COMP 355
Advanced Algorithms
Dynamic Programming:
Manhattan Tourist Problem
Sequence Alignment
Section 6.6-6.7(KT)
The Chips game: Dynamic programming activity

This is a game between two players. It starts with two piles of chips, ten chips per pile. On each turn, a player may take one chip from one pile, or two chips, one from each pile. If one pile is empty, you can only take one chip from the other pile. The player who takes the last chip or chips wins the game.

1. Set up and play the game with another person. Try to win! Play a few more times. Keep track of whether the first player wins or the first player loses (and what moves the players made).

2. Would you rather be the first player or not? Does it matter? Why or why not?

While you are playing, think about how you could use a DP algorithm to determine the answers to the above questions.
What you were playing can be called the “10-10” game.

- If you go first and take one chip from each pile, your opponent becomes the first player in the “9-9” game.
- If you take one chip from one pile, your opponent becomes the first player in the “9-10” game or the “10-9” game.

But these games still have too many chips in them to know exactly what will happen in the end.

Let’s make it simpler. Try the “1-1” game. Should the first player always win? How?

Write W if the first player can guarantee a win, and L if the first player is going to lose.

- **W** 1-1 The first player always wins, so I filled in the blank with W
- **_** 0-1
- **_** 1-0
- **L** 2-0 → 1-0 (W) The first player always loses the 2-0 game, so I filled in the blank with L
- **_** 3-0 This is the same as the 0-3 game, right?
Finding a Winning Strategy

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

Does the first player win the 10-10 game?

Looking at your analysis of the game, try to describe a strategy for playing that will guarantee a win to the player who follows it.
Finding a Winning Strategy

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

Does the first player win the 10-10 game?

Looking at your analysis of the game, try to describe a strategy for playing that will guarantee a win to the player who follows it.
Yet Another DP Example

Manhattan Tourist Problem (MTP)

Imagine seeking a path from source to destination in a Manhattan-like city grid that maximizes the number of attractions (*) passed. With the following caveat—
at every step you must make progress towards the goal.

We treat the city map as a graph, with a “vertices” at each corner, and weighted edges along each block. The weights are the number of attractions along each block.
Manhattan Tourist Problem: Formulation

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

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

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

Greedy Algorithm:
At each step select the maximum weight block.

Greed has a short horizon

promising start, but leads to bad choices!
MTP as a Dynamic Program

![Diagram showing a grid with labeled nodes and edges, indicating a path from source to dest with numbers representing costs or weights.](image)
MTP Strategy

• Instead of solving the Manhattan Tourist problem directly, (i.e. the path from (0,0) to (n,m)) we will solve a more general problem: find the longest path from (0,0) to any arbitrary vertex (i,j).

• If the longest path from (0,0) to (n,m) passes through some vertex (i,j), then the path from (0,0) to (i,j) must be the longest. Otherwise, you could increase your path by changing it.
MTP: Simple Recursive Program

What’s wrong with this approach?

\[ \text{MT}(n,m) \]

\begin{align*}
\text{if } n &= 0 \text{ and } m = 0 \\
\quad &\text{return } 0 \\
\text{if } n &= 0 \\
\quad &\text{return } \text{MT}(0,m-1) + \text{len(edge)} \text{ from } (0,m-1) \text{ to } (0,m) \\
\text{if } m &= 0 \\
\quad &\text{return } \text{MT}(n-1,0) + \text{len(edge)} \text{ from } (n-1,0) \text{ to } (n,0) \\
x &\leftarrow \text{MT}(n-1,m) + \text{len(edge)} \text{ from } (n-1,m) \text{ to } (n,m) \\
y &\leftarrow \text{MT}(n,m-1) + \text{len(edge)} \text{ from } (n,m-1) \text{ to } (n,m) \\
\text{return } \max\{x,y\}
\end{align*}

We saw this in our recursive change algorithm. It computes the same paths multiple times.
MTP: Ordering Evaluations

- 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 connecting edge in between

First, fill in the easy ones! Those 1 block from the source
First, fill in the easy ones!

Then grow the solution a block at a time while tabulating the results for each intersection.

Note: We'll use our table to keep track of two things. The value of the best path to the given intersection, and the direction from where it came.
MTP: Dynamic Programming (cont’d)

Keep growing… (3 blocks)
MTP: Dynamic Programming (cont’d)

And growing…
(4 blocks)

$S_{1,3} = 8$

$S_{2,2} = 12$

$S_{3,1} = 9$
MTP: Dynamic Programming (cont’d)

And growing...
(5 blocks)

$S_{2,3} = 15$

$S_{3,2} = 9$
Once the “destination” node (intersection) is reached, we’re done.

Our table will have the answer of the maximum number of attractions stored in the entry associated with the destination.

We use the “links” back in the table to recover the path. (Backtracking)
MTP: Recurrence

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

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

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

(You visit all intersections once, and performed 2 tests)

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

What about diagonals?
Broadway, Greenwich, etc.

• Easy to fix. Just adds more recursion cases.
• The score at point B is given by:

\[ s_B = \max \begin{cases} 
  s_{A1} + \text{weight of the edge } (A_1, B) \\
  s_{A2} + \text{weight of the edge } (A_2, B) \\
  s_{A3} + \text{weight of the edge } (A_3, B) 
\end{cases} \]
How similar are two strings?

Spell correction
• The user typed “graffe”
Which is closest?
  – graf
  – graft
  – grail
  – giraffe

Computational Biology
• Align two sequences of nucleotides
  AGGCTATCACCTGACCTCCAGGCGGATGCCCTAGCTATCACGACCGCGGTCGATTTGCCCCGAC
• Resulting alignment:
  -AGGCTATCACCTGACCTCCAGGCGGATGCCCTAGCTATCACGACCGCGGTCGATTTGCCCCGAC

Also for Machine Translation, Information Extraction, Speech Recognition
Edit Distance

• The minimum edit distance between two strings
• Is the minimum number of editing operations
  – Insertion
  – Deletion
  – Substitution
• Needed to transform one into the other
Minimum Edit Distance

- Two strings and their alignment:

\[
\text{INTEGRATION} \\
\text{EXECUTION}
\]
Minimum Edit Distance

If each operation has cost of 1
- Distance between these is 5

If substitutions cost 2 (Levenshtein)
- Distance between them is 8
Alignment in Computational Biology

- Given a sequence of bases
  
  \[
  \begin{align*}
  \text{AGGCTATCACCTGACCTCCAGGCGATGCCC} \\
  \text{TAGCTATCACGACCGCGGCGTGATTTGCCCCGAC}
  \end{align*}
  \]

- An alignment:

  \[
  \begin{align*}
  \text{AGGCTATCACCTGACCTCCAGGCGATGCCC} & \text{TGCCC} \\
  \text{TAGCTATCACGACCGCGGCGTGATTTGCCCCGAC} & \text{TAC}
  \end{align*}
  \]

- Given two sequences, align each letter to a letter or gap
How to find the Min Edit Distance?

Searching for a path (sequence of edits) from the start string to the final string:

- **Initial state**: the word we’re transforming
- **Operators**: insert, delete, substitute
- **Goal state**: the word we’re trying to get to
- **Path cost**: what we want to minimize: the number of edits
Minimum Edit as Search

• But the space of all edit sequences is huge!
  – We can’t afford to navigate naïvely
  – Lots of distinct paths wind up at the same state.
    • We don’t have to keep track of all of them
    • Just the shortest path to each of those revisited states.
Defining Min Edit Distance

• For two strings
  – X of length $n$
  – Y of length $m$

• We define $D(i,j)$
  – the edit distance between $X[1..i]$ and $Y[1..j]$
    • i.e., the first $i$ characters of X and the first $j$ characters of Y
  – The edit distance between X and Y is thus $D(n,m)$
Defining Min Edit Distance
(Levenshtein)

Initialization

\[ D(i,0) = i \]
\[ D(0,j) = j \]

Recurrence Relation:

For each \( i = 1 \ldots M \)

For each \( j = 1 \ldots N \)

\[ D(i,j) = \min \begin{cases} 
D(i-1,j) + 1 \\
D(i,j-1) + 1 \\
D(i-1,j-1) + \begin{cases} 
2; & \text{if } X(i) \neq Y(j) \\
0; & \text{if } X(i) = Y(j) 
\end{cases} 
\end{cases} \]

Termination:

\[ D(N,M) \text{ is distance} \]
Min Edit Distance Algorithm

"Calculate Levenstein edit distance for strings s1 and s2."

len1 = len(s1)  # vertically
len2 = len(s2)  # horizontally

# Allocate the table
table = [None]*(len2+1)
for i in range(len2+1): table[i] = [0]*(len1+1)

# Initialize the table
for i in range(1, len2+1): table[i][0] = i
for i in range(1, len1+1): table[0][i] = i

# Do dynamic programming
for i in range(1, len2+1):
    for j in range(1, len1+1):
        if s1[j-1] == s2[i-1]:
            d = 0
        else:
            d = 2
        table[i][j] = min(table[i-1][j-1] + d,
                          table[i-1][j]+1,
                          table[i][j-1]+1)
# The Edit Distance Table

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>E</th>
<th>X</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>T</th>
<th>I</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## The Edit Distance Table

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>E</th>
<th>X</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>T</th>
<th>I</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>#0</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$D(i,j) = \min \begin{cases} 
D(i-1,j) + 1 \\
D(i,j-1) + 1 \\
D(i-1,j-1) + \begin{cases} 
2; & \text{if } S_1(i) \neq S_2(j) \\
0; & \text{if } S_1(i) = S_2(j) 
\end{cases}
\end{cases}$
# Edit Distance

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>E</th>
<th>X</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>T</th>
<th>I</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# The Edit Distance Table

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>E</th>
<th>X</th>
<th>E</th>
<th>C</th>
<th>U</th>
<th>T</th>
<th>I</th>
<th>O</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>N</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>N</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>T</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>I</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>O</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>N</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>
Applications.
• Basis for Unix diff.
• Speech recognition.
• Computational biology.

• Gap penalty $\delta$; mismatch penalty $\alpha_{pq}$.
• Cost = sum of gap and mismatch penalties.

\[
\begin{align*}
\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA} &= 2\delta + \alpha_{CA}
\end{align*}
\]
Goal: Given two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$ find alignment of minimum cost.

Def. An alignment $M$ is a set of ordered pairs $x_i$-$y_j$ such that each item occurs in at most one pair and no crossings.

Def. The pair $x_i$-$y_j$ and $x_{i'}$-$y_{j'}$ cross if $i < i'$, but $j > j'$.

$$\text{cost}(M) = \sum_{(x_i, y_j) \in M} \alpha_{x_i y_j} + \sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta$$

Ex: CTACCG VS. TACATG. Sol: $M = x_2$-$y_1$, $x_3$-$y_2$, $x_4$-$y_3$, $x_5$-$y_4$, $x_6$-$y_6$. 
Sequence Alignment: Problem Structure

**Def.** \( \text{OPT}(i, j) = \text{min cost of aligning strings } x_1 x_2 \ldots x_i \text{ and } y_1 y_2 \ldots y_j. \)

- **Case 1:** OPT matches \( x_i-y_j. \)
  - pay mismatch for \( x_i-y_j \) + \( \text{min cost of aligning two strings } \)
    \( x_1 x_2 \ldots x_{i-1} \text{ and } y_1 y_2 \ldots y_{j-1} \)
- **Case 2a:** OPT leaves \( x_i \) unmatched.
  - pay gap for \( x_i \) and \( \text{min cost of aligning } x_1 x_2 \ldots x_{i-1} \text{ and } y_1 y_2 \ldots y_j \)
- **Case 2b:** OPT leaves \( y_j \) unmatched.
  - pay gap for \( y_j \) and \( \text{min cost of aligning } x_1 x_2 \ldots x_i \text{ and } y_1 y_2 \ldots y_{j-1} \)

\[
\text{OPT}(i, j) = \begin{cases} 
  j \delta & \text{if } i = 0 \\
  \min \begin{cases} 
    \alpha_{x_i, y_j} + \text{OPT}(i-1, j-1) \\
    \delta + \text{OPT}(i-1, j) \\
    \delta + \text{OPT}(i, j-1) \\
    i \delta 
  \end{cases} & \text{otherwise} \\
  i \delta & \text{if } j = 0 
\end{cases}
\]

2/27/2014
Sequence Alignment: Algorithm

Sequence-Alignment(m, n, \(x_1x_2\ldots x_m\), \(y_1y_2\ldots y_n\), \(\delta\), \(\alpha\)) {
    \text{for } i = 0 \text{ to } m
    M[0, i] = i\delta
    \text{for } j = 0 \text{ to } n
    M[j, 0] = j\delta

    \text{for } i = 1 \text{ to } m
        \text{for } j = 1 \text{ to } n
            M[i, j] = \min(\alpha[x_i, y_j] + M[i-1, j-1],$
                       \delta + M[i-1, j],$
                       \delta + M[i, j-1])

    \text{return } M[m, n]
}

• Analysis. \(\Theta(mn)\) time and space.
• English words or sentences: \(m, n \leq 10\).
• Computational biology: \(m = n = 100,000\). 10 billions ops OK, but 10GB array?
Q. Can we avoid using quadratic space?

Easy. Optimal value in $O(m + n)$ space and $O(mn)$ time.

- Compute $OPT(i, \bullet)$ from $OPT(i-1, \bullet)$.
- No longer a simple way to recover alignment itself.

Theorem. [Hirschberg 1975] Optimal alignment in $O(m + n)$ space and $O(mn)$ time.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.
Edit distance graph.

- Let $f(i, j)$ be the shortest path from $(0,0)$ to $(i, j)$.
- Observation: $f(i, j) = \text{OPT}(i, j)$.
Sequence Alignment: Linear Space

Edit distance graph.

- Let $f(i, j)$ be shortest path from $(0,0)$ to $(i, j)$.
- Can compute $f(\bullet, j)$ for any $j$ in $O(mn)$ time and $O(m + n)$ space.
Space-Efficient Alignment Algorithm

Space-Efficient-Alignment(X,Y)

Array $B[0 \ldots m, 0 \ldots 1]$

Initialize $B[i,0] = i \delta$ (just as in column 0 of A)

For $j = 1, \ldots, n$

$B[0,1] = j \delta$ (since this corresponds to entry $A[0,j]$)

For $i = 1, \ldots, m$

$B[i,1] = \min[\alpha x_i, y_j + B[i-1,0],$

$\delta + B[i-1,1], \delta + B[i,0]]$.

Endfor

Move column 1 of B to column 0 to make room for next iteration:

Update $B[i,0] = B[i,1]$ for each $i$

Endfor
Sequence Alignment: Linear Space

Edit distance graph.

- Let $g(i, j)$ be shortest path from $(i, j)$ to $(m, n)$.
- Can compute by reversing the edge orientations and inverting the roles of $(0, 0)$ and $(m, n)$

![Diagram of edit distance graph](image-url)
Sequence Alignment: Linear Space

Edit distance graph.

- Let $g(i, j)$ be shortest path from $(i, j)$ to $(m, n)$.
- Can compute $g(\bullet, j)$ for any $j$ in $O(mn)$ time and $O(m + n)$ space.
Observation 1. The cost of the shortest path that uses \((i, j)\) is 
\[ f(i, j) + g(i, j). \]
Observation 2. Let $q$ be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, the shortest path from $(0, 0)$ to $(m, n)$ uses $(q, n/2)$. 
Sequence Alignment: Linear Space

Divide: find index q that minimizes $f(q, n/2) + g(q, n/2)$ using DP.
• Align $x_q$ and $y_{n/2}$.
Conquer: recursively compute optimal alignment in each piece.
Theorem. Let $T(m, n) = \text{max running time of algorithm on strings of length at most } m \text{ and } n$. $T(m, n) = O(mn \log n)$.

Remark. Analysis is not tight because two sub-problems are of size $(q, n/2)$ and $(m - q, n/2)$. In next slide, we save $\log n$ factor.
Sequence Alignment: Running Time Analysis

**Theorem.** Let $T(m, n) = \text{max running time of algorithm on strings of length } m \text{ and } n$. $T(m, n) = O(mn)$.

**Pf.** (by induction on $n$)
- $O(mn)$ time to compute $f(\bullet, n/2)$ and $g(\bullet, n/2)$ and find index $q$.
- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.
- Choose constant $c$ so that:
  
  \[
  \begin{align*}
  T(m, 2) & \leq cm \\
  T(2, n) & \leq cn \\
  T(m, n) & \leq cmn + T(q, n/2) + T(m - q, n/2)
  \end{align*}
  \]

- Base cases: $m = 2$ or $n = 2$.
- Inductive hypothesis: $T(m, n) \leq 2cmn$.

\[
\begin{align*}
T(m,n) & \leq T(q,n/2) + T(m - q, n/2) + cmn \\
& \leq 2cqn/2 + 2c(m - q)n/2 + cmn \\
& = cqn + cmn - cqn + cmn \\
& = 2cmn
\end{align*}
\]
Divide-and-Conquer Alignment Algorithm

Divide-and-Conquer-Alignment\((X,Y)\)

Let \(m\) be the number of symbols in \(X\)
Let \(n\) be the number of symbols in \(Y\)
If \(m \leq 2\) or \(n \leq 2\) then
    Compute optimal alignment using Alignment\((X,Y)\)
Call Space-Efficient-Alignment\((X,Y[1:n/2])\),
    obtaining array \(B\)
Call Backward-Space-Efficient-Alignment\((X,Y[n/2 + 1:n])\),
    obtaining array \(B'\)
Let \(q\) be the index minimizing \(B[q,1] + B'[q,n]\)
Add \((q,n/2)\) to global list \(P\)
Divide-and-Conquer-Alignment\((X[1:q],Y[1:n/2])\)
Divide-and-Conquer-Alignment\((X[q+1:n],Y[n/2+1:n])\)
Return \(P\)
Practice

Find the Levenshtein minimum edit distance of the words `ocurrance` and `occurrence`.

Assume a mismatch penalty of 2, and a gap (indel) penalty of 1.
Next Time

• Network Flow: Basic Definitions