CSE 421
Introduction to Algorithms

Lecture 13: Dynamic Programming
RNA folding, Sequence Alignment
Dynamic Programming for Optimization

1. Formulate the *(optimum) value* as a recurrence relation or recursive algorithm.

2. Figure out the possible values of parameters in the recursive calls.
   - This should be “small”, i.e., bounded by a low-degree polynomial.
   - Can use memoization to store a cache of previously computing values.

3. Specify an order of evaluation for the recurrence so that you already have the partial results stored in memory when you need them.
   - Produces iterative code.
   - Store extra information to be able to reconstruct *optimal solution* and add reconstruction code.

Once you have an iterative DP solution: see if you can save space.
Dynamic Programming Patterns so far

Fibonacci pattern:
- 1-D, $O(1)$ immediately prior
- $O(1)$ space

Weighted interval scheduling pattern:
- 1-D, $O(1)$ arbitrary prior
- $O(n)$ space

Longest increasing subsequence pattern:
- 1-D, all $n - 1$ prior
- $O(n)$ space

Knapsack pattern:
- 2-D, $O(1)$ elements in previous row, above and arbitrary far to the left
- $O(nW)$ space

- $O(W)$ space if only optimum value needed
  - Maintain current and previous rows
Dynamic Programming over Intervals

In this different class of problems from ones we have seen before, there are

• 1-dimensional inputs
• A notion of optimization over intervals in that 1 dimension

A number of important problems fit this paradigm

• We focus on a version of one these: RNA Secondary Structure
RNA Secondary Structure

RNA (ribonucleic acid): String $B = b_1 b_2 \cdots b_n$ of bases over alphabet \{A, C, G, U\} standing for adenine, cytosine, guanine, and uracil.

RNA Secondary Structure: RNA is single-stranded and tends to loop back and form bonds between pairs of its bases “base-pairs”. This structure is essential for understanding behavior of the molecule.

Ex: GUCGAUUGAGCGAAUGUAACACGUGGCUCACGGCGAGA

complementary base pairs: A-U, C-G
RNA Secondary Structure

Defn: A *secondary structure* for an RNA sequence $B = b_1 b_2 \cdots b_n$ is a set of pairs $S = \{(b_i, b_j)\}$ that satisfy:

- **[Watson-Crick condition]** $S$ is a matching and each pair in $S$ is a Watson-Crick complement: $A-U,$ $U-A,$ $C-G,$ or $G-C.$
- **[No sharp bends]** The ends of each pair are separated by at least 4 intervening bases. That is, if $(b_i, b_j) \in S,$ then $i < j - 4.$
- **[Non-crossing]** If $(b_i, b_j)$ and $(b_k, b_\ell)$ are two pairs in $S,$ then we cannot have $i < k < j < \ell.$

**Optimizing energy:** The usual hypothesis is that an RNA molecule will form a secondary structure that optimizes the total free energy. Maximizing the # of base pairs in $S$ roughly maximizes free energy.

**Given:** an RNA molecule $B = b_1 b_2 \cdots b_n,$

**Find:** a secondary structure $S$ for $B$ maximizing the number of base pairs in $S.$
RNA Secondary Structure: Examples

Examples.

- Base pair
- Sharp bend
- Crossing
RNA Secondary Structure: False Start

As usual we consider two cases based on the status of the last base in an optimal secondary structure.

**First attempt:** Define $\text{OPT}(j) =$ maximum # of base pairs in a secondary structure of the substring $b_1b_2 \cdots b_j$.

Case 1: $\text{OPT}$ does not match base $b_j$. Value is $\text{OPT}(j - 1)$.

Case 2: $\text{OPT}$ contains some base pair $(b_k, b_j)$.

Two independent* subproblems:

- One on $b_1b_2 \cdots b_{k-1}$ with value $\text{OPT}(k - 1)$
- One on $b_{k+1}b_2 \cdots b_{j-1}$
  - Not of the same type: Need to allow starting index $\neq 1$

* Independence guaranteed by non-crossing property
RNA Secondary Structure: DP over Intervals

**Defn:** Define $\text{OPT}(i, j) = \text{maximum \# of base pairs in a secondary structure of the substring } b_i b_2 \ldots b_j$.

Case 1: $\text{OPT}$ does not match base $b_j$. Value is $\text{OPT}(i, j - 1)$.

Case 2: $\text{OPT}$ contains some base pair $(b_k, b_j)$.

Two independent subproblems:

- One on $b_i b_2 \ldots b_{k-1}$ with value $\text{OPT}(i, k - 1)$
- One on $b_{k+1} b_2 \ldots b_{j-1}$ with value $\text{OPT}(k + 1, j - 1)$

$$
\text{OPT}(i, j) = \begin{cases} 
0 & \text{if } j \leq i + 4 \\
\max\{\text{OPT}(i, j - 1), \max\{1 + \text{OPT}(i, k - 1) + \text{OPT}(k + 1, j - 1) : j > k + 4 \}, b_k \sim b_j\} & \text{if } j > i + 4
\end{cases}
$$

where we write $b \sim b'$ iff they are Watson-Crick complement pairs $A-U, U-A, C-G, \text{or } G-C$. Intervals for recursive calls are shorter.
Dynamic Programming Over Intervals: Iterative Solution

Evaluate in order of increasing interval length

```c
RNA(b_1, \ldots, b_n) {  // interval length
  for m = 0 to n-1
    // interval start
    j = i + m
    if m < 5
      OPT[i, j] = 0
    else {
      OPT[i, j] = OPT[i, j-1]
      for k = i to j-5  // split point
        if WatsonCrick(b_k, b_j)
          if 1+OPT[i, k-1] + OPT[k+1, j-1] > OPT[i, j] {
            OPT[i, j] = 1 + OPT[i, k-1] + OPT[k+1, j-1]
          }
    }
  return OPT[1, n]
}
```
Dynamic Programming Over Intervals: Iterative Solution

DP over intervals pattern
- 2-D lower triangular portion
- Fill sub-diagonals in order of distance from the diagonal
- Each of the $O(n^2)$ entries uses $O(n)$ pairs of entries in
  - a fixed row to the left and
  - a column above
- Time $O(n^3)$, space $O(n^2)$
Sequence Alignment
String Similarity

How similar are two strings?

• occurance
• occurrence

Clearly a better matching

Maybe a better matching
• depends on cost of gaps vs mismatches

6 mismatches, 1 gap

1 mismatch, 1 gap

0 mismatches, 3 gaps
Edit Distance

Applications:
• Basis for Unix \texttt{diff}.
• Speech recognition.
• Computational biology.

Edit distance: [Levenshtein 1966, Needleman-Wunsch 1970]
• Gap penalty \( \delta \); mismatch penalty \( \alpha_{pq} \) if symbol \( p \) is replaced by symbol \( q \).
• Cost = gap penalties + mismatch penalties.

\[
\begin{align*}
\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA} & \quad C \quad T \quad G \quad A \quad C \quad C \quad T \\
2\delta + \alpha_{CA} & \quad C \quad C \quad T \quad G \quad A \quad C \quad T
\end{align*}
\]
Sequence Alignment

Sequence Alignment:

**Given:** Two strings $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$

**Find:** “Alignment” of $X$ and $Y$ of minimum edit distance.

**Defn:** An alignment $M$ of $X$ and $Y$ is a set of ordered pairs $x_i$-$y_j$

\[ \text{s.t. each symbol of } X \text{ and } Y \text{ occurs in at most one pair} \]

with no “crossing pairs”.

The pairs $x_i$-$y_j$ and $x_{i'}$-$y_{j'}$ cross iff $i < i'$ but $j > j'$.

**Example:**

CTACCG vs TACATG

Note: if $x_i = y_j$ then $\alpha_{x_iy_j} = 0$
Sequence Alignment: Problem Structure

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

Case 1: \( \text{OPT} \) matches \( x_i - y_j. \)

- Pay mismatch cost \( \alpha_{x_i y_j} \) for \( x_i - y_j \) + min cost of aligning strings \( x_1 x_2 ... x_{i-1} \) and \( y_1 y_2 ... y_{j-1} \)

Note: if \( x_i = y_j \) then \( \alpha_{x_i y_j} = 0 \)

Case 2a: \( \text{OPT} \) leaves \( x_i \) unmatched.

- Pay gap cost \( \delta \) for \( x_i \) + min cost of aligning \( x_1 x_2 ... x_{i-1} \) and \( y_1 y_2 ... y_j \)

Case 2b: \( \text{OPT} \) leaves \( y_j \) unmatched.

- Pay gap cost \( \delta \) for \( y_j \) + min cost of aligning \( x_1 x_2 ... x_i \) and \( y_1 y_2 ... y_{j-1} \)

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

```
Sequence-Alignment(m, n, x_1x_2...x_m, y_1y_2...y_n, δ, α) {
    for i = 0 to m
        OPT[i, 0] = i δ
    for j = 0 to n
        OPT[0, j] = j δ
    for i = 1 to m
        for j = 1 to n
            OPT[i, j] = min(α[x_i, y_j] + OPT[i-1, j-1],
                             δ + OPT[i-1, j],
                             δ + OPT[i, j-1])
    return OPT[m, n]
}
```

\( O(mn) \)
Example run with \textit{AGACATTTG} and \textit{GAGTTA}: $\delta = \alpha_{\text{mis}} = 1$
Example run with **AGACATTG** and **GAGTTA**: $\delta = \alpha_{\text{mis}} = 1$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>5</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>
</tr>
</tbody>
</table>
Example run with **AGACATTTG** and **GAGTTA**: \( \delta = \alpha_{\text{mis}} = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</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>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example run with **AGACATTG** and **GAGTTA**: \( \delta = \alpha_{\text{mis}} = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<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>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example run with \textbf{AGACATTG} and \textbf{GAGTTA}: $\delta = \alpha_{\text{mis}} = 1$

\begin{table}[h]
\centering
\begin{tabular}{cccccccc}
  & A & G & A & C & A & T & T & G \\
G & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
A & 1 & 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
G & 2 & 1 & 2 & 1 & 2 & 3 & 4 & 5 & 6 \\
A & 3 & 2 & 1 & 2 & 2 & 3 & 4 & 5 & 5 \\
G & 4 & 3 & 2 & 2 & 3 & 3 & 3 & 4 & 5 \\
T & 5 & 4 & 3 & 3 & 3 & 4 & 3 & 3 & 4 \\
A & 6 & 5 & 4 & 3 & 4 & 3 & 4 & 4 & 4 \\
\end{tabular}
\end{table}
Example run with **AGACATTG** and **GAGTTA**: $\delta = \alpha_{\text{mis}} = 1$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>A</th>
<th>C</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</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>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Example run with $\text{AGACATTG}$ and $\text{GAGTTA}$: $\delta = \alpha_{\text{mis}} = 1$

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

Optimal Alignment

$\text{AGACATTG}$

$\text{GAGTTA}$
Genbank and WGS Statistics
Sequence Alignment: Linear Space

- Lines of code for **diff**: \( m, n \) at most in 1000’s
- Computational biology: \( m, n \) may be in 100,000’s.
  
  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 \( \text{OPT}(i, \bullet) \) from \( \text{OPT}(i - 1, \bullet) \).
  
  • 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.
Sequence Alignment: Linear Space

**Edit distance graph:** Horizontal & vertical edges weight $\delta$

Diagonal edge into each node $(i, j)$ weight $\alpha_{x_iy_j}$
Sequence Alignment: Linear Space

**Edit distance graph:** Horizontal & vertical edges weight $\delta$

Diagonal edge into each node $(i, j)$ weight $\alpha_{x_i y_j}$

Let $d_{\text{start}}(i, j) =$ length of shortest path from $(0, 0)$ to $(i, j)$

Then $\text{OPT}(i, j) = d_{\text{start}}(i, j)$.

For any fixed $j$ can compute all $d_{\text{start}}(\cdot, j)$ in $O(n + m)$ space $O(nm)$ time
**Sequence Alignment: Linear Space**

**Reversed edit distance graph:** Horizontal & vertical edges weight $\delta$

Diagonal edge into each node $(i, j)$ weight $\alpha_{x_{i+1}y_{j+1}}$
Sequence Alignment: Linear Space

Reversed edit distance graph: Horizontal & vertical edges weight $\delta$

Diagonal edge into each node $(i, j)$ weight $\alpha_{x_{i+1}y_{j+1}}$

Let $d_{end}(i, j) =$ length of shortest path from $(m, n)$ to $(i, j)$

For any fixed $j$ can compute all $d_{end}(i, j)$ in $O(n + m)$ space $O(nm)$ time
Sequence Alignment: Linear Space

Edit distance graph: Horizontal & vertical edges weight $\delta$

Diagonal edge into each node $(i, j)$ weight $\alpha_{xiyj}$

Optimal alignment includes exactly one node $(i, j)$ in column $j$

That node minimizes $d_{\text{start}}(i, j) + d_{\text{end}}(i, j)$

which equals $\text{OPT}(m, n)$

Divide & conquer:
Find this for $j = n/2$ and recurse
Sequence Alignment: Linear Space

**Edit distance graph:** Horizontal & vertical edges weight \( \delta \)

Diagonal edge into each node \((i, j)\) weight \(\alpha_{x_iy_j}\)

Optimal alignment includes exactly one node \((i, j)\) in column \(j\)

That node minimizes \(d_{\text{start}}(i, j) + d_{\text{end}}(i, j)\) which equals \(\text{OPT}(m, n)\)

Divide & Conquer:
Find this for \(j = n/2\) and recurse
Re-use space for second call.
Analytical details

Write $T(m, n)$ for the time cost.

• Recurrence $T(m, n) = T(i, n/2) + T(m - i, n/2) + O(mn)$

\[
T(1, n) = O(n), \quad T(m, 1) = O(m)
\]

• Solution $T(m, n) = O(mn)$.
  • Not only is the value of $n$ halved for the two subproblems, but the lengths of the first strings still only sum to $m$.
  • Proof via induction (Exercise).
Another side of practice

In practice the algorithm is usually run on smaller chunks of a large string, e.g. $m$ and $n$ are lengths of genes so a few thousand characters

- Researchers want all alignments that are close to optimal not just the optimal solution
- Basic algorithm is run with
  - 2 rows/columns for values as in the space-saving solution, but
  - all $mn$ pointers since the whole table of pointers (2 bits each) will fit in RAM