Tuesday, March 8, 2022 10:49 AM

More Dynamic Programming

Step 1: Define subproblems.
Step 2: Define Recursive Formula to solve subproblems
Step 3: Memoization & Evaluation order → Iterative Algo.

Ex 1: (Edit Distance) (post-cursor & longest common subseq.)

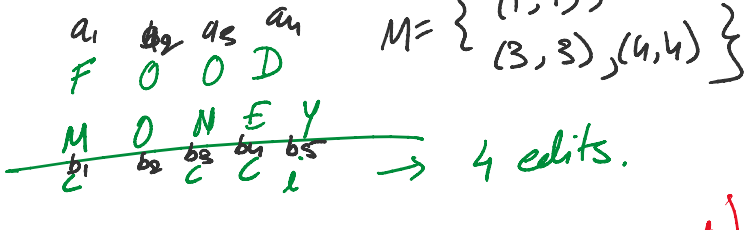
Given two sequences $A = a_1 a_2 \cdots a_m$
$B = b_1 b_2 \cdots b_n$ $\quad m \neq n$.

Edit distance (A, B) = # symbol/char edits needed to go from A to B.

Edits: insert / delete / change

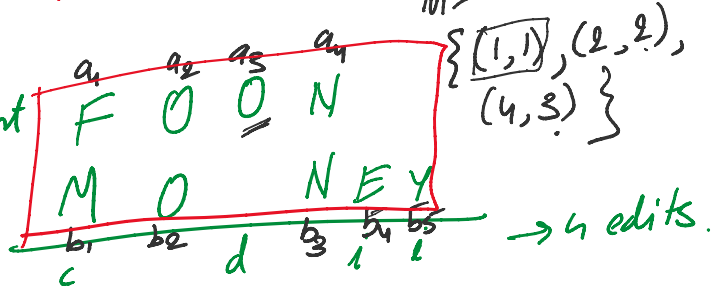
Find Minimum edit distance bet^n A & B.

Eg. A = FOOD
B = MONEY



(Intuitively: Find LCS, and align those characters. For the rest first try to change then insert / delete.

A = FOON
B = MONEY



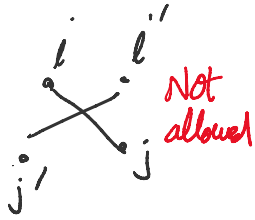
Rephrase: Find "alignment" of A & B with ... "mismatches".

Rephrase: Find "alignment" of A & B with smallest number of "mismatches."

Alignment $M = \{(i,j) \mid i \in \{1 \dots m\},\ j \in \{1 \dots n\}\}$

s.t. $\forall (i,j), (i',j') \in M$, $i \neq i' \Leftrightarrow j \neq j'$

AND if $i < i'$ then $j < j'$ ⇒ Non-crossing.



cost. (M) = # mismatched pairs. $\forall (i,j) \in M$ s.t. $a_i \neq b_j$

+ # Missing indices.

Mismatch (i,j) → change a_i to b_j

Missing $i \in \{1..m\}$ → delete a_i

Missing $j \in \{1..n\}$ → insert b_j.

Step 1: Define Subproblems.

Intuition:

$A = \alpha\ a_m$

$B = \beta\ b_n$

$\alpha = a_1 \dots a_{m-1}$ $\quad \beta = b_1 \dots b_{n-1}$

Case I

$\alpha\ a_m$

$\beta\ b_n$

$(m,n) \in M$

Case II

$\alpha\ a_m$

$\beta\ b_n$

(Delete a_m)

m will be a missing index in A.

Case III

$\alpha\ a_m$

$\beta\ b_n$

(insert b_n)

n will be a missing index in B.

$C(i,j)$ = Min. edit distance between $a_1 \dots a_i$ and $b_1 \dots b_j$

$i = 0 \ldots m$ & $j = 0 \ldots n$

→ Answer : $C(m, n)$

→ Base cases : $i = 0$ or $j = 0$

$C(0, j) = j$, $C(i, 0) = i$

→ Recursive Formula : $C(i, j)$

$\alpha = a_1 \cdots a_{i-1}$
$\beta = b_1 \cdots b_{j-1}$

case I

$\alpha\ a_i$
$\beta\ b_j$

$\rightarrow$ $C(i-1, j-1) + 1$ if $a_i \neq b_j$

$\rightarrow$ $C(i-1, j-1)$ if $a_i = b_j$

case II

delete a_i

$\alpha\ a_i$
$\beta\ b_j$

$\rightarrow$ $C(i-1, j) + 1$

$a_1 \ldots a_{i-1}$
$b_1 \ldots b_j$

case III

insert b_j

$\alpha\ a_i$
$\beta\ b_j$

$\rightarrow$ $C(i, j-1) + 1$

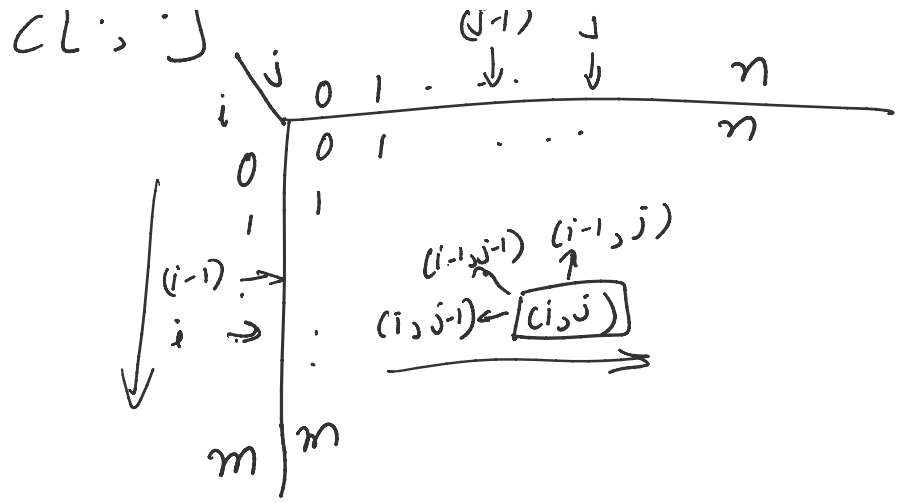
$$C(i,j) = \min \begin{cases} C(i-1, j-1) + 1, \\ C(i-1, j) + 1, \\ C(i, j-1) + 1 \end{cases} \quad \text{if } a_i \neq b_j$$

$$\min \begin{cases} C(i-1, j-1), \\ C(i-1, j) + 1, \\ C(i, j-1) + 1 \end{cases} \quad \text{if } a_i = b_j$$

→ Memoization

$C[\cdot, \cdot]$

$i \backslash j$	0	1	$\cdots$	$(j-1)$	j	$\cdots$	n



→ Evaluation Order : $i = 1$ to m
For each i, $j = 1$ to n

→ Psuedo code.

EditDist (A=a1 ... am, B=b1 ... bn)

{

for $i = 0$ to m
$C[i,0] = i$ → $O(1)$ } $O(m)$

for $j = 0$ to n
$C[0,j] = j$ → $O(1)$ } $O(n)$

for $i = 1$ to m do

for $j = 1$ to n do

if $a_i \neq b_j$
Then $C[i,j] = \min\{C[i-1,j-1]+1, C[i-1,j]+1, C[i,j-1]+1\}$

Else $C[i,j] = \min\{C[i-1,j-1], C[i-1,j]+1, C[i,j-1]+1\}$

$O(1)$

$O(n)$

$m \times O(n) = mn$ (constant c) $= cmn = O(mn)$

Return $C[m,n]$;

→ Analysis: $O(mn)$ time
$O(mn)$ subproblems. each takes $O(1)$ time.

$O(mn)$ space.

↳ If only want to find min. editdistance then can improve to $O(n)$.
Store previous row. prev[0..n] → (i-1)th row
curr[0..n] → ith row

→ Output the edits.

```
OutputAns(i, j)   // outputs set M.
{ if i=0 or j=0 then return.
  if (ai ≠ bj & C[i,j] = C[i-1, j-1]+1) OR
     (ai = bj & C[i,j] = C[i-1, j-1])
  then OutputAns(i-1, j-1); Print "(i,j)";
  Else if C[i,j] = C[i-1, j] + 1
     then OutputAns(i-1, j); // i is missing in M
                             hence ai should be deleted
  Else // C[i,j] = C[i, j-1] + 1
     then OutputAns(i, j-1); // j is missing in M
                             hence bj should be
                                 inserted.
}
```

$O(m+n)$ because every recursive call either reduces i or j.