Dynamic Programming:
String Editing
Outline

- DNA Sequence Comparison: First Success Stories
- Manhattan Tourist Problem
- Longest Paths in Grid Graphs
- Sequence Alignment
- Edit Distance
- Back to Longest Common Subsequence Problem
- Reducing the space requirement
The Central Dogma of Molecular Biology

**Genome:** The digital backbone of molecular biology

Transcripts: Perform functions encoded in the genome
The Sequence Alignment Problem

Compare two strings $A$ and $B$ and measure their similarity by finding the optimal alignment between them.

The alignment is classically based on the transformation of one sequence into the other, via operations of substitutions, insertions, and deletions (indels).

The Scoring Matrix $\delta$

$$
\begin{array}{|c|c|c|c|c|}
\hline
 & a & c & g & t \\
\hline
- & -1 & -1 & -1 & -1 \\
a & -1 & 1 & -1 & -1 & -1 \\
c & -1 & -1 & 1 & -1 & -1 \\
g & -1 & -1 & -1 & 1 & -1 \\
t & -1 & -1 & -1 & -1 & 1 \\
\hline
\end{array}
$$
Two Sequence Alignment Problems

Global Alignment.

\[ A = \text{c t a c g a g a c} \]

\[ B = \text{a a c g a c g a t} \]

Local Alignment.

\[ A = \text{c t a c g a g a c} \]

\[ B = \text{a a c g a c g a t} \]
Two Sequence Alignment Problems

Global Alignment.

A = c t a c g a g a c
B = a a c g a c g a t

Local Alignment.

A = c t a c g a g a c
B = a a c g a c g a t

The Scoring Matrix \( \delta \)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>a</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>g</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Two Sequence Alignment Problems

Global Alignment.

A = ctacgagac
| | | | | | |
B = aacgacgat

Value: 2

Local Alignment.

A = ct acgagac
| | | | | | |
B = aacgacgat

The Scoring Matrix \( \delta \)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>a</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>g</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Two Sequence Alignment Problems

Global Alignment.

A = ctacgagac
   |   |   |   |   |
B = aacgacgat

Value: 2

Local Alignment.

A = ct

   |   |   |   |   |
B = aacgacgagt

Value: 5
The Sequence Alignment Problem

A = ctacgagac
B = aacgcgaat

The Scoring Matrix $\delta$

Compare two strings A and B and measure their similarity by finding the optimal alignment between them.

The alignment is classically based on the transformation of one sequence into the other, via operations of substitutions, insertions, and deletions (indels).
Two Sequence Alignment Problems

Global Alignment.

\[ A = \begin{array}{cccccc}
    c & t & a & c & g & a & g & a & c \\
\end{array} \]

\[ B = \begin{array}{ccccccc}
    a & a & c & g & a & c & g & a & t \\
\end{array} \]

Value: 2

Local Alignment.

\[ A = \begin{array}{ccc}
    c & t & \ \\
\end{array} \]

\[ \begin{array}{ccccccc}
    a & c & g & a & g & a & c \\
\end{array} \]

\[ B = \begin{array}{ccccccc}
    a & a & c & g & a & c & g & a & t \\
\end{array} \]

Value: 5
• Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene’s function

• In 1984 Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene
Cystic Fibrosis

- **Cystic fibrosis** (CF) is a chronic and frequently fatal genetic disease of the body's mucus glands (abnormally high level of mucus in glands). CF primarily affects the respiratory systems in children.

- Mucus is a slimy material that coats many epithelial surfaces and is secreted into fluids such as saliva.
Cystic Fibrosis: Inheritance

• In early 1980s biologists hypothesized that CF is an autosomal recessive disorder caused by mutations in a gene that remained unknown till 1989

• Heterozygous carriers are asymptomatic

• Must be homozygously recessive in this gene in order to be diagnosed with CF
Cystic Fibrosis: Finding the Gene
Finding Similarities between the Cystic Fibrosis Gene and ATP binding proteins

• ATP binding proteins are present on cell membrane and act as transport channel

• In 1989 biologists found similarity between the cystic fibrosis gene and ATP binding proteins

• A plausible function for cystic fibrosis gene, given the fact that CF involves sweet secretion with abnormally high sodium level
Cystic Fibrosis: Mutation Analysis

If a high % of cystic fibrosis (CF) patients have a certain mutation in the gene and the normal patients don’t, then that could be an indicator of a mutation that is related to CF

A certain mutation was found in 70% of CF patients, convincing evidence that it is a predominant genetic diagnostics marker for CF
Cystic Fibrosis and the CFTR Protein

- **CFTR (Cystic Fibrosis Transmembrane conductance Regulator)** protein is acting in the cell membrane of epithelial cells that secrete mucus.
- These cells line the airways of the nose, lungs, the stomach wall, etc.
Mechanism of Cystic Fibrosis

- The **CFTR protein** (1480 amino acids) regulates a chloride ion channel
- Adjusts the “wateriness” of fluids secreted by the cell
- Those with cystic fibrosis are missing one single amino acid in their CFTR
- Mucus ends up being too thick, affecting many organs
Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.
Manhattan Tourist Problem: Formulation

**Goal:** Find the longest path in a weighted grid.

**Input:** A weighted grid $G$ with two distinct vertices, one labeled “source” and the other labeled “sink”

**Output:** A longest path in $G$ from “source” to “sink”
MTP: An Example

source

sink

j coordinate

i coordinate
MTP: Greedy Algorithm Is Not Optimal

promising start, but leads to bad choices!
MTP: Simple Recursive Program

\[
\text{MT}(n,m) \\
\begin{align*}
\text{if } n=0 \text{ or } m=0 & \\
\quad \text{return } \text{MT}(n,m) \\
\ & x \leftarrow \text{MT}(n-1,m) + \\
\ & \quad \text{length of the edge from } (n-1,m) \text{ to } (n,m) \\
\ & y \leftarrow \text{MT}(n,m-1) + \\
\ & \quad \text{length of the edge from } (n,m-1) \text{ to } (n,m) \\
\ & \text{return max}\{x,y\}
\end{align*}
\]
MTP: Simple Recursive Program

\[
\text{MT}(n,m)
\]
\[
\begin{align*}
x & \leftarrow \text{MT}(n-1,m) + \\
& \quad \text{length of the edge from } (n-1,m) \text{ to } (n,m) \\
y & \leftarrow \text{MT}(n,m-1) + \\
& \quad \text{length of the edge from } (n,m-1) \text{ to } (n,m) \\
\text{return } \min\{x,y\}
\end{align*}
\]

What’s wrong with this approach?
MTP: Dynamic Programming

- Calculate optimal path score for each vertex in the graph
- Each vertex’s score is the maximum of the prior vertices score plus the weight of the respective edge in between
MTP: Dynamic Programming (cont’d)

\[ S_{0,2} = 3 \]
\[ S_{1,1} = 4 \]
\[ S_{2,0} = 8 \]
MTP: Dynamic Programming
(cont’d)

source

\[ S_{3,0} = 8 \]
\[ S_{1,2} = 13 \]
\[ S_{2,1} = 9 \]
MTP: Dynamic Programming (cont’d)

The diagram illustrates a problem involving a grid with numbers and arrows indicating possible moves. The source is at position (0,0), and the goal is to reach (3,3) with the minimum cost, represented by the labels at each node.

- From (0,0) to (1,0) with a cost of -5.
- From (1,0) to (1,1) with a cost of -5.
- From (1,1) to (2,1) with a cost of 10.
- From (2,1) to (2,2) with a cost of -5.
- From (2,2) to (3,2) with a cost of -5.
- From (3,2) to (3,3) with a cost of 8.

The values at each node are calculated as follows:

- $S_{1,3} = 8$
- $S_{2,2} = 12$
- $S_{3,1} = 9$

A greedy algorithm would fail in this scenario, as indicated by the red arrow from (2,1) to (3,1) with a cost of -3, which is higher than the direct path from (2,2) to (3,3) with a cost of 8.
MTP: Dynamic Programming (cont’d)

\[ S_{2,3} = 15 \]

\[ S_{3,2} = 9 \]
MTP: Dynamic Programming
(cont’d)

(source)

Done!

(showing all back-traces)

\[ S_{3,3} = 16 \]
MTP: Recurrence

Computing the score for a point \((i,j)\) by the recurrence relation:

\[
\begin{align*}
    s_{i,j} &= \max \begin{cases} 
        s_{i-1,j} + \text{weight of the edge between (i-1, j) and (i, j)} \\
        s_{i,j-1} + \text{weight of the edge between (i, j-1) and (i, j)}
    \end{cases}
\end{align*}
\]

The running time is \(n \times m\) for a \(n\) by \(m\) grid

\((n = \# \text{ of rows}, \ m = \# \text{ of columns})\)
Manhattan Is Not A Perfect Grid

What about diagonals?

- The score at point B is given by:

\[ s_B = \max \left\{ s_{A_1} + \text{weight of the edge } (A_1, B), \\
      s_{A_2} + \text{weight of the edge } (A_2, B), \\
      s_{A_3} + \text{weight of the edge } (A_3, B) \right\} \]
Manhattan Is Not A Perfect Grid (cont’d)

Computing the score for point \( x \) is given by the recurrence relation:

\[
s_x = \max \left\{ s_y + \text{weight of vertex } (y, x) \mid y \in \text{Predecessors}(x) \right\}
\]

- Predecessors \((x)\) – set of vertices that have edges leading to \( x \)

- The running time for a grid graph \( G(V, E) \) (\( V \) is the set of all vertices and \( E \) is the set of all edges) is \( O(E) \) since each edge is evaluated once
Traveling in the Grid

• The only hitch is that one must decide on the order in which visit the vertices.

• By the time the vertex $x$ is analyzed, the values $s_y$ for all its predecessors $y$ should be computed – otherwise we are in trouble.

• We need to traverse the vertices in some order.

• Try to find such order for a directed cycle.
DAG: Directed Acyclic Graph

- Since Manhattan is not a perfect regular grid, we represent it as a DAG
- DAG for Dressing in the morning problem
Topological Ordering

- A numbering of vertices of the graph is called **topological ordering** of the DAG if every edge of the DAG connects a vertex with a smaller label to a vertex with a larger label.

- In other words, if vertices are positioned on a line in an increasing order of labels then all edges go from left to right.
Topological ordering

- 2 different topological orderings of the DAG
Longest Path in DAG Problem

- **Goal**: Find a longest path between two vertices in a weighted DAG

- **Input**: A weighted DAG $G$ with source and sink vertices

- **Output**: A longest path in $G$ from source to sink
Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.
Manhattan Tourist Problem: Formulation

**Goal**: Find the longest path in a weighted grid.

**Input**: A weighted grid $G$ with two distinct vertices, one labeled “source” and the other labeled “sink”

**Output**: A longest path in $G$ from “source” to “sink”
MTP: Greedy Algorithm Is Not Optimal

promising start, but leads to bad choices!
MTP: Dynamic Programming

(source)

$S_{3,3} = 16$

Done!

(showing all back-traces)
MTP: Recurrence

Computing the score for a point \((i,j)\) by the recurrence relation:

\[
    s_{i,j} = \max \left\{ s_{i-1,j} + \text{weight of the edge between } (i-1,j) \text{ and } (i,j),
                           s_{i,j-1} + \text{weight of the edge between } (i,j-1) \text{ and } (i,j) \right\}
\]

The running time is \(n \times m\) for a \(n\) by \(m\) grid

\((n = \# \text{ of rows}, m = \# \text{ of columns})\)
Manhattan Is Not A Perfect Grid

What about diagonals?

- The score at point B is given by:

\[ s_B = \max \{ s_{A_1} + \text{weight of the edge } (A_1, B), \]
\[ s_{A_2} + \text{weight of the edge } (A_2, B), \]
\[ s_{A_3} + \text{weight of the edge } (A_3, B) \]
Manhattan Is Not A Perfect Grid (cont’d)

Computing the score for point $x$ is given by the recurrence relation:

$$s_x = \max \left\{ s_y + \text{weight of vertex } (y, x) \text{ where } y \in \text{Predecessors}(x) \right\}$$

- Predecessors $(x)$ – set of vertices that have edges leading to $x$
- The running time for a grid graph $G(V, E)$ ($V$ is the set of all vertices and $E$ is the set of all edges) is $O(E)$ since each edge is evaluated once
Longest Path in DAG Problem

- **Goal**: Find a longest path between two vertices in a weighted DAG

- **Input**: A weighted DAG $G$ with source and sink vertices

- **Output**: A longest path in $G$ from source to sink
Longest Path in DAG: Dynamic Programming

• Suppose vertex $v$ has indegree 3 and predecessors $\{u_1, u_2, u_3\}$
• Longest path to $v$ from source is:

$$
\phi_v = \max \begin{cases} 
\phi_{u_1} + \text{weight of edge from } u_1 \text{ to } v \\
\phi_{u_2} + \text{weight of edge from } u_2 \text{ to } v \\
\phi_{u_3} + \text{weight of edge from } u_3 \text{ to } v 
\end{cases}
$$

In General:

$$
s_v = \max_u (s_u + \text{weight of edge from } u \text{ to } v)
$$
Traversing the Manhattan Grid

- 3 different strategies:
  - a) Column by column
  - b) Row by row
  - c) Along diagonals
Alignment: 2 row representation

Given 2 DNA sequences $v$ and $w$:

- $v : \textcolor{red}{A} \textcolor{blue}{T} \textcolor{brown}{C} \textcolor{blue}{T} \textcolor{brown}{G} \textcolor{blue}{A} \textcolor{red}{T}$  \quad m = 7
- $w : \textcolor{green}{T} \textcolor{purple}{G} \textcolor{green}{C} \textcolor{purple}{A} \textcolor{green}{T} \textcolor{purple}{A} \textcolor{green}{A}$  \quad n = 6

Alignment: $2 \times k$ matrix ($k > m, n$)

<table>
<thead>
<tr>
<th>$v$</th>
<th>$w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>--</td>
</tr>
<tr>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>--</td>
</tr>
<tr>
<td>T</td>
<td>--</td>
</tr>
</tbody>
</table>

4 matches  \quad 2 insertions  \quad 2 deletions
Aligning DNA Sequences

\[ V = \text{ATCTGATG} \quad n = 8 \]
\[ W = \text{TGCATAC} \quad m = 7 \]

4 matches
1 mismatches
2 insertions
2 deletions

Indels

Deletion

Insertion
Edit Script: Example

TGCATAT → ATCCGAT in 5 steps

TGCATAT → (delete last T)
TGCATA → (delete last A)
TGCAT → (insert A at front)
ATGCAT → (substitute C for 3rd G)
ATCCCAT → (insert G before last A)
ATCCGAT → (Done)
Edit Script: Example

TGCATAT $\rightarrow$ ATCCGAT in 5 steps

- TGCATAT $\rightarrow$ (delete last T)
- TGCATA $\rightarrow$ (delete last A)
- TGCAT $\rightarrow$ (insert A at front)
- ATGCAT $\rightarrow$ (substitute C for 3rd G)
- ATCCCAT $\rightarrow$ (insert G before last A)
- ATCCGAT $\rightarrow$ (Done)

What is the minimal edit script?
Edit Script: Example (cont’d)

TGCATAT → ATCCGAT in 4 steps

TGCATAT → (insert A at front)
ATGCATAT → (delete 6th T)
ATGCATA → (substitute G for 5th A)
ATGCGTA → (substitute C for 3rd G)
ATCCGAT  (Done)
Edit Script: Example (cont’d)

TGCATAT $\rightarrow$ ATCCGAT in 4 steps

- TGCATAT $\rightarrow$ (insert A at front)
- ATGCATAT $\rightarrow$ (delete 6\textsuperscript{th} T)
- ATGCATA $\rightarrow$ (substitute G for 5\textsuperscript{th} A)
- ATGCAGTA $\rightarrow$ (substitute C for 3\textsuperscript{rd} G)
- ATCCGAT (Done)

Can it be done in 3 steps???
Two Sequence Alignment Problems

Global Alignment.

\[ A = \textcolor{red}{c \ t \ a \ c \ g \ a \ g \ a \ c} \]
\[ B = \textcolor{red}{a \ a \ c \ g \ a \ c \ g \ a \ t} \]

Local Alignment.

\[ A = \textcolor{red}{c \ t \ a \ c \ g \ a \ g \ a \ c} \]
\[ B = \textcolor{red}{a \ a \ c \ g \ a \ c \ g \ a \ t} \]
The $O(n^2)$ time, Classical Dynamic Programming Algorithm

The Alignment Graph

<table>
<thead>
<tr>
<th>$B$</th>
<th>$a$</th>
<th>$c$</th>
<th>$g$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$c$</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$g$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>$t$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

The Scoring Matrix $\delta$

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$c$</th>
<th>$g$</th>
<th>$t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$a$</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$c$</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>$g$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>$t$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>
Computing the Optimal Global Alignment Value

Classical Dynamic Programming: $O(n^2)$
Computing an Optimal Local Alignment Value

Classical Dynamic Programming: $O(n^2)$
Computing the Optimal Global Alignment Value

Classical Dynamic Programming: $O(n^2)$
Dynamic programming

Design technique, like divide-and-conquer.

Example: Longest Common Subsequence (LCS)
• Given two sequences $x[1 \ldots m]$ and $y[1 \ldots n]$, find a longest subsequence common to them both.

“a” not “the”

$x$: A B C B D A B

$y$: B D C A B A

$BCBA = LCS(x, y)$

functional notation, but not a function
Every path is a common subsequence.

Every diagonal edge adds an extra element to the common subsequence.

**LCS Problem:** Find a path with maximum number of diagonal edges.
Alignment: Dynamic Programming

\[ s_{i,j} = \max \begin{cases} 
  s_{i-1, j-1} + 1 & \text{if } v_i = w_j \\
  s_{i-1, j} + 0 \\
  s_{i, j-1} + 0 
\end{cases} \]

This recurrence corresponds to the Manhattan Tourist problem (three incoming edges into a vertex) with all horizontal and vertical edges weighted by zero and diagonal edges weighted by one.
Alignment as a Path in the Edit Graph

Every path in the edit graph corresponds to an alignment:
Alignment as a Path in the Edit Graph

Old Alignment

\[
\begin{array}{c}
0122345677 \\
v= \text{AT}_G\text{T}\text{T}\text{TAT}_T \\
w= \text{ATCGT}_T\text{A}_C \\
0123455667
\end{array}
\]

New Alignment

\[
\begin{array}{c}
0122345677 \\
v= \text{AT}_G\text{T}\text{T}\text{TAT}_T \\
w= \text{ATCG}_T\text{TA}_C \\
0123445667
\end{array}
\]
Alignment as a Path in the Edit Graph

\[
\begin{align*}
 v &=  \text{AT\_GTTA\_T} \\
 w &=  \text{ATCGT\_A\_C} \\
 (0,0), (1,1), (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7) \\
\end{align*}
\]
Alignment: Dynamic Programming

\[ s_{i,j} = \begin{cases} 
  s_{i-1, j-1} + 1 & \text{if } v_i = w_j \\
  \max \{ s_{i-1, j}, s_i, j-1 \} & \text{otherwise}
\end{cases} \]
Dynamic Programming Example

Initialize 1\textsuperscript{st} row and 1\textsuperscript{st} column to be all zeroes.

Or, to be more precise, initialize 0\textsuperscript{th} row and 0\textsuperscript{th} column to be all zeroes.
Dynamic Programming Example

$$S_{i,j} = \max\left\{ S_{i-1,j-1}, S_{i-1,j}, S_{i,j-1}\right\}$$
- value from NW +1, if $v_i = w_j$
- value from North (top)
- value from West (left)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Programming Example

\[
S_{i,j} = \begin{cases} 
S_{i-1, j-1} & \text{value from NW +1, if } v_i = w_j \\
\max \{ S_{i-1, j}, S_{i, j-1} \} & \text{value from North (top)} \\
S_{i, j-1} & \text{value from West (left)} 
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
w & A & T & C & G & T & A & C \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
4 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
5 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
6 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
7 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
Alignment: Backtracking

Arrows show where the score originated from.

- if from the top
- if from the left
- if $v_i = w_j$
Alignment: Dynamic Programming

\[ s_{i,j} = \begin{cases} 
  s_{i-1, j-1} + 1 & \text{if } v_i = w_j \\
  \max \
  \quad s_{i-1, j} \\
  \quad s_i, j-1 
\end{cases} \]
LCS Algorithm

1. **LCS**(v, w)
2. for \( i \leftarrow 1 \) to \( n \)
3. \( s_{i,0} \leftarrow 0 \)
4. for \( j \leftarrow 1 \) to \( m \)
5. \( s_{0,j} \leftarrow 0 \)
6. for \( i \leftarrow 1 \) to \( n \)
7. for \( j \leftarrow 1 \) to \( m \)
8. \( s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases} \)
9. \( b_{i,j} \leftarrow \begin{cases} " \leftarrow " & \text{ if } s_{i,j} = s_{i-1,j} \\ " \uparrow " & \text{ if } s_{i,j} = s_{i,j-1} \\ " \rightarrow " & \text{ if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases} \)
10. return \((s_{n,m}, b)\)
Now What?

• LCS(v,w) created the alignment grid

• Now we need a way to read the best alignment of v and w

• Follow the arrows backwards from sink
Printing LCS: Backtracking

1. PrintLCS(b,v,i,j)
2. if $i = 0$ or $j = 0$
3. return
4. if $b_{i,j} = \text{"\diagup"}$
5. PrintLCS(b,v,$i-1$,$j-1$)
6. print $v_i$
7. else
8. if $b_{i,j} = \text{"\downarrow"}$
9. PrintLCS(b,v,$i-1$,$j$)
10. else
11. PrintLCS(b,v,$i,j-1$)
LCS Runtime

• It takes $O(nm)$ time to fill in the $n \times m$ dynamic programming matrix.

• Why $O(nm)$? The pseudocode consists of a nested “for” loop inside of another “for” loop to set up a $n \times m$ matrix.

• Memory? Naively $O(nm)$