What is the running time of the following algorithm:

Consider computing $f(x, y)$ by recursive function + memoization.

$$f(x, y) = \min(x, y) \sum_{i=1}^{x} x \ast f(x + y - i, i - 1),$$

$f(0, y) = y$  
$f(x, 0) = x$.

The resulting algorithm when computing $f(n, n)$ would take:

(a) $O(n^2)$  
(b) $O(n^3)$  
(c) $O(2^n)$  
(d) $O(n^n)$  
(e) The function is ill defined - it can not be computed.
What is the running time of the following algorithm:

Consider computing $f(x, y)$ by recursive function + memoization.

$$f(x, y) = \sum_{i=1}^{\min(x,y)} x \times f(x + y - i, i - 1),$$

$$f(0, y) = y \quad f(x, 0) = x.$$

The resulting algorithm when computing $f(n, n)$ would take:

(a) $O(n^2)$
(b) $O(n^3)$
(c) $O(2^n)$
(d) $O(n^n)$
(e) The function is ill defined - it can not be computed.
Recipe for Dynamic Programming

1. Develop a recursive backtracking style algorithm $A$ for given problem.

2. Identify structure of subproblems generated by $A$ on an instance $I$ of size $n$
   2.1 Estimate number of different subproblems generated as a function of $n$. Is it polynomial or exponential in $n$?
   2.2 If the number of problems is "small" (polynomial) then they typically have some "clean" structure.

3. Rewrite subproblems in a compact fashion.

4. Rewrite recursive algorithm in terms of notation for subproblems.

5. Convert to iterative algorithm by bottom up evaluation in an appropriate order.

6. Optimize further with data structures and/or additional ideas.
Edit Distance and Sequence Alignment
Spell Checking Problem

Given a string “exponen” that is not in the dictionary, how should a spell checker suggest a nearby string?
Spell Checking Problem

Given a string “exponen” that is not in the dictionary, how should a spell checker suggest a nearby string?

What does nearness mean?

**Question:** Given two strings $x_1x_2\ldots x_n$ and $y_1y_2\ldots y_m$ what is a distance between them?
Spell Checking Problem

Given a string “exponen” that is not in the dictionary, how should a spell checker suggest a nearby string?

What does nearness mean?

Question: Given two strings $x_1 x_2 \ldots x_n$ and $y_1 y_2 \ldots y_m$ what is a distance between them?

Edit Distance: minimum number of “edits” to transform $x$ into $y$. 
**Definition**

*Edit distance* between two words $X$ and $Y$ is the number of letter insertions, letter deletions and letter substitutions required to obtain $Y$ from $X$.

**Example**

The edit distance between *FOOD* and *MONEY* is at least 4:

$$
\text{FOOD} \rightarrow \text{MOOD} \rightarrow \text{MONOD} \rightarrow \text{MONED} \rightarrow \text{MONEY}
$$

4 “edits”
Alignment
Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

Mismatched letters mean substitution (same as deletion followed by an insertion).
Alignment
Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

Formally, an alignment is a set $M$ of pairs $(i, j)$ such that each index appears at most once, and there is no “crossing”: $i < i'$ and $i$ is matched to $j$ implies $i'$ is matched to $j' > j$. In the above example, this is $M = \{(1, 1), (2, 2), (3, 3), (4, 5)\}$. 
Alignment
Place words one on top of the other, with gaps in the first word indicating insertions, and gaps in the second word indicating deletions.

\[
\begin{align*}
F & \quad O \quad O \quad D \\
M & \quad O \quad N \quad E \quad Y
\end{align*}
\]

Formally, an alignment is a set \( M \) of pairs \((i, j)\) such that each index appears at most once, and there is no “crossing”: \( i < i' \) and \( i \) is matched to \( j \) implies \( i' \) is matched to \( j' > j \). In the above example, this is \( M = \{(1, 1), (2, 2), (3, 3), (4, 5)\} \). Cost of an alignment is the number of mismatched columns plus number of unmatched indices in both strings.
Problem
Given two words, find the edit distance between them, i.e., an alignment of smallest cost.
Applications

- Spell-checkers and Dictionaries
- Unix diff
- DNA sequence alignment ... but, we need a new metric
Definition
For two strings $X$ and $Y$, the cost of alignment $M$ is

- **[Gap penalty]** For each gap in the alignment, we incur a cost $\delta$.

- **[Mismatch cost]** For each pair $p$ and $q$ that have been matched in $M$, we incur cost $\alpha_{pq}$; typically $\alpha_{pp} = 0$. 

Edit distance is a special case when $\delta = \alpha_{pq} = 1$. 
Definition
For two strings \(X\) and \(Y\), the cost of alignment \(M\) is

- **[Gap penalty]** For each gap in the alignment, we incur a cost \(\delta\).
- **[Mismatch cost]** For each pair \(p\) and \(q\) that have been matched in \(M\), we incur cost \(\alpha_{pq}\); typically \(\alpha_{pp} = 0\).

Edit distance is special case when \(\delta = \alpha_{pq} = 1\).
Edit distance as alignment
Example

(1) \[
\begin{array}{cccccccc}
\rightarrow & o & c & u & r & r & a & n & c & e \\
\rightarrow & o & c & c & u & r & r & e & n & c & e \\
\end{array}
\]

Cost = \( \delta + \alpha_{ae} \)

Alternative:

(2) \[
\begin{array}{cccccccc}
o & c & u & r & r & a & n & c & e \\
o & c & c & u & r & r & e & n & c & e \\
\end{array}
\]

Cost = \( 3\delta \)

Or a really stupid solution (delete string, insert other string):

(3) \[
\begin{array}{cccccccc}
o & c & u & r & r & a & n & c & e \\
o & o & c & c & u & r & r & e & n & c & e \\
\end{array}
\]

Cost = \( 19\delta \).
What is the **minimum edit distance** for the following two strings, if insertion/deletion/change of a single character cost 1 unit?

374

473

(a) 1

(b) 2

(c) 3

(d) 4

(e) 5
What is the minimum edit distance for the following two strings, if insertion/deletion/change of a single character cost 1 unit?

373
473

(a) 1
(b) 2
(c) 3
(d) 4
(e) 5
What is the edit distance between...

What is the minimum edit distance for the following two strings, if insertion/deletion/change of a single character cost 1 unit?

37
473

(a) 1
(b) 2
(c) 3
(d) 4
(e) 5
Input  Given two words $X$ and $Y$, and gap penalty $\delta$ and mismatch costs $\alpha_{pq}$

Goal  Find alignment of minimum cost
Edit distance: The algorithm
Edit distance - Basic observation

Let $X = \alpha x$ and $Y = \beta y$

$\alpha, \beta$: strings.

$x$ and $y$ single characters.

Think about optimal edit distance between $X$ and $Y$ as alignment, and consider last column of alignment of the two strings:

**Case 1**

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$x$</th>
<th>or</th>
<th>$\alpha$</th>
<th>$x$</th>
<th>or</th>
<th>$\alpha x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>$y$</td>
<td></td>
<td>$\beta y$</td>
<td></td>
<td></td>
<td>$\beta$</td>
</tr>
</tbody>
</table>

Prefixes must have optimal alignment!
Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. If $(m, n)$ are not matched then either the $m^{th}$ position of $X$ remains unmatched or the $n^{th}$ position of $Y$ remains unmatched.

- **Case $x_m$ and $y_n$ are matched.**
  - Pay mismatch cost $\alpha_{x_m y_n}$ plus cost of aligning strings $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_{n-1}$

- **Case $x_m$ is unmatched.**
  - Pay gap penalty plus cost of aligning $x_1 \cdots x_{m-1}$ and $y_1 \cdots y_n$

- **Case $y_n$ is unmatched.**
  - Pay gap penalty plus cost of aligning $x_1 \cdots x_m$ and $y_1 \cdots y_{n-1}$
Subproblems and Recurrence

| $x_1 \ldots x_{i-1}$ | $x_i$ | or | $x_1 \ldots x_{i-1}$ | $x$ | or | $x_1 \ldots x_{i-1}x_i$ | $y_1 \ldots y_{j-1}$ | $y_j$ |
|----------------------|------|    |----------------------|-----|    |----------------------|----------------------|------|
| $y_1 \ldots y_{j-1}$ | $y_j$ |    | $y_1 \ldots y_{j-1}y_j$ |   |    | $y_1 \ldots y_{j-1}$ | $y_j$ |

**Optimal Costs**

Let $\text{Opt}(i,j)$ be optimal cost of aligning $x_1 \ldots x_i$ and $y_1 \ldots y_j$.

Then

$$\text{Opt}(i,j) = \min \left\{ \alpha_{x_iy_j} + \text{Opt}(i-1,j-1), \right.$$  
\left. \delta + \text{Opt}(i-1,j), \right.$$  
\left. \delta + \text{Opt}(i,j-1) \right\}$$
Subproblems and Recurrence

Optimal Costs
Let \( \text{Opt}(i, j) \) be optimal cost of aligning \( x_1 \cdots x_i \) and \( y_1 \cdots y_j \).

Then

\[
\text{Opt}(i, j) = \min \left\{ \begin{array}{l}
\alpha_{x_i y_j} + \text{Opt}(i - 1, j - 1), \\
\delta + \text{Opt}(i - 1, j), \\
\delta + \text{Opt}(i, j - 1)
\end{array} \right. 
\]

Base Cases: \( \text{Opt}(i, 0) = \delta \cdot i \) and \( \text{Opt}(0, j) = \delta \cdot j \)
Recursive Algorithm

Assume $X$ is stored in array $A[1..m]$ and $Y$ is stored in $B[1..n]$. Array $COST$ stores cost of matching two chars. Thus $COST[a, b]$ give the cost of matching character $a$ to character $b$.

$$EDIST(A[1..m], B[1..n])$$

If $(m = 0)$ return $n\delta$
If $(n = 0)$ return $m\delta$
$m_1 = \delta + EDIST(A[1..(m-1)], B[1..n])$
$m_2 = \delta + EDIST(A[1..m], B[1..(n-1)])$
$m_3 = COST[A[m], B[n]] + EDIST(A[1..(m-1)], B[1..(n-1)])$
return min$(m_1, m_2, m_3)$

($\mathcal{X} \mathcal{Y}$)
Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>ε</td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Opt}(i, j) = \begin{cases} 
\alpha_{x_i y_j} + \text{Opt}(i-1, j-1), \\
\delta + \text{Opt}(i-1, j), \\
\delta + \text{Opt}(i, j-1)
\end{cases}
\]

Base Cases:

- \(\text{Opt}(i, 0) = \delta \cdot i\)
- \(\text{Opt}(0, j) = \delta \cdot j\)
### Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Opt}(i, j) = \min \begin{cases} 
\alpha_{x_i,y_j} + \text{Opt}(i-1, j-1), \\
\delta + \text{Opt}(i-1, j), \\
\delta + \text{Opt}(i, j-1) 
\end{cases}
\]

**Base Cases:**
- \( \text{Opt}(i, 0) = \delta \cdot i \)
- \( \text{Opt}(0, j) = \delta \cdot j \)
**Example: DEED and DREAD**

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Opt}(i, j) = \begin{cases} 
\alpha_{x_i y_j} + \text{Opt}(i-1, j-1), \\
\delta + \text{Opt}(i-1, j), \\
\delta + \text{Opt}(i, j-1) 
\end{cases} \]

**Base Cases:**
- \( \text{Opt}(i, 0) = \delta \cdot i \)
- \( \text{Opt}(0, j) = \delta \cdot j \)
Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$E$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$E$</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Opt$(i,j) =$

\[
\begin{cases}
\alpha_{x_iy_j} + \text{Opt}(i-1,j-1), \\
\delta + \text{Opt}(i-1,j), \\
\delta + \text{Opt}(i,j-1)
\end{cases}
\]

Base Cases:

- Opt$(i,0) = \delta \cdot i$
- Opt$(0,j) = \delta \cdot j$
### Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Opt}(i, j) = \begin{cases} 
\alpha_{x_i y_j} + \text{Opt}(i - 1, j - 1), \\
\delta + \text{Opt}(i - 1, j), \\
\delta + \text{Opt}(i, j - 1)
\end{cases}
\]

**Base Cases:**
- \( \text{Opt}(i, 0) = \delta \cdot i \)
- \( \text{Opt}(0, j) = \delta \cdot j \)
### Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>ε</td>
<td>D</td>
<td>R</td>
<td>E</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
### Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$E$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$E$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$D$</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>$E$</td>
<td>$E$</td>
<td>$A$</td>
<td>$D$</td>
</tr>
</tbody>
</table>

![Diagram of DEED and DREAD network connections]
Example: DEED and DREAD

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Chosen

Cost of the 4 path = 2

Shortest path problem.

Cost = 2
Example: **DEED and DREAD**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic programming algorithm for edit-distance
As part of the input...

The cost of aligning a character against another character

$\Sigma$: Alphabet

We are given a cost function (in a table):

$\forall b, c \in \Sigma \quad COST[b][c] = \text{cost of aligning } b \text{ with } c.$

$\forall b \in \Sigma \quad COST[b][b] = 0$

$\delta$: price of deletion or insertion of a single character
**Dynamic program for edit distance**

\[
EDIST(A[1..m], B[1..n])
\]

\[
\begin{align*}
    &\text{int } M[0..m][0..n] \\
    &\text{for } i = 1 \text{ to } m \text{ do } M[i, 0] = i\delta \\
    &\text{for } j = 1 \text{ to } n \text{ do } M[0, j] = j\delta \\
    &\text{for } i = 1 \text{ to } m \text{ do } \\
        &\text{for } j = 1 \text{ to } n \text{ do } \\
        &\quad M[i][j] = \min\left\{ \begin{array}{l} \\
            \text{COST}[A[i]][B[j]] + M[i-1][j-1], \\
            \delta + M[i-1][j], \\
            \delta + M[i][j-1] \end{array} \right. \\
\end{align*}
\]

\[(R14)\]
Dynamic program for edit distance

\[ EDIST(A[1..m], B[1..n]) \]

\[ \text{int } M[0..m][0..n] \]

\[ \text{for } i = 1 \text{ to } m \text{ do } M[i, 0] = i\delta \]

\[ \text{for } j = 1 \text{ to } n \text{ do } M[0, j] = j\delta \]

\[ \text{for } i = 1 \text{ to } m \text{ do } \]

\[ \text{for } j = 1 \text{ to } n \text{ do } \]

\[ M[i][j] = \min \left\{ \begin{array}{l}
COST[A[i]][B[j]] + M[i-1][j-1], \\
\delta + M[i-1][j], \\
\delta + M[i][j-1]
\end{array} \right. \]

Analysis

- Running time is \( O(mn) \).
- Space used is \( O(mn) \).

\[ \text{cost for each subproblem : } O(1) \xrightarrow{} O(mn) \]

\[ \text{# of subproblems : } O(mn) \]

21
Reducing space for edit distance
Figure 1: Iterative algorithm in previous slide computes values in row order.
Optimizing Space

- Recall

\[ M(i, j) = \min \begin{cases} 
\alpha_{x_i y_j} + M(i - 1, j - 1), \\
\delta + M(i - 1, j), \\
\delta + M(i, j - 1) 
\end{cases} \]

- Entries in \( j^{th} \) column only depend on \( (j - 1)^{st} \) column and earlier entries in \( j^{th} \) column

- Only store the current column and the previous column reusing space; \( N(i, 0) \) stores \( M(i, j - 1) \) and \( N(i, 1) \) stores \( M(i, j) \)
Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ m \]

or

\[ 2 \text{ columns} = 2m \rightarrow O(m) \]

\[ 2 \text{ rows} = 2n \rightarrow O(n) \]

\[ O(\min(m,n)) \]
Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>ε</th>
<th>D</th>
<th>R</th>
<th>E</th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>$\varepsilon$</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$\varepsilon$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$E$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$D$</td>
<td>3</td>
<td></td>
<td>3</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
### Example: DEED vs. DREAD filled by column

<table>
<thead>
<tr>
<th></th>
<th>$\varepsilon$</th>
<th>$D$</th>
<th>$R$</th>
<th>$E$</th>
<th>$A$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$E$</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$E$</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$D$</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Computing in column order to save space

Figure 2: $M(i, j)$ only depends on previous column values. Keep only two columns and compute in column order.
Space Efficient Algorithm

\begin{verbatim}
for all  i do  N[i, 0] = i\delta
for j = 1 to n do
    N[0, 1] = j\delta  (* corresponds to M(0,j) *)
    for i = 1 to m do
        \[ N[i, 1] = \min\left\{ \alpha_{xi,yj} + N[i-1, 0] \right\}
        \[ N[i, 1] = \min\left\{ \delta + N[i-1, 1] \right\}
        \[ N[i, 1] = \delta + N[i, 0] \]
    for i = 1 to m do
        Copy  N[i, 0] = N[i, 1]
\end{verbatim}

Analysis
Running time is $O(mn)$ and space used is $O(2m) = O(m)$
Analyzing Space Efficiency

- From the $m \times n$ matrix $M$ we can construct the actual alignment (exercise)
- Matrix $N$ computes cost of optimal alignment but no way to construct the actual alignment
Longest Common Subsequence Problem
LCS Problem

Definition

LCS between two strings $X$ and $Y$ is the length of longest common subsequence between $X$ and $Y$.

```
ABAZDC → AB
BACBAD → AB
```

$AB$: Common Subsequence (CS)

```
ABAZDC → ABAD
BACBAD → BACBAD
```

$ABAD$: Longest CS (LCS)
LCS Problem

Definition

LCS between two strings \( X \) and \( Y \) is the length of longest common subsequence between \( X \) and \( Y \).

\[
\begin{align*}
ABAZDC & \quad ABAZDC \\
BACBAD & \quad BACBAD
\end{align*}
\]

Example

LCS between ABAZDC and BACBAD is 4 via ABAD
LCS Problem

Definition
LCS between two strings $X$ and $Y$ is the length of longest common subsequence between $X$ and $Y$.

\[
\begin{array}{c}
\text{ABAZDC} \\
\text{BACBAD}
\end{array}
\quad
\begin{array}{c}
\text{ABAZDC} \\
\text{BACBAD}
\end{array}
\]

Example
LCS between ABAZDC and BACBAD is 4 via ABAD

Derive a dynamic programming algorithm for the problem.
How do we plan out the recursion?

Start off with $A[m]$ and $B[n]$ and reason the following:

- Assuming $A[m] \neq B[n]$
  - One or neither of the end characters are in the LCS. Therefore:
    $$\max(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1]))$$

- Assuming $A[m] = B[n]$
  - $A[m]$ and $B[n]$ are both in the LCS. Therefore:
    $$LCS(A[1...m], B[1...n]) = 1 + LCS(A[1...m-1], B[1...n-1])$$

- Base Case: $A$ is empty or $B$ is empty
How do we plan out the recursion?

Start off with $A[1...m]$ and $B[1...n]$ and reason the following:

• Assuming $A[m] \neq B[n]$
  • One or neither of the end characters are in the LCS. Therefore:
    \[
    \text{max} \left( \text{LCS}(A[1...m-1], B[1...n]), \text{LCS}(A[1...m], B[1...n-1]) \right)
    \]

• Assuming $A[m] = B[n]$
  • $A[m]$ and $B[n]$ are both in the LCS. Therefore:
    \[
    \text{LCS}(A[1...m], B[1...n]) = 1 + \text{LCS}(A[1...m-1], B[1...n-1])
    \]

• Base Case: $A$ is empty or $B$ is empty
How do we plan out the recursion?

Start off with $A[1...m]$ and $B[1...n]$ and reason the following:

- Assuming $A[m] \neq B[n]
  - One or neither of the end characters are in the LCS. Therefore:
    $$\max (\text{LCS}(A[1...m - 1], B[1...n]), \text{LCS}(A[1...m], B[1...n - 1]))$$

- Base Case: $A$ is empty or $B$ is empty
How do we plan out the recursion?

Start off with $A[1...m]$ and $B[1...n]$ and reason the following:

- Assuming $A[m] \neq B[n]$
  - One or neither of the end characters are in the LCS. Therefore:
    \[
    \max(LCS(A[1...m - 1], B[1...n]), LCS(A[1...m], B[1...n - 1]))
    \]
- Assuming $A[m] = B[n]$

29
How do we plan out the recursion?

Start off with $A[1\ldots m]$ and $B[1\ldots n]$ and reason the following:

- Assuming $A[m] \neq B[n]$
  - One or neither of the end characters are in the LCS. Therefore:
    $$\max(LCS(A[1\ldots m-1], B[1\ldots n]), LCS(A[1\ldots m], B[1\ldots n-1]))$$

- Assuming $A[m] = B[n]$
  - $A[m]$ and $B[n]$ are both in the LCS. Therefore:
    $$LCS(A[1\ldots m], B[1\ldots n]) = 1 + LCS(A[1\ldots m-1], B[1\ldots n-1])$$

• Base Case: $A$ is empty or $B$ is empty
How do we plan out the recursion?

Start off with $A[1...m]$ and $B[1...n]$ and reason the following:

- Assuming $A[m] \neq B[n]$
  - One or neither of the end characters are in the LCS. Therefore:
    \[
    \max (LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1]))
    \]

- Assuming $A[m] = B[n]$
  - $A[m]$ and $B[n]$ are both in the LCS. Therefore:
    \[
    LCS(A[1...m], B[1...n]) = 1 + LCS(A[1...m-1], B[1...n-1])
    \]

- Base Case: $A$ is empty or $B$ is empty
LCS recursive definition

\[ A[1..n], B[1..m]: \text{Input strings.} \]

\[
LCS(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max \left( \begin{array}{c}
LCS(i-1, j), \\
LCS(i, j-1), \\
1 + LCS(i-1, j-1)
\end{array} \right) & \text{otherwise}
\end{cases}
\]

Running time: Similar to edit distance... \( O(nm) \)

Space: \( O(m) \) space.
LCS recursive definition

$A[1..n], B[1..m]$: Input strings.

$$LCS(i,j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max\left( \begin{array}{l} 
LCS(i-1,j), \\
LCS(i,j-1)
\end{array} \right) & \text{if } A[i] \neq B[j] \\
1 + LCS(i-1,j-1) & \text{if } A[i] = B[j]
\end{cases}$$

**Running time:** Similar to edit distance... $O(nm)$

**Space:** $O(m)$ space.
Longest common subsequence is just edit distance for the two sequences...

\[ A, B: \text{ input sequences, } \Sigma: \text{ “alphabet” all the different values in } A \text{ and } B \]

\[ \forall b, c \in \Sigma : b \neq c \quad \text{COST}[b][c] = +\infty. \]

\[ \forall b \in \Sigma \quad \text{COST}[b][b] = 1 \]

1 : price of deletion or insertion of a single character
Longest common subsequence is just edit distance for the two sequences...

\( A, B \): input sequences, \( \Sigma \): “alphabet” all the different values in \( A \) and \( B \)

\[
\forall b, c \in \Sigma : b \neq c \quad \Rightarrow \quad COST[b][c] = +\infty.
\]

\[
\forall b \in \Sigma \quad COST[b][b] = 1
\]

1: price of deletion of insertion of a single character

<table>
<thead>
<tr>
<th></th>
<th>( ED )</th>
<th>( LCS )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum ED</td>
<td><img src="image" alt="Maximum ED" /></td>
<td>( 9 + 0 = 9 )</td>
</tr>
<tr>
<td>Min LCS</td>
<td><img src="image" alt="Min LCS" /></td>
<td>( 9 + 0 = 9 )</td>
</tr>
<tr>
<td>Sub-opt ED</td>
<td><img src="image" alt="Sub-opt ED" /></td>
<td>( 8 + 1 = 9 )</td>
</tr>
<tr>
<td>Sub-opt LCS</td>
<td><img src="image" alt="Sub-opt LCS" /></td>
<td>( 8 + 1 = 9 )</td>
</tr>
<tr>
<td>Min ED</td>
<td><img src="image" alt="Min ED" /></td>
<td>( 6 + 3 = 9 )</td>
</tr>
<tr>
<td>Max LCS</td>
<td><img src="image" alt="Max LCS" /></td>
<td>( 6 + 3 = 9 )</td>
</tr>
</tbody>
</table>
Longest common subsequence is just edit distance for the two sequences...

\(A, B\): input sequences, \(\Sigma\): “alphabet” all the different values in \(A\) and \(B\)

\[
\forall b, c \in \Sigma : b \neq c \quad \text{COST}[b][c] = +\infty.
\]

\[
\forall b \in \Sigma \quad \text{COST}[b][b] = 1
\]

1: price of deletion of insertion of a single character

Length of longest common sub-sequence = \(m + n - \text{ed}(A, B)\)