CSE 421: Algorithms

Winter 2014
Lecture 17: Sequence alignment and Bellman-Ford

Reading:
Sections 6.6-6.10

sequence alignment: edit distance

• **Given:**
  – Two strings of characters \( A = a_1 a_2 ... a_n \) and \( B = b_1 b_2 ... b_m \)

• **Find:**
  – The minimum number of edit steps needed to transform \( A \) into \( B \) where an edit can be:
    – insert a single character
    – delete a single character
    – substitute one character by another

recursive solutions

• **Sub-problems:** Edit distance problems for all prefixes of \( A \) and \( B \) that don’t include all of both \( A \) and \( B \)

• Let \( D(i,j) \) be the number of edits required to transform \( a_1 a_2 ... a_i \) into \( b_1 b_2 ... b_j \)

• Clearly \( D(0,0) = 0 \)

computing \( D(n,m) \)

• Imagine how best sequence handles the last characters \( a_n \) and \( b_m \)

• Think of \( b_1 b_2 ... b_m \) as fixed and we want to edit \( a_1 a_2 ... a_n \). How will the last character become \( b_m \)?
computing $D(n,m)$

- Imagine how best sequence handles the last characters $a_n$ and $b_m$.
- If best sequence of operations
  - deletes $a_n$ then $D(n,m) =$
  - inserts $b_m$ then $D(n,m) =$
  - replaces $a_n$ by $b_m$ then
    $D(n,m) =$
  - matches $a_n$ and $b_m$ then
    $D(n,m) =$

recursive algorithm $D(n,m)$

\[
\text{if } n = 0 \text{ then return } (m) \text{ else if } m = 0 \text{ then return } (n) \text{ else if } a_n = b_m \text{ then replace-cost } \leftarrow 0 \text{ else replace-cost } \leftarrow 1 \text{ endif return } (\min(D(n-1, m) + 1, D(n, m-1) + 1, D(n-1, m-1) + \text{replace-cost}))
\]

dynamic programming

\[
\text{for } j = 0 \text{ to } m: \quad D(0,j) \leftarrow j \text{ endfor for } i = 1 \text{ to } n: \quad D(i,0) \leftarrow i \text{ endfor for } i = 1 \text{ to } n \text{ for } j = 1 \text{ to } m: \quad \text{if } a_i = b_j \text{ then replace-cost } \leftarrow 0 \text{ else replace-cost } \leftarrow 1 \text{ endif endfor } D(i,j) \leftarrow \min(D(i-1, j-1) + \text{replace-cost}, D(i-1, j) + 1, D(i, j-1) + 1)
\]
### Example Run with AGACATTG and GAGTTA

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</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>T</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>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example Run with AGACATTG and GAGTTA

<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></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></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></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example Run with AGACATTG and GAGTTA

<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></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></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></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Reading off the operations

- Follow the sequence and use each color of arrow to tell you what operation was performed.
- From the operations can derive an optimal alignment / edit sequence:

  A G A C A T T G  
  _ G A G _ T T A
computing edit distance on strings of length $m$ and $n$

- **Time:**
- **Space:**

saving space

- To compute the distance values we only need the last two rows (or columns)
  - $O(\min(m,n))$ space
- To compute the alignment/sequence of operations
  - seem to need to store all $O(mn)$ pointers/arrow colors
- Nifty divide and conquer variant that allows one to do this in $O(\min(m,n))$ space and retain $O(mn)$ time
  - 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
  - Basic algorithm is run since the whole table of pointers (2 bits each) will fit in RAM
  - Ideas are neat, though

saving space

- Alignment corresponds to a path through the table from lower right to upper left
  - Must pass through the middle column
- Recursively compute the entries for the middle column from the left
  - If we knew the cost of completing each then we could figure out where the path crossed
  - Problem
    - There are $n$ possible strings to start from.
  - Solution
    - Recursively calculate the right half costs for each entry in this column using alignments starting at the other ends of the two input strings
    - Can reuse the storage on the left when solving the right hand problem

<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>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>2</td>
<td>3</td>
<td>4</td>
<td>5</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>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</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>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
recurrence

\[ T(m,n) \leq cmn + T\left(q, \frac{n}{2}\right) + T\left(m - q, \frac{n}{2}\right) \]
\[ T(m,1) \leq cm \]
\[ T(1,n) \leq cn \]

shortest paths with negative edge weights

• Diškrta’s algorithm failed with negative-cost edges
  – What can we do in this case?
  – Negative-cost cycles could result in shortest paths with length \(-\infty\)

• Suppose no negative-cost cycles in \(G\)
  – Shortest path from \(s\) to \(t\) has at most \(n-1\) edges
    If not, there would be a repeated vertex which would create a cycle that could be removed since cycle can’t have negative cost

shortest paths with negative edge weights

We want to grow paths from \(s\) to \(t\) based on the \# of edges in the path
• Let \(\text{Cost}(s,t,i)\)=cost of minimum-length path from \(s\) to \(t\) using up to \(i\) hops.

  – \(\text{Cost}(v,t,0) = \begin{cases} 0 & \text{if } v=t \\ \infty & \text{otherwise} \end{cases} \)

  – \(\text{Cost}(v,t,i) = \min\{ \text{Cost}(v,t,i-1), \min_{(w) \in E} (c_{vw} + \text{Cost}(w,t,i-1)) \} \)
Bellman-Ford

- Observe that the recursion for $\text{Cost}(s,t,i)$ doesn’t change $t$
  - Only store an entry for each $v$ and $i$
    - Termed $\text{OPT}(v,i)$ in the text
- Also observe that to compute $\text{OPT}(*,i)$ we only need $\text{OPT}(*,i-1)$
  - Can store a current and previous copy in $O(n)$ space.

Bellman-Ford

```
Bellman-Ford

\text{ShortestPath}(G,s,t)
\text{for all } v \in V
  \text{OPT}[v] \leftarrow \infty
\text{OPT}[t] \leftarrow 0
\text{for } i = 1 \text{ to } n-1 \text{ do}
  \text{for all } v \in V \text{ do}
    \text{OPT}'[v] \leftarrow \min_{(v,w) \in E} (c_{vw} + \text{OPT}[w])
  \text{for all } v \in V \text{ do}
    \text{OPT}[v] \leftarrow \min(\text{OPT}'[v], \text{OPT}[v])
\text{return } \text{OPT}[s]
```

Bellman-Ford
Bellman-Ford

Bellman-Ford

Bellman-Ford

Bellman-Ford
negative cycles

• **Claim:** There is a negative-cost cycle that can reach \( t \) iff for some vertex \( v \in V, \ Cost(v,t,n) < Cost(v,t,n-1) \)

• **Proof:**
  
  – We already know that if there aren’t any then we only need paths of length up to \( n-1 \)
  
  – For the other direction
    
    The recurrence computes \( Cost(v,t,i) \) correctly for any number of hops \( i \)
    
    The recurrence reaches a fixed point if for every \( v \in V, \ Cost(v,t,i)=Cost(v,t,i-1) \)
    
    A negative-cost cycle means that eventually some \( Cost(v,t,i) \) gets smaller than any given bound
    
    Can’t have a negative cost cycle if for every \( v \in V, \ Cost(v,t,n)=Cost(v,t,n-1) \)
last details

- Can run algorithm and stop early if the $OPT$ and $OPT'$ arrays are ever equal
  - Even better, one can update only neighbors $v$ of vertices $w$ with $OPT'[w] \neq OPT[w]$
- Can store a successor pointer when we compute $OPT$
  - Homework assignment

- By running for $n$ steps we can find some vertex $v$ on a negative cycle and use the successor pointers to find the cycle