Pre-lecture brain teaser

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

A(n)

$$f(x,y) = \sum_{i=1}^{min(x,y)} x * f(x+y-i, i-1), \quad \leftarrow B(n)$$

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

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.

ECE-374-B: Lecture 13 - Dynamic Programming II

Instructor: Nickvash Kani

October 14, 2025

University of Illinois Urbana-Champaign

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

$O(n^2)$ subproblems

$O(n)$

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

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

$A(n)$ $B(n)$ = Ideal Running time

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

$O(n^3)$

(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.

n $2n$

$2n-5$

$2n-4$

$2n-3$

$2n-2$

$2n-1$

n

Recipe for Dynamic Programming

1. Develop a recursive backtracking style algorithm $\mathcal{A}$ for given problem.
2. Identify structure of subproblems generated by $\mathcal{A}$ 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. recurrence
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.

Why is it called dynamic programming?

Dynamic programming was a technique “invented” by Richard Bellman. From his autobiography:

> *I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes. An interesting question is, Where did the name, dynamic programming, come from? The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term research in his presence. You can imagine how he felt, then, about the term mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose?* ***In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word “programming”. I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying I thought, lets kill two birds with one stone. Lets take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense.*** *Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name.* ***It was something not even a Congressman could object to. So I used it as an umbrella for my activities.***

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 $x_1 x_2 \ldots x_n$ and $y_1 y_2 \ldots y_m$ 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 <u>nearby</u> string?

What does nearness mean?

Question: Given two strings $x_1x_2 \dots x_n$ and $y_1y_2 \dots y_m$ what is a <u>distance</u> between them?

Edit Distance: minimum number of "edits" to transform x into y.

Edit Distance

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:

$$\underline{\text{F}}\text{OOD} \to \text{MO}\underline{\text{O}}\text{D} \to \text{MON}\underline{\text{O}}\text{D} \to \text{MONE}\underline{\text{D}} \to \text{MONEY}$$

Edit Distance: Alternate View

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.

$$\begin{array}{ccccc} F & O & O & & D \\ M & O & N & E & Y \end{array}$$

Edit Distance: Alternate View

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.

$$\begin{array}{ccccc} \mathrm{F} & \mathrm{O} & \mathrm{O} & & \mathrm{D} \\ \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

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 $M = \{(1,1), (2,2), (3,3), (4,5)\}$.

Edit Distance: Alternate View

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.

$$\begin{array}{ccccc} \mathrm{F} & \mathrm{O} & \mathrm{O} & & \mathrm{D} \\ \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

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.

Edit Distance Problem

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

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

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

Edit distance is special case when $\delta = \alpha_{pq} = 1$.

Edit distance as alignment

An Example

Example

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|}
o & & c & u & r & r & a & n & c & e \\
o & c & c & u & r & r & e & n & c & e
\end{array}
\qquad \text{Cost } = \delta + \alpha_{ae}$$

Alternative:

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
o & & c & u & r & r & & a & n & c & e \\
o & c & c & u & r & r & e & & n & c & e
\end{array}
\qquad \text{Cost } = 3\delta$$

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

$$\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}
o & c & u & r & r & a & n & c & e & & & & & & & & & & \\
& & & & & & & & & o & c & c & u & r & r & e & n & c & e
\end{array}$$

Cost = 19δ. $\delta(n+m)$

Sequence Alignment

Input Given two words X and Y, and gap penalty δ and mismatch costs α_{pq}

Goal Find alignment of minimum cost

Edit distance: The algorithm

Edit distance - Basic observation

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

α, β: 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:

α	x	or	α	x	or	αx	
β	y		βy			β	y

Prefixes must have optimal alignment!

$(0, 1)$

$OptAlign(X, Y) = OptAlign(\alpha, \beta) + \alpha_{x \, y}$

$OptAlign(X, Y) = OptAlign(\alpha, \beta y) + \delta$

Problem Structure

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
$y_1 \ldots y_{j-1}$	y_j

or

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

or

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

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

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

$\mathrm{Opt}(0,0) = 0$

$\mathrm{Opt}(i,0) = i$

$\mathrm{Opt}(0,j) = j$

Subproblems and Recurrence

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

or

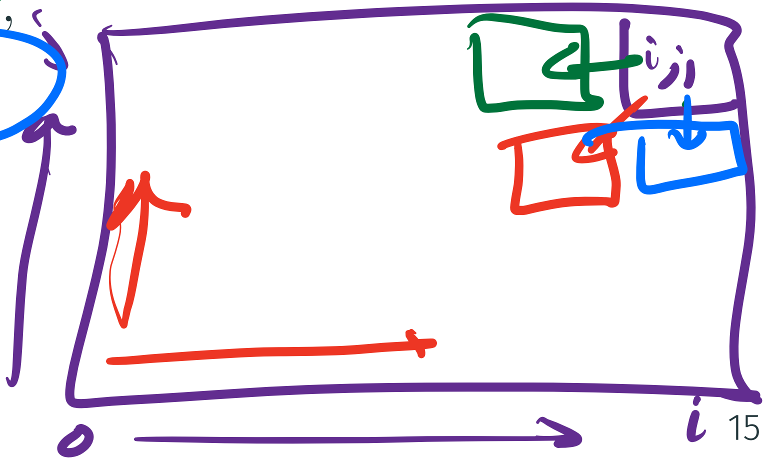
$x_1 \ldots x_{i-1}$	x
$y_1 \ldots y_{j-1} y_j$	

or

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

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

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



Base Cases: $\mathrm{Opt}(i, 0) = \delta \cdot i$ and $\mathrm{Opt}(0, j) = \delta \cdot j$

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 nδ
    If (n = 0) return mδ
    m_1 = δ + EDIST(A[1..(m − 1)], B[1..n])
    m_2 = δ + 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)
```

Example: DEED and DREAD

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1					
E	2					
E	3					
D	~~3~~ 4					

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

Base Cases:

- $\mathrm{Opt}(i, 0) = \delta \cdot i$
- $\mathrm{Opt}(0, j) = \delta \cdot j$

Example: DEED and DREAD

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2					
E	3					
D	3					

$$\mathrm{Opt}(i, j) =$$

$$\min \begin{cases} \alpha_{x_i y_j} + \mathrm{Opt}(i-1, j-1), \\ \delta + \mathrm{Opt}(i-1, j), \\ \delta + \mathrm{Opt}(i, j-1) \end{cases}$$

Base Cases:

- $\mathrm{Opt}(i, 0) = \delta \cdot i$
- $\mathrm{Opt}(0, j) = \delta \cdot j$

Example: DEED and DREAD

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2				
D	3					

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

α_{DE}

Opt(0, DE)

Base Cases:

- $\mathrm{Opt}(i,0) = \delta \cdot i$
- $\mathrm{Opt}(0,j) = \delta \cdot j$

Opt(1,3)

Opt(0, DEE)

Opt(ε, DE

Example: DEED and DREAD

i →

j ↓

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3					

Opt(DRE, DE)

$\text{Opt}(i, j) =$

DR D

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

1, 0; 1, DR DE, 2; 1, DRE D, 2; DR DE, 3

Base Cases:

- $\text{Opt}(i, 0) = \delta \cdot i$
- $\text{Opt}(0, j) = \delta \cdot j$

Example: DEED and DREAD

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	~~3~~ 4	3	3	2	2	2

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

Base Cases:

- $\mathrm{Opt}(i, 0) = \delta \cdot i$
- $\mathrm{Opt}(0, j) = \delta \cdot j$

D R E A D
D _ E E D

Example: DEED and DREAD

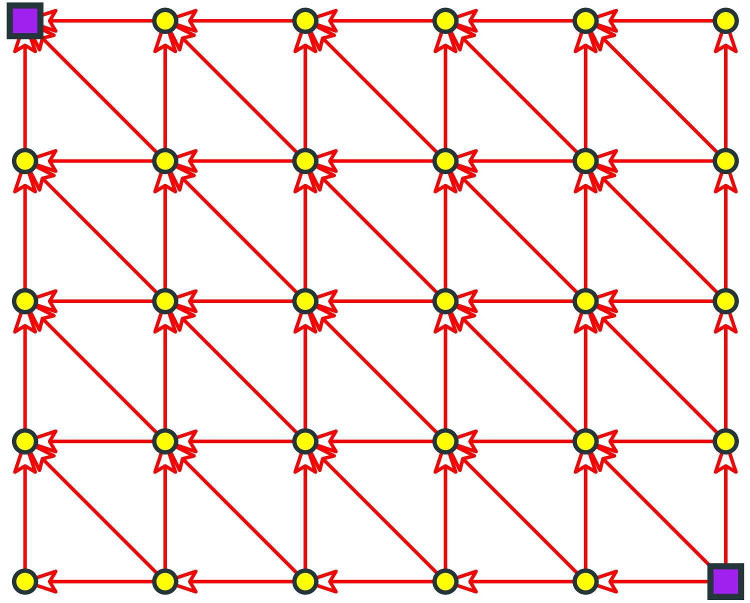
	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3	3	3	2	2	2

D	R	E	A	D
D	E	E		D

Example: DEED and DREAD

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3	3	3	2	2	2

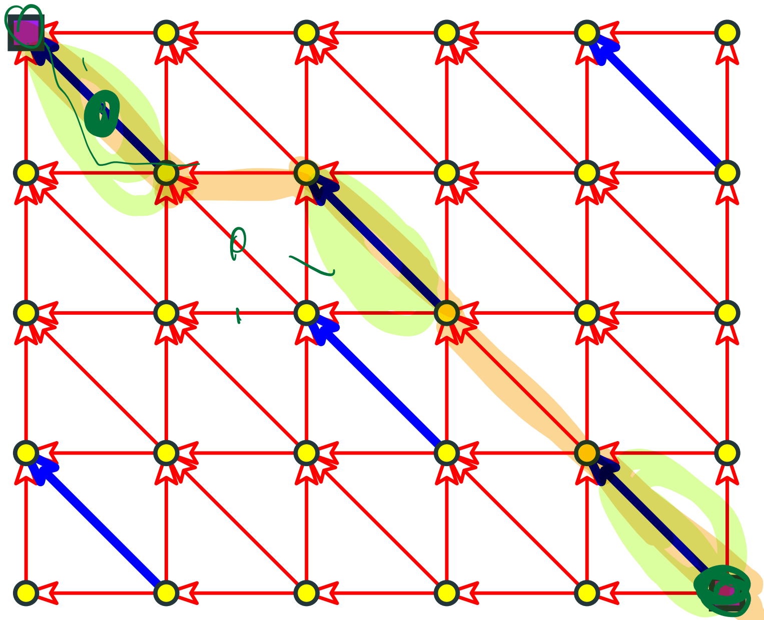
D	R	E	A	D
D	E	E		D



Example: DEED and DREAD

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3	3	3	2	2	2

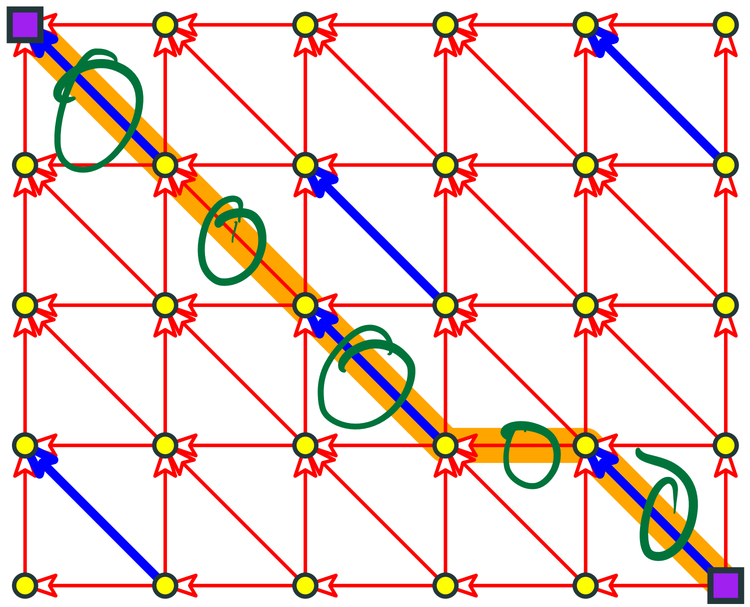
D	R	E	A	D
D		E	E	D



Example: DEED and DREAD

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3	3	3	2	2	2

D	R	E	A	D
D	E	E		D



Dynamic programming algorithm for edit-distance

As part of the input...

The cost of aligning a character against another character

Σ: Alphabet

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

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

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

δ : price of deletion of insertion of a single character

Dynamic program for edit distance

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

 int $M[0..m][0..n]$

 for $i = 1$ `to` m **do** $M[i,0] = i\delta$

 for $j = 1$ `to` n **do** $M[0,j] = j\delta$

 for $i = 1$ `to` m **do**

  for $j = 1$ `to` n **do**

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

Dynamic program for edit distance

$EDIST(A[1..m], B[1..n])$
    $int \quad M[0..m][0..n]$
    for $i = 1$ `to` m **do** $M[i, 0] = i\delta$
    for $j = 1$ `to` n **do** $M[0, j] = j\delta$

    for $i = 1$ `to` m **do**
        for $j = 1$ `to` n **do**

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

Analysis

- Running time is $O(mn)$
- Space used is $O(mn)$

Reducing space for edit distance

Matrix and DAG of computation of edit distance

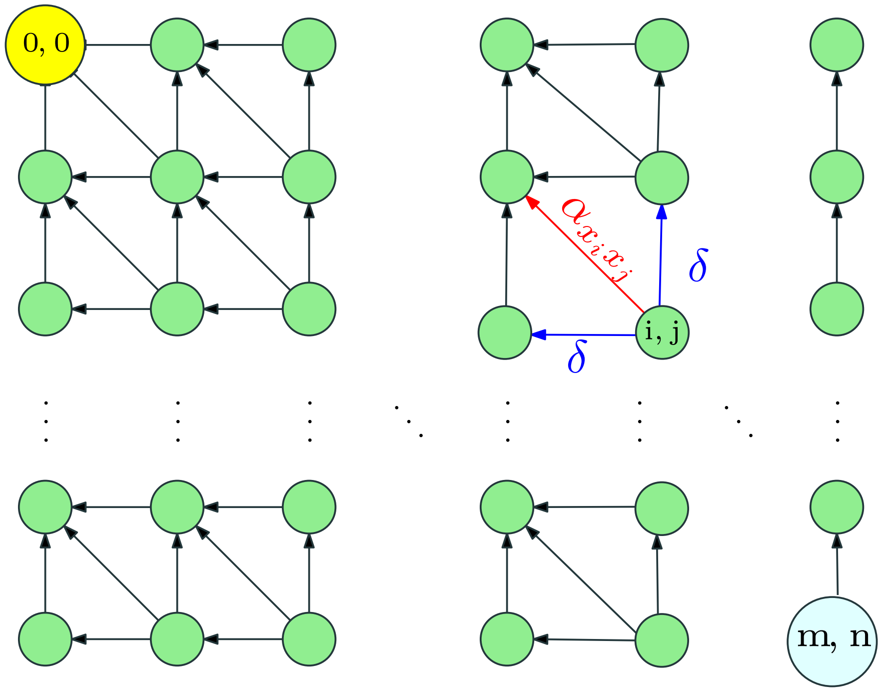


Figure 1: Iterative algorithm in previous slide computes values in row order.

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

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1					
E	2					
E	3					
D	3					

Example: DEED vs. DREAD filled by column

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0				
E	2	1				
E	3	2				
D	3	3				

Example: DEED vs. DREAD filled by column

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1			
E	2	1	1			
E	3	2	2			
D	3	3	3			

Example: DEED vs. DREAD filled by column

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2		
E	2	1	1	1		
E	3	2	2	1		
D	3	3	3	2		

Example: DEED vs. DREAD filled by column

	ε	*D*	*R*	*E*	*A*	*D*
ε	0	1	2	3	4	5
D	1	0	1	2	3	
E	2	1	1	1	2	
E	3	2	2	1	2	
D	3	3	3	2	2	

Example: DEED vs. DREAD filled by column

	ε	D	R	E	A	D
ε	0	1	2	3	4	5
D	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	2	3
D	3	3	3	2	2	2

Computing in column order to save space

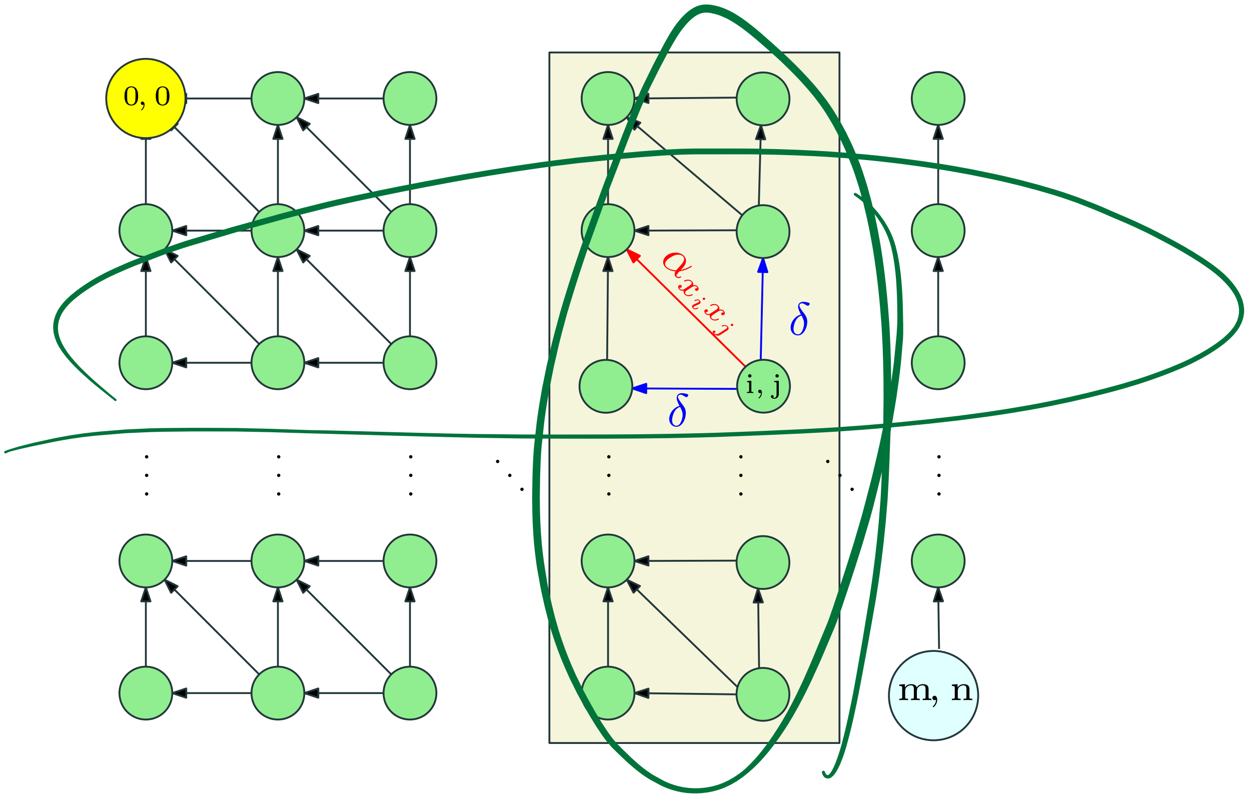


Figure 2: $M(i,j)$ only depends on previous column values. Keep only two columns and compute in column order.

Space Efficient Algorithm

$$
\begin{array}{l}
\textbf{for } \texttt{all } i \textbf{ do } N[i,0] = i\delta \\
\textbf{for } j = 1 \texttt{ to } n \textbf{ do} \\
\quad N[0,1] = j\delta \ \texttt{(* corresponds to } M(0,j) \texttt{ *)} \\
\quad \textbf{for } i = 1 \texttt{ to } m \textbf{ do} \\
\qquad N[i,1] = \min \begin{cases} \alpha_{x_i y_j} + N[i-1,0] \\ \delta + N[i-1,1] \\ \delta + N[i,0] \end{cases} \\
\quad \textbf{for } i = 1 \texttt{ to } m \textbf{ do} \\
\qquad \texttt{Copy } N[i,0] = N[i,1]
\end{array}
$$

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
- Space efficient computation of alignment? More complicated algorithm — see notes and Kleinberg-Tardos book.

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.

$ABAZDC$ $\quad\quad$ $ABAZDC$

$BACBAD$ $\quad\quad$ $BACBAD$

AC

BAD $\quad\quad$ ABAD

Definition

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

$$\begin{array}{ccc} ABAZDC & \qquad & ABAZDC \\ BACBAD & \qquad & BACBAD \end{array}$$

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.

$$\begin{array}{ccc} ABAZDC & \qquad & ABAZDC \\ BACBAD & \qquad & BACBAD \end{array}$$

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?

How do we plan out the recursion?

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

How do we plan out the recursion?

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

- Assuming $A[m] \neq B[n]$
  - The one or neither of the end characters are in the LCS. Therefore becomes:

$$\max\left(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1])\right)$$

How do we plan out the recursion?

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

- Assuming $A[m] \neq B[n]$
  - The one or neither of the end characters are in the LCS. Therefore becomes:

$$\max\left(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1])\right)$$

- Assuming $A[m] = B[n]$

How do we plan out the recursion?

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

- Assuming $A[m] \neq B[n]$
  - The one or neither of the end characters are in the LCS. Therefore becomes:

$$\max\left(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1])\right)$$

- Assuming $A[m] = B[n]$
  - Either $A[m]$ and $B[n]$ are both in the LCS. Therefore:

$$LCS(A[1...m], B[1...n]) = 1 + LCS(A[1...m-1], B[1...n-1])$$

How do we plan out the recursion?

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

- Assuming $A[m] \neq B[n]$
  - The one or neither of the end characters are in the LCS. Therefore becomes:

$$\max\left(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1])\right)$$

- Assuming $A[m] = B[n]$
  - Either $A[m]$ and $B[n]$ are both in the LCS. Therefore:

$$LCS(A[1...m], B[1...n]) = 1 + LCS(A[1...m-1], B[1...n-1])$$

  - Or $A[m]$ and $B[n]$ is not in the LCS. Therefore the LCS is either:

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

How do we plan out the recursion?

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

- Assuming $A[m] \neq B[n]$
  - The one or neither of the end characters are in the LCS. Therefore becomes:

$$\max\left(LCS(A[1...m-1], B[1...n]), LCS(A[1...m], B[1...n-1])\right)$$

- Assuming $A[m] = B[n]$
  - Either $A[m]$ and $B[n]$ are both in the LCS. Therefore:

$$LCS(A[1...m], B[1...n]) = 1 + LCS(A[1...m-1], B[1...n-1])$$

  - Or $A[m]$ and $B[n]$ is not in the LCS. Therefore the LCS is either:

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

- Base Case:

LCS recursive definition

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

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

LCS recursive definition

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

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

Running time: $O(nm)$

Space: $O(nm) \longrightarrow O(\min(n,m))$

Longest common subsequence is just edit distance for the two sequences...

A, B: input sequences, Σ: "alphabet" all the different values in A and B

$$\forall b, c \in \Sigma : b \neq c \qquad COST[b][c] = +\infty.$$
$$\forall b \in \Sigma \qquad COST[b][b] = 1$$

$1:$ price of deletion of insertion of a single character

Longest common subsequence is just edit distance for the two sequences...

A, B: input sequences, Σ: "alphabet" all the different values in A and B

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

$$\forall b \in \Sigma \qquad COST[b][b] = 1$$

$1:$ price of deletion of insertion of a single character

										ED	LCS
Maximum ED	D	R	E	A	D					9	0
Min LCS						D	E	E	D		
Sub-opt ED	D	R	E	A	D					8	1
Sub-opt LCS					D	E	E	D			
Min ED	D	R	E	A		D	LCS = DED			6	3 (m+n − ED)
Max LCS	D		E		E	D					

Longest common subsequence is just edit distance for the two sequences...

A, B: input sequences, Σ: "alphabet" all the different values in A and B

$$\forall b, c \in \Sigma : b \neq c \qquad COST[b][c] = +\infty.$$
$$\forall b \in \Sigma \qquad COST[b][b] = 1$$

1 : price of deletion of insertion of a single character

Length of longest common sub-sequence = $n + m - ED$

How to improve dynamic programming?

Key skills you need to successfully come up with dynamic programming solutions:

- Formulate recurrences for various problems (There's only like 10-20 dynamic programming problems in general, rest are rewrites of the same concepts).
- Be able to describe recurrences *in plain english*.
- Identify subproblem order
- PRACTICE.