CS3000: Algorithms & Data
Jonathan Ullman

Lecture 9:
• Dynamic Programming: Edit Distance, RNA Folding

Oct 5, 2018
<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon</td>
<td>4:30-6:30</td>
</tr>
<tr>
<td>Tue</td>
<td>5:00-7:00</td>
</tr>
<tr>
<td>Wed</td>
<td>8:00-5:00</td>
</tr>
<tr>
<td>Thu</td>
<td>12:00-2:00, 5:00-7:00</td>
</tr>
</tbody>
</table>
Edit Distance Alignments
Distance Between Strings

• Autocorrect works by finding similar strings

  ![Google Search Screenshot]

  Did you mean: occurrence

• *ocurrance* and *occurrence* seem similar, but only if we define similarity carefully.

  ![Similarity Diagram]

  7 mismatches

  2 mismatches
Edit Distance / Alignments

• Given two strings $x \in \Sigma^n, y \in \Sigma^m$, the edit distance is the number of insertions, deletions, and swaps required to turn $x$ into $y$.

$$\text{Edit Dist} = \text{Minimum Cost Alignment}$$

• Given an alignment, the cost is the number of positions where the two strings don’t agree.

Cost of the alignment is the # of columns where the two symbols disagree.
Ask the Audience

• What is the minimum cost alignment of the strings **smitten** and **sitting**
Edit Distance / Alignments

- **Input:** Two strings \( x \in \Sigma^n, y \in \Sigma^m \)
- **Output:** The minimum cost alignment of \( x \) and \( y \)
  - **Edit Distance** = cost of the minimum cost alignment

![Diagram](image)
Dynamic Programming

• Consider the **optimal** alignment of \( x, y \)

• Three choices for the final column
  • **Case I:** only use \( x \) (\( x_n, - \))
  • **Case II:** only use \( y \) (\( -, y_m \))
  • **Case III:** use one symbol from each (\( x_n, y_m \))
Consider the **optimal** alignment of $x, y$

**Case I:** only use $x$ ($x_n, -$)
  - deletion + optimal alignment of $x_{1:n-1}, y_{1:m}$

**Case II:** only use $y$ ($-, y_m$)
  - insertion + optimal alignment of $x_{1:n}, y_{1:m-1}$

**Case III:** use one symbol from each ($x_n, y_m$)
  - If $x_n = y_m$: optimal alignment of $x_{1:n-1}, y_{1:m-1}$
  - If $x_n \neq y_m$: mismatch + opt. alignment of $x_{1:n-1}, y_{1:m-1}$

To decide which case is the best, I need to know the edit distance between $x[1:i]$ and $y[1:j]$.
Dynamic Programming

0 ≤ i ≤ n   0 ≤ j ≤ m ⇒ O(nm) problems

• \(\text{OPT}(i, j)\) = cost of opt. alignment of \(x_{1:i}\) and \(y_{1:j}\)

  Case I: only use \(x\) (\(x_i, -\))  \(1 + \text{OPT}(i-1, j)\)

  Case II: only use \(y\) (\(-, y_j\))  \(1 + \text{OPT}(i, j-1)\)

  Case III: use one symbol from each (\(x_i, y_j\))

\[
\text{OPT}(i, j) = \begin{cases} 
\text{OPT}(i-1, j-1) + 1 & \text{if } x_i \neq y_j \\
\text{OPT}(i-1, j-1) & \text{if } x_i = y_j \\
m_{m} \left\{ \begin{array}{ll}
1 + \text{OPT}(i-1, j), & 1 + \text{OPT}(i, j-1), & \text{OPT}(i-1, j-1)
\end{array} \right\} & x_i \neq y_j 
\end{cases}
\]
Dynamic Programming

• \( \text{OPT}(i, j) = \) cost of opt. alignment of \( x_{1:i} \) and \( y_{1:j} \)
• **Case I:** only use \( x \) ( \( x_i, - \) )
• **Case II:** only use \( y \) ( \( -, y_j \) )
• **Case III:** use one symbol from each ( \( x_i, y_j \) )

**Recurrence:**
\[
\text{OPT}(i, j) = \begin{cases} 
1 + \min \{ \text{OPT}(i - 1, j), \text{OPT}(i, j - 1), \text{OPT}(i - 1, j - 1) \} \\
\min \{ 1 + \text{OPT}(i - 1, j), 1 + \text{OPT}(i, j - 1), \text{OPT}(i - 1, j - 1) \}
\end{cases}
\]

**Base Cases:**
\[
\text{OPT}(i, 0) = i, \text{OPT}(0, j) = j
\]
Example

\[ x = \text{pert} \]
\[ y = \text{beast} \]

Edit dist. of "beast" and "p"

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>e</th>
<th>a</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>p</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>r</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>t</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Finding the Alignment

- \( \text{OPT}(i, j) = \text{cost of opt. alignment of } x_{1:i} \text{ and } y_{1:j} \)
- **Case I:** only use \( x \) ( \( x_i, - \) )
- **Case II:** only use \( y \) ( \( -, y_j \) )
- **Case III:** use one symbol from each ( \( x_i, y_j \) )
Edit Distance ("Bottom-Up")

// All inputs are global vars
FindOPT(n,m):
    M[0,j] ← j, M[i,0] ← i

    for (i = 1,...,n):
        for (j = 1,...,m):
            if (xᵢ = yⱼ):
                M[i,j] = min{1+M[i-1,j],1+M[i,j-1],M[i-1,j-1]}
            elseif (xᵢ ≠ yⱼ):
                M[i,j] = 1+min{M[i-1,j],M[i,j-1],M[i-1,j-1]}

    return M[n,m]

\[ \begin{bmatrix} (n+1)(m+1) \text{ entries} \end{bmatrix} \times \begin{bmatrix} O(1) \text{ operations per entry} \end{bmatrix} = O(nm) \text{ time} \]

Space is also \( O(nm) \)
• Suppose **inserting/deleting costs** $\delta > 0$ and **swapping** $a \leftrightarrow b$ **costs** $c_{a,b} > 0$

• Write a recurrence for the min-cost alignment
Edit Distance Summary

• Compute the edit distance, or min-cost alignment between two strings in time/space $O(nm)$

• Dynamic Programming:
  • Decide the final pair of symbols in the alignment

• Space can be prohibitive in practice
  • Compute edit distance in space $O(\min\{n, m\})$
  • Can also find alignment in space $O(n + m)$ using a clever divide-and-conquer algorithm!
RNA Folding
DNA

- DNA is a string of four bases \{A, C, G, T\}
- Two complementary strands of DNA stick together and form a **double helix**
  - A—T and C—G are complementary pairs
RNA Folding

- RNA is a string of four bases \{A,C,G,U\}
- A single RNA strand sticks to itself and folds into complex structures
  - A—U and C—G are complementary pairs
RNA Folding

• RNA strand will try to **minimize energy** (form the most bonds) subject to **constraints**
RNA Folding

- RNA is a string of bases \( b_1, \ldots, b_n \in \{A, C, G, U\} \)
- The structure is given by a set of bonds \( S \) consisting of pairs \((i, j)\) with \( i < j \)
  - (Complements) Only \( A - U \) or \( C - G \) can be paired
  - (Matching) No base \( b_i \) is in two pairs in \( S \)
  - (No Sharp Turns) If \((i, j) \in S\), then \( i < j - 4 \)
  - (Non-Crossing) If \((i, j), (k, \ell) \in S\) then it cannot be the case that \( i < k < j < \ell \)
RNA Folding

• **Input:** RNA sequence \( b_1, \ldots, b_n \in \{A, C, G, U\} \)

• **Output:** A set of pairs \( S \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\} \)
  
  - **Goal:** maximize the size of \( S \)
  
  - **(Complements)** Only \( A \rightarrow U \) or \( C \rightarrow G \) can be paired
  
  - **(Matching)** No base \( b_i \) is in two pairs in \( S \)
  
  - **(No Sharp Turns)** If \( (i, j) \in S \), then \( i < j - 4 \)
  
  - **(Non-Crossing)** If \( (i, j), (k, \ell) \in S \) then it cannot be the case that \( i < k < j < \ell \)
Dynamic Programming

• Let $O$ be the optimal set of pairs for $b_1 \cdots b_n$
• **Case 1:** $n$ pairs with nothing in $O$
  
  $O$ is the optimal set of pairs for $b_1 \cdots b_{n-1}$

• **Case 2:** $n$ pairs with some $t < n - 4$ in $O$
  
  $O$ is opt for $b_1 \cdots b_{t-1}$
  + opt for $b_{t+1} \cdots b_{n-1}$
  + $(t,n)$

---

![Diagram showing dynamic programming concepts](image)
Dynamic Programming

• Let $O_{i,j}$ be the optimal set of pairs for $b_i \cdots b_j$

• **Case 1:** $j$ pairs with nothing in $O_{i,j}$

• **Case 2:** $j$ pairs with some $t < j - 4$ in $O_{i,j}$
Dynamic Programming

- Let $\text{OPT}(i, j)$ be the opt. **number** of pairs for $b_i \cdots b_j$

- **Case 1:** $j$ pairs with nothing in $O_{i,j}$
  \[
  \text{OPT}(i,j) = \text{OPT}(i,j-1)
  \]

- **Case 2:** $j$ pairs with $t < j - 4$ in $O_{i,j}$
  - $\text{OPT}(i,j) = 1 + \text{OPT}(t+1,j-1) + \text{OPT}(1,t-1)$
  - Consider all $i \leq t < j-4$ s.t. $b_t, b_j$ are complements
Dynamic Programming

• Let OPT\((i, j)\) be the opt. number of pairs for \(b_i \cdots b_j\)

• **Case 1:** \(j\) pairs with nothing in \(O_{i,j}\)

• **Case 2:** \(j\) pairs with \(t < j - 4\) in \(O_{i,j}\)

**Recurrence:**
\[
\text{OPT}(i, j) = \max\{\text{OPT}(i, j - 1), \max\{\text{OPT}(i, t - 1) + \text{OPT}(t + 1, j - 1)\}\}\]

**Base Cases:**
\[
\text{OPT}(i, j) = 0 \text{ if } i \geq j - 4
\]

**Because of no-sharp turns**
Filling the Table

Sequence: $ACCGGUAGU$

Recurrence:

$$OPT(i, j) = \max \left\{ OPT(i, j - 1), \max_{\text{possible } t} \{ OPT(i, t - 1) + OPT(t + 1, j - 1) \} \right\}$$

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>$j = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i = 1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RNA Folding Summary

• Compute the **optimal RNA folding** in time $O(n^3)$ and space $O(n^2)$

• **Dynamic Programming:**
  • Decide on an optimal pair $b_t - b_n$
  • Remaining RNA is two non-overlapping pieces
  • **Adding variables:** one subproblem for each interval

• **Non-crossing** and **matching** are critical
  • Think about how the dynamic programming algorithm changes if we remove each of the conditions