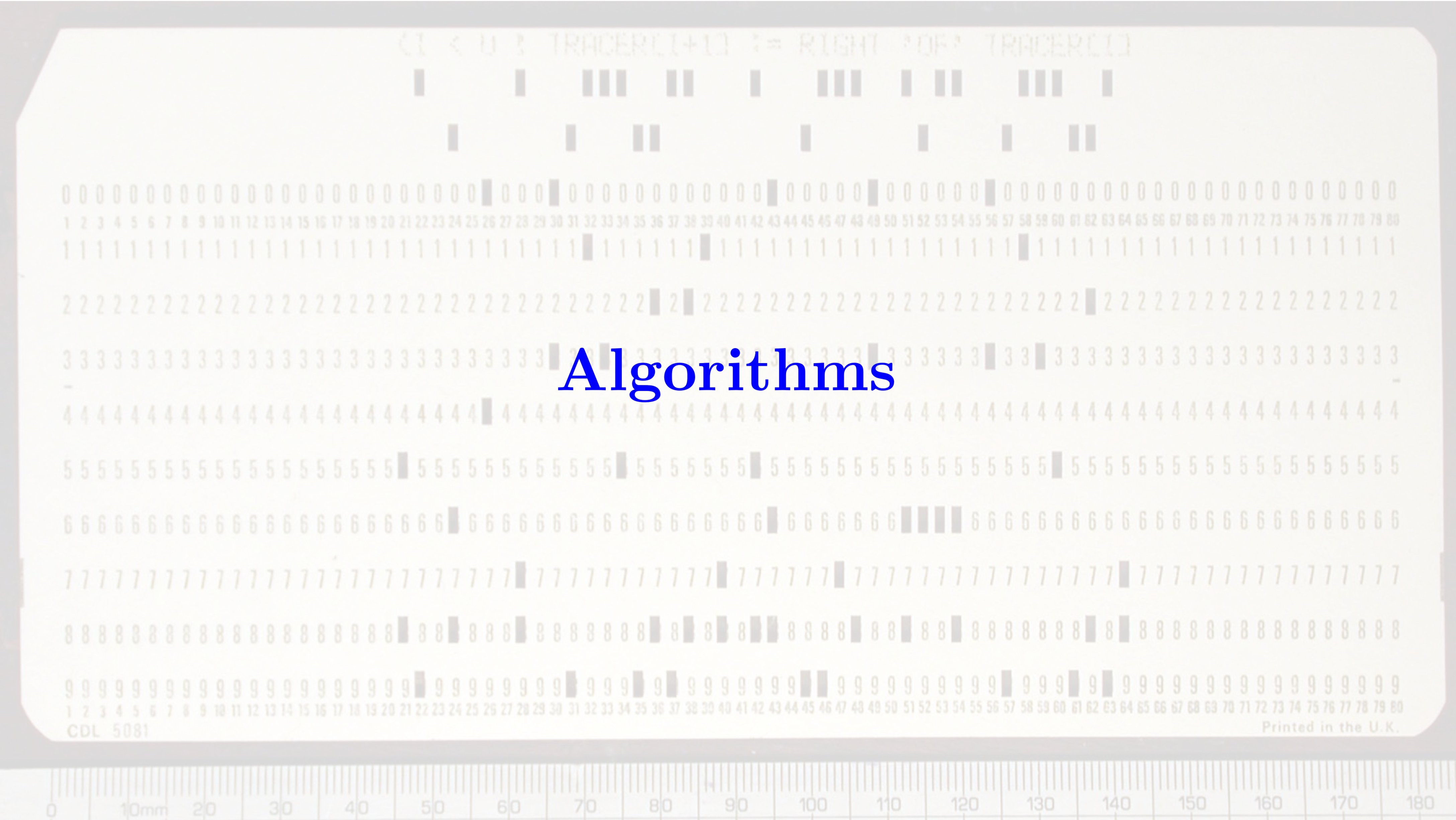
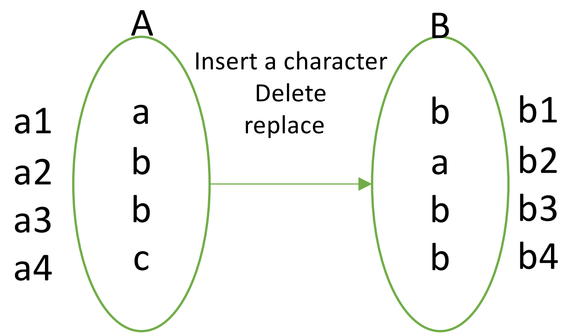
Algorithms



Edit Distance



Given two strings : $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_m$

Compute the minimum number of edits to change A into B

Type of edits allowed: *insert, delete, replace.*

Example. $A = abbc,\ B = babb$

$$abbc \to bbc \to babc \to babb \ (3 \ edits)$$

$$abbc \to babbc \to babb \ (2 \ edits)$$

Impossible with 1 *edit. So, edit distance* $= 2$.

Useful when we want to archive many versions of a file.
It is more efficient to store the differences than entire files.

What is the edit distance between "One fine day" and "On the final day" ?

Seems difficult for large strings.

Edit Distance

Given two strings : $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_m$,

define $A_i = a_1 \cdots a_i$ and $B_j = b_1, \cdots b_j$

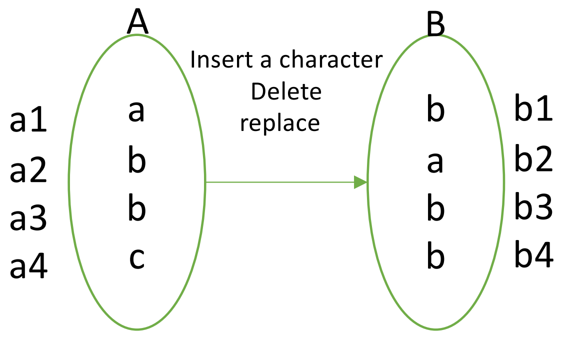
$C(i, j)$ = edit distance between A_i to B_j

Example. $A =$ *"string"*, $B =$ *"saturn"*.

$C(3,5) =$ *edit distance between "str" and "satur"*

We seek $C(n, m)$.

Edit Distance



Consider what happens to a_n in an edit sequence:

1. deleted
2. mapped to b_m
3. mapped to something other than b_m

Which cases are the following?

Example. $A = abbc$, $B = babb$

$$abbc \to bbc \to babc \to babb$$

$$abbc \to babbc \to babb$$

Edit Distance

Compute the minimum number of edits to change A into B

Case 1: a_n is deleted.

$$A = a_1 a_2 \cdots a_{n-1} a_n$$

$$A_{n-1} = a_1 a_2 \cdots a_{n-1} \times$$

$$\downarrow$$

$$B = b_1 b_2 \cdots b_{m-1} b_m$$

We delete a_n and then convert A_{n-1} to B_m.

$C(n, m) = C(n-1, m) + 1$

Edit Distance

Case 2: a_n is mapped to b_m.

$$\begin{array}{rl} A = & a_1 a_2 \cdots a_{n-1} \; a_n \\ & \quad\;\; \downarrow \qquad\;\; \downarrow \\ B = & b_1 b_2 \cdots b_{m-1} \; b_m \end{array}$$

<u>Replace a_n by b_m if $a_n \neq b_m$</u>. Then convert A_{n-1} to B_{m-1}.

$$C(n, m) = C(n-1, m-1) + \lambda(n, m)$$

where, $\lambda(i, j) = 1$ if $a_i \neq b_j$ and 0 otherwise.

Edit Distance

Case 3: a_n is mapped to something other than b_m

$$A = a_1 a_2 \cdots a_{n-1} a_n$$

$$\Downarrow$$

$$B = b_1 b_2 \cdots b_k \cdots b_{m-1} b_m$$

Inserted character

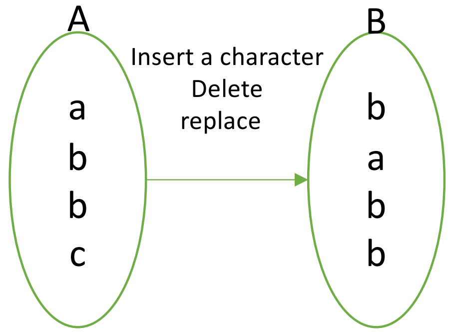
Insert b_m at the end. Then convert A_n to B_{m-1}.

$$C(n, m) = C(n, m-1) + 1$$

Edit Distance

It follows that:

$$C(n,m) = \min \begin{cases} C(n-1,m)+1 \\ C(n-1,m-1)+\lambda(n,m) \\ C(n,m-1)+1 \end{cases}$$



a b b c — c deleted
a b b — abb mapped to abb
b a a b — b was inserted

Recurrence relation. $C(i,j) = \min \{ C(i,j-1)+1,$
$C(i-1,j)+1,$
$C(i-1,j-1)+\lambda(i,j) \}$

$$\text{for } i,j \geq 1$$

$$C(i,0) = i,\ C(0,j) = j \text{ for any } i,j \geq 0$$

This give us a recursive algorithm. But as before, we can do a bottom-up computation and store computed results.

Edit Distance

$$
\begin{aligned}
C(i,j) = min \{ \; & C(i,j-1)+1, \\
& C(i-1,j)+1, \\
& C(i-1,j-1)+\lambda(i,j) \; \}
\end{aligned}
$$

$$
\begin{tikzcd}
C(i-1,j-1) \arrow[dr] & C(i,j-1) \arrow[d] \\
C(i-1,j) \arrow[r] & C(i,j)
\end{tikzcd}
$$

Observation. $C(i,j)$ can be computed in constant time if the other three are known.

Edit Distance

Implementation: Use a 2D matrix C, where $C[i,j]$ stores $C(i,j)$

$A = SPAKE$
$B = PARK$

B

A

		P	A	R	K
	0	1	2	3	4
S	1	?			
P	2				
A	3				
K	4				
E	5				

$$C[1,1] = min\ \{C[1,0]+1,\ C[0,1]+1,\ C[0,0]+\lambda(1,1)\}$$

		P	A	R	K
	0	1	2	3	4
S	1	1			
P	2				
A	3				
K	4				
E	5				

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	?			
A	3				
K	4				
E	5				

		P	A	R	K
	0	1	2	3	4
S	1	1	2	3	4
P	2	1	2	3	4
A	3	2	1	2	3
K	4	3	2	2	2
E	5	4	3	3	3

Edit Distance

Running time for computing the edit distance between two strings of lengths m and n respectively: $O(mn)$.

Exercise. Implement the algorithm in your favorite programming language.

Sequence Alignment

Given two sequences (or strings), we would like "align" them so that the number of mismatches is *small* by introducing *few* "gaps" in the sequences.

Example. The strings "occurrance" and "occurrence" can be aligned as:

`o-currance` `occurrence`	OR	`o-curr-ance` `occurre-nce`
one gap, one mismatch		*three gaps, no mismatch*

In this case, it is not clear which alignment is better.

To resolve this, we let applications specify the *costs* of mismatches and gaps.

This problem comes up in computational biology where is it used for aligning gene sequences.

```
A C T C G C A A T A T G C T A G G C C A G C
A C T _ _ _ _ T T A T G C T A T G C _ _ G C
```

Sequence Alignment

Gap penalty. Each gap costs δ, where $\delta > 0$.

Mismatch cost. The cost of a mismatch depends on the things are mismatched.

α_{pq} : cost of lining up p with q

$\alpha_{pp} = 0$ for all p

Example.

```
o-currance
occurrence
```

cost: $\delta + \alpha_{ae}$

OR

```
o-curr-ance
occurre-nce
```

cost: 3δ

Which one is preferred depends on whether $\alpha_{ae} < 2\delta$.

Question. Given two sequences $x_1\, x_2 \cdots x_m$ and $y_1\, y_2 \cdots y_n$ and the gap and mismatch costs for every pair of characters, how do we find the alignment that minimizes the total cost?

Homework

Sequence Alignment

Let $\text{OPT}(i, j)$ = the minimum cost of aligning $x_1\, x_2 \cdots x_i$ and $y_1\, y_2 \cdots y_j$.

We seek $\text{OPT}(m, n)$.

In the optimal alignment, at least one of the following happens:

- x_i is matched with y_j *cost:* $\alpha_{x_i y_j}$
- x_i is not matched *cost:* δ
- y_j is not matched *cost:* δ

This implies that:

$$\text{OPT}(i, j) = \min\{\, \alpha_{x_i y_j} + \text{OPT}(i-1, j-1),\ \delta + \text{OPT}(i-1, j),\ \delta + \text{OPT}(i, j-1)\,\}$$

We can use memoization to convert this into an efficient algorithm.

Running time? $O(mn)$ Space? $O(mn)$

Check https://www.youtube.com/watch?v=bLgkCPm5btY&list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6&index=48