CS 466
Introduction to Bioinformatics

Instructor: Jian Peng
Teaching Assistant: Wesley Qian
### BLOcks SUbstitution Matrix (BLOSUM)

<table>
<thead>
<tr>
<th></th>
<th>Ala</th>
<th>Arg</th>
<th>Asn</th>
<th>Asp</th>
<th>Cys</th>
<th>Gln</th>
<th>Glu</th>
<th>Gly</th>
<th>His</th>
<th>Ile</th>
<th>Leu</th>
<th>Lys</th>
<th>Met</th>
<th>Phe</th>
<th>Pro</th>
<th>Ser</th>
<th>Thr</th>
<th>Trp</th>
<th>Tyr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ala</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arg</td>
<td>-1</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asn</td>
<td>-2</td>
<td>0</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asp</td>
<td>-2</td>
<td>-2</td>
<td>1</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cys</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gln</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glu</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<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>
<td></td>
<td></td>
</tr>
<tr>
<td>Gly</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>2</td>
<td>2</td>
<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>
<td></td>
</tr>
<tr>
<td>His</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<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>Ile</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leu</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
<td>2</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>Lys</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>1</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>-3</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Met</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phe</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pro</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ser</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thr</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trp</td>
<td>-3</td>
<td>-3</td>
<td>-4</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tyr</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Val</td>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>-1</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>-1</td>
</tr>
</tbody>
</table>

**amino acids**
Recursion for generalized edit distance

\[
\text{OPT}(i, j) = \max \begin{cases} 
\text{score}(x_i, y_j) + \text{OPT}(i - 1, j - 1) \\
\text{s}_{\text{gap}} + \text{OPT}(i - 1, j) \\
\text{s}_{\text{gap}} + \text{OPT}(i, j - 1) 
\end{cases}
\]

Complexity?
Categories of pairwise alignments

Global: Require an end-to-end alignment of \( x, y \)

Semi-global (glocal): Gaps at the beginning or end of \( x \) or \( y \) are free — useful when one string is significantly shorter than the other or for finding overlaps between strings

Local: Find the highest scoring alignment between \( x' \) a substring of \( x \) and \( y' \) a substring of \( y \) — useful for finding similar regions in strings that may not be globally similar
The recurrence remains the same, we only change the base case of the recurrence and the origin of the backtrack

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Change in Base Case</th>
<th>Change in Traceback</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Ignore gaps before ( x )</td>
<td>( \text{OPT}(0,j) = 0 )</td>
<td></td>
</tr>
<tr>
<td>2)</td>
<td>Ignore gaps after ( x )</td>
<td>( \text{change traceback; start from max } \text{OPT}(n,j) )</td>
<td>( 0 &lt; j \leq m )</td>
</tr>
<tr>
<td>3)</td>
<td>Ignore gaps before ( y )</td>
<td>( \text{OPT}(i,0) = 0 )</td>
<td></td>
</tr>
<tr>
<td>4)</td>
<td>Ignore gaps after ( y )</td>
<td>( \text{change traceback; start from max } \text{OPT}(i,m) )</td>
<td>( 0 &lt; i \leq n )</td>
</tr>
</tbody>
</table>
Semi-global alignment

What is the **same** and **different** between the “global” and semi-global alignment problems?
*assuming \(|y| < |x|\) and we are “fitting” \(y\) into \(x\)

<table>
<thead>
<tr>
<th>Global</th>
<th>Semi-global (“fitting”)</th>
</tr>
</thead>
</table>
| \(\text{OPT}(i,j) = \max\) \begin{cases} 
\text{score}(x_i, y_j) + \text{OPT}(i-1, j-1) \\
\text{s}_{\text{gap}} + \text{OPT}(i-1, j) \\
\text{s}_{\text{gap}} + \text{OPT}(i, j-1) \end{cases} 
| \(\text{OPT}(i,j) = \max\) \begin{cases} 
\text{score}(x_i, y_j) + \text{OPT}(i-1, j-1) \\
\text{s}_{\text{gap}} + \text{OPT}(i-1, j) \\
\text{s}_{\text{gap}} + \text{OPT}(i, j-1) \end{cases} |

Base case: \(\text{OPT}(i,0) = i \times \text{s}_{\text{gap}}\)

Base case: \(\text{OPT}(i,0) = 0\)

Traceback starts at \(\text{OPT}(n,m)\)

Traceback starts at \(\max_{0<j<n} \text{OPT}(j,m)\)
Local alignment: naive algorithm

• Long run time $O(n^4)$:
  - In the grid of size $n \times n$ there are $n^2$ vertices $(i,j)$ that may serve as a source.
  - For each such vertex computing alignments from $(i,j)$ to $(i',j')$ takes $O(n^2)$ time.

• This can be remedied by allowing every point to be the starting point
Local alignment: Smith-Waterman algorithm

Local alignment between a and b: Best alignment between a subsequence of a and a subsequence of b.

$$\text{OPT}(i, j) = \max \begin{cases} 
\text{score}(x_i, y_j) + \text{OPT}(i - 1, j - 1), \\
\text{s}_{\text{gap}} + \text{OPT}(i - 1, j), \\
\text{s}_{\text{gap}} + \text{OPT}(i, j - 1), \\
0 
\end{cases}$$

Idea: start over from any entry!
Local alignment

- Initialize first row and first column to be 0.

- The score of the best local alignment is the largest value in the entire array.

- To find the actual local alignment:
  - start at an entry with the maximum score
  - traceback as usual
  - stop when we reach an entry with a score of 0
<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<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>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>3</td>
<td>20</td>
<td>13</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>13</td>
<td>30</td>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>6</td>
<td>23</td>
<td>25</td>
<td>18</td>
</tr>
</tbody>
</table>

Score(match) = 10
Score(mismatch) = -5
Score(gap) = -7
```python
local_align("catdogfish", "dog")
```

```
  *  c  a  t  d  o  g  f  i  s  h
  *  0  0  0  0  0  0  0  0  0  0  0
  d  0  0  0  0  0  10  3  0  0  0  0
  o  0  0  0  0  3  20  13  6  0  0  0
  g  0  0  0  0  0  0  13  30  23  16  9  2
```

```python
local_align("mississippi", "issp")
```

```
  *  m  i  s  s  s  i  p  p  p  i
  *  0  0  0  0  0  0  0  0  0  0  0
  i  0  0  10  3  0  0  10  3  0  10  3  0  10
  s  0  0  3  20  13  6  20  13  6  5  0  3
  s  0  0  0  13  30  23  16  30  23  16  9  2
  p  0  0  0  6  23  25  18  23  25  33  26  19
```
**Gap score/penalty**

\[
\begin{align*}
\text{AAAGAATTCA} & \quad \text{vs.} \quad \text{AAAGAATTCA} \\
\text{A-A-A-T-C-A} & \quad \text{AAA-----TCA}
\end{align*}
\]

These have the same score, but the second one is often more plausible.

A single insertion of “GAAT” into the first string could change it into the second — Biologically, this is much more likely as \(x\) could be transformed into \(y\) in “one fell swoop”.

- Currently, the score of a run of \(k\) gaps is \(s_{gap} \times k\).

- It might be more realistic to support general gap penalty, so that the score of a run of \(k\) gaps is \(g_{score}(k) \mid < \mid (s_{gap} \times k)\).

- Then, the optimization will prefer to group gaps together.
Affine gap penalty

- We encourage spaces to group together using a special case of general penalties called **affine gap penalties**:
  
  \[ g_{\text{start}} = \text{the cost of starting a gap} \]
  
  \[ g_{\text{extend}} = \text{the cost of extending a gap by one more space} \]
  
  \[ g_{\text{score}}(k) = g_{\text{start}} + (k-1) \times g_{\text{extend}} \]

**Question:** How to develop an efficient dynamic programming algorithm for affine gap penalties?
Exact Pattern Matching

Goal: Find all occurrences of a pattern in a text

Input: Pattern \( p = p_1 \ldots p_n \) and text \( t = t_1 \ldots t_m \)

Output: All positions \( 1 \leq i \leq (m - n + 1) \) such that the \( n \)-letter substring of \( t \) starting at \( i \) matches \( p \)

Motivation: Searching database for a known pattern
Pattern Matching: Running Time

- Naïve runtime: $O(nm)$
  - How?

- On average, it should be close to $O(m)$
  - Why?

- Can solve problem in $O(m)$ time?
  - Yes, we’ll see how (in a later lecture)
Naive algorithm is inefficient

As we saw, our alignment algorithms scale as $O(nm)$. When $n \approx 10^9$ and $m \approx 10^2$, this becomes intractable (especially when we 10 of millions of strings of length $\sim m$).

Even ignoring, e.g., memory access, say filling in each matrix cell takes $C = 10$ CPU cycles.

$$N = 10^9 \quad M = 10^2 \quad R = 10^7$$

order of genome \hspace{1cm} order of read length \hspace{1cm} order of # of reads

$$\text{# of ops} \approx N \times M \times R \times C = 10^{19}$$

$$\text{ops/sec} \approx 3 \times 10^9 \text{ (3GHz CPU)}$$

$$\text{# ops / (ops/sec)} = \text{secs} \approx 10^{19} / (3 \times 10^9) = (1/3) \times 10^{10}$$

$\sim 10^6$ Years! (for a relatively small 10M read dataset)
Multiple Pattern Matching

Goal: Given a set of patterns and a text, find all occurrences of any of patterns in text

Input: k patterns \( p^1, \ldots, p^k \), and text \( t = t_1 \ldots t_m \)

Output: Positions \( 1 < i < m \) where substring of \( t \) starting at \( i \) matches \( p_j \) for \( 1 < j < k \)

Motivation: Searching database for known multiple patterns
Multiple Pattern Matching

• **Solution:** k “pattern matching problems”: O(kmn)

• **Another Solution:**
  • Using “Keyword trees” => O(kn+nm) where n is maximum length of $p^i$
  • Preprocess all k patterns to construct a “keyword tree”
  • Now, any given text, all occurrences of all patterns can be found in time O(m)
Keyword tree approach

- **Keyword tree**: Apple
Keyword tree approach

- **Keyword tree:**
  - Apple
  - Apropos
Keyword tree approach

• **Keyword tree:**
  – Apple
  – Apropos
  – Banana
Keyword tree approach

- **Keyword tree:**
  - Apple
  - Apropos
  - Banana
  - Bandana
Keyword tree approach

- **Keyword tree:**
  - Apple
  - Apropos
  - Banana
  - Bandana
  - Orange
Keyword tree approach: Properties

- Stores a set of keywords in a rooted labeled tree
- Each edge labeled with a letter from an alphabet
- Any two edges coming out of the same vertex have distinct labels
- Every keyword stored can be spelled on a path from root to some leaf
Keyword tree approach: Complexity

• Build keyword tree in $O(kn)$ time; $kn$ is total length of all patterns

• Start “threading” at each position in text; at most $n$ steps tell us if there is a match here to any $p^i$

• $O(kn + nm)$
  • We’re down from $O(kmn)$ to this

• The next big idea, Aho-Corasick algorithm: $O(kn + m)$