Longest Common Subsequence and Global Sequence Alignment

Jonathan Backer
backer@cs.ubc.ca

Department of Computer Science
University of British Columbia

July 5, 2007

Introduction

Reading:

- "Longest Common Subsequence", 15.4 CLRS
- "Text Similarity Testing", 9.4 GT

Look at two dynamic programming algorithms that measure the similarity of two sequences.

Longest common subsequence

Definition

Given a sequence of values $(a_1, a_2, \ldots, a_m)$, a subsequence is a sequence $(a_{i_1}, a_{i_2}, \ldots, a_{i_t})$ where $1 \leq i_1 < i_2 < \ldots i_t \leq m$.

Alternatively,

- $()$ is a subsequence of every sequence.
- $(b_1, \ldots, b_t)$ is a subsequence of $(a_1, \ldots, a_m)$ if there is some i such that $b_t = a_i$ and $(b_1, \ldots, b_{t-1})$ is a subsequence of $(a_1, \ldots, a_{i-1})$.

Problem

Given sequences $(a_1, a_2, \ldots, a_m)$ and $(b_1, b_2, \ldots, b_n)$, find the longest subsequence that is common to both sequences.

Example

The LCS of $(1, 1, 2, 3, 4, 5)$ and $(5, 2, 3, 4, 1, 1)$ is $(2, 3, 4)$.

Optimal substructure

Lemma

Let $(x_1, \ldots, x_t)$ be a LCS of $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$. Then there is some i, j such that $x_t = a_i = b_j$ and $(x_1, \ldots, x_{t-1})$ is a LCS of $(a_1, \ldots, a_{i-1})$ and $(b_1, \ldots, b_{j-1})$.

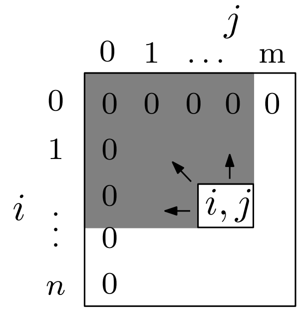
Proof

By definition, there must be some i, j such that $x_t = a_i = b_j$ and $(x_1, \ldots, x_{t-1})$ is a subsequence of $(a_1, \ldots, a_{i-1})$ and $(b_1, \ldots, b_{j-1})$. If some $(w_1, \ldots, w_s)$ is a longer subsequence of $(a_1, \ldots, a_{i-1})$ and $(b_1, \ldots, b_{j-1})$ than $(x_1, \ldots, x_{t-1})$, then $(w_1, \ldots, w_s, x_t)$ is a longer common subsequence of $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$ than $(x_1, \ldots, x_t)$, a contradiction. □

Data structure

Let $L[i,j]$ be the length of the LCS of $(a_1,\ldots,a_i)$ and $(b_1,\ldots,b_j)$. Then

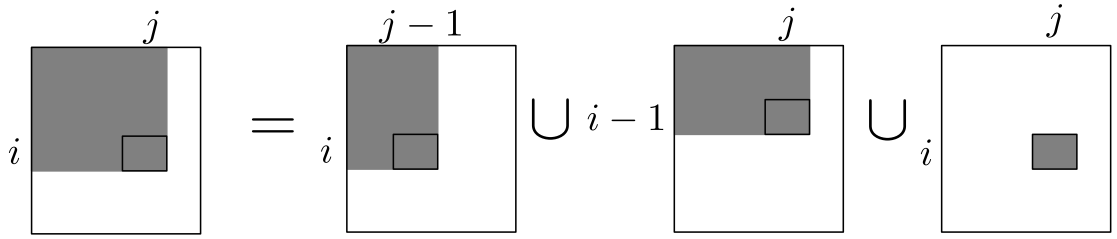
$$L[i,j] = \max_{r \le i, s \le j} \begin{cases} 0 & \text{if } r = 0 \text{ or } s = 0 \\ L[r-1,s-1]+1 & \text{if } a_r = b_s \\ 0 & \text{otherwise} \end{cases}$$



$L[i,j]$ depends on the values of all of the cells in the shaded region. So fill each row from left-to-right, starting with the top row and proceeding to the bottom.

Each cell in L takes $O(nm)$ time to fill. There are $O(nm)$ cells. So the total runtime is $O(n^2m^2)$.

Redundancy

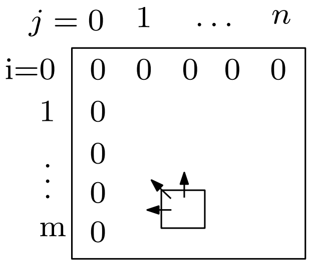


The region being maximized over can be represented by three smaller regions. The maximization of two of the regions are subproblems stored in the table. So

$$L[i,j] = \begin{cases} 0 & \text{if } r = 0 \text{ or } s = 0 \\ \max\left\{ \begin{array}{l} L[i-1,j-1]+1, \\ L[i-1,j], L[i,j-1] \end{array} \right\} & \text{if } a_i = b_j \\ \max\{L[i-1,j], L[i,j-1]\} & \text{otherwise} \end{cases}$$

Initialization

- zero the left column and bottom row
- $lcs[i, j]$ depends on values left and above, so fill each row left-to-right starting from the top going towards the bottom



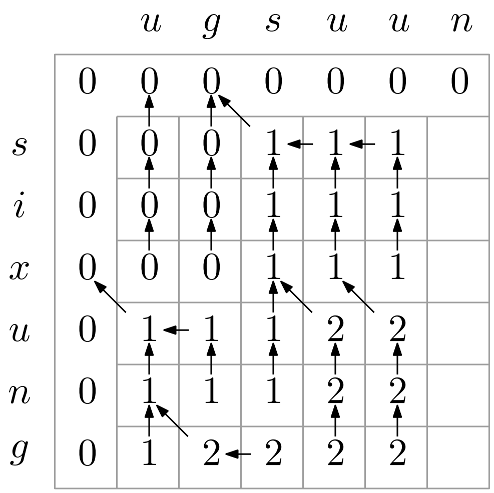
```
Algorithm LCS(A,B)
    int  $lcs[A.\texttt{length}+1, B.\texttt{length}+1]$
    char $backtrack[A.\texttt{length}+1, B.\texttt{length}+1]$
    for $i \leftarrow 0$ to $A$.length do
        $lcs[i, 0] \leftarrow 0$
    for $j \leftarrow 0$ to $B$.length do
        $lcs[0, j] \leftarrow 0$
```

Main loop

```
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $m$ do
            $lcs[i, j] \leftarrow lcs[i-1, j]$
            $backtrack[i, j] \leftarrow$ '$\uparrow$'
            if $lcs[i, j-1] > lcs[i, j]$ then
                $lcs[i, j] \leftarrow lcs[i, j-1]$
                $backtrack[i, j] \leftarrow$ '$\leftarrow$'
            if $a_i = b_j$ and
                $lcs[i-1, j-1] + 1 > lcs[i, j]$
            then
                $lcs[i, j] \leftarrow lcs[i-1, j-1] + 1$
                $backtrack[i, j] \leftarrow$ '$\nwarrow$'
```

Example

Find the longest common subsequence of (s, i, x, u, n, g) and (u, g, s, u, u, n).



		u	g	s	u	u	n
	0	0	0	0	0	0	0
s	0	0	0	1	1	1	
i	0	0	0	1	1	1	
x	0	0	0	1	1	1	
u	0	1	1	1	2	2	
n	0	1	1	1	2	2	
g	0	1	2	2	2	2	

Global sequence alignment

Given two sequences $X = (a_1, \ldots, a_m)$ and $Y = (b_1, \ldots, b_n)$, we want to figure out how similar they are.

A bit similar to longest common sequence, but:

- we may want to match 2 different letters

  e.g. $\texttt{AA}\underline{\texttt{C}}\texttt{CAT}\underline{\texttt{G}}\texttt{TC}$
  $\texttt{AA}\underline{\texttt{G}}\texttt{CAT}\underline{\texttt{A}}\texttt{TC}$

- we may want to allow gaps (i.e. insertions or deletions)

  e.g. `AGCCGCT_CC`
  `AGC__CTGCC`

Application: DNA sequence alignment

Matching two pieces of DNA

- evolution may have inserted/removed pieces
- some bases (i.e. the letters) may have mutated

Formalization

Definition

A pairwise sequence alignment of X, Y is a pair of sequences X', Y', possibly containing gaps ("-") such that

- X' minus the gaps is X,
- Y' minus the gaps is Y,
- $|X'| = |Y'|$, and
- $X'_i = Y'_i =$"-" never happens

The score of the alignment X', Y' is

$$\sum_{i=1}^{|X'|} s(X'_i, Y'_i),$$

where s is a score function defined by biologists to tell us how good/bad a mismatch is.

- e.g. $s(A, A) = 8$, $s(*, -) = -\delta$ (gap penalty), etc.

Optimal substructure

Call the prefix of a sequence everything but the last element. Let X', Y' be an optimal sequence alignment. Then X' ends in either a_m or $-$ and Y' ends in either b_n or $-$.

So four cases to consider:

1. X' ends in a_m and Y' ends in b_n: then the prefixes of X', Y' are an optimal alignment of the prefixes of X, Y.
2. X' ends in a_m and Y' ends in $-$: then the prefixes of X', Y' are an optimal alignment of the prefix of X and the whole of Y.
3. X' ends in $-$ and Y' ends in b_n: then the prefixes of X', Y' are an optimal alignment of the whole of X and the prefix of Y.
4. X' and Y' cannot both end in $-$

Data structure

Let $gsa[i,j]$ be the score of the optimal sequence alignment of $(a_1, \ldots, a_i)$ and $(b_1, \ldots, b_j)$. Then

$$gsa[i,j] = \begin{cases} 0 & \text{if } i = j = 0 \\ -i\delta & \text{if } i > 0, j = 0 \\ -j\delta & \text{if } i = 0, j > 0 \\ \max \left\{ \begin{array}{l} gsa[i-1,j-1]+ \\ \quad s(a_i, b_j), \\ gsa[i-1,j] - \delta, \\ gsa[i,j-1] - \delta \end{array} \right\} & \text{if } i > 0, j > 0 \end{cases}$$

by the optimal substructure, where δ is the gap penalty. We can fill in the table the same way as for the longest common subsequence.

Initialization

```
Algorithm Smith-Wasserman(X,Y)
   int    $gsa[X.\mathtt{length}+1, Y.\mathtt{length}+1]$
   string $backtrack[X.\mathtt{length}+1, Y.\mathtt{length}+1]$
   for $i \leftarrow 1$ to $m$ do
      $gsa[i,0] \leftarrow -i\delta$
      $backtrack[i,0] \leftarrow$ 'gap in X'
   for $j \leftarrow 1$ to $n$ do
      $gsa[0,j] \leftarrow -j\delta$
      $backtrack[i,0] \leftarrow$ 'gap in Y'
```

Main loop

```
for $i \leftarrow 1$ to $m$ do
    for $j \leftarrow 1$ to $n$ do
        $mscore \leftarrow gsa[i-1, j-1] + s(a_i, b_j)$
        $xscore \leftarrow gsa[i-1, j] - \delta$
        $yscore \leftarrow gsa[i, j-1] - \delta$
        if $mscore \geq xscore$ and $mscore \geq yscore$ then
            $gsa[i, j] \leftarrow mscore$
            $backtrack[i, j] \leftarrow$ ‘match’
        else if $xscore \geq yscore$ then
            $gsa[i, j] \leftarrow xscore$
            $backtrack[i, j] \leftarrow$‘gap in X’
        else
            $gsa[i, j] \leftarrow yscore$
            $backtrack[i, j] \leftarrow$‘gap in Y’
```

Path recovery

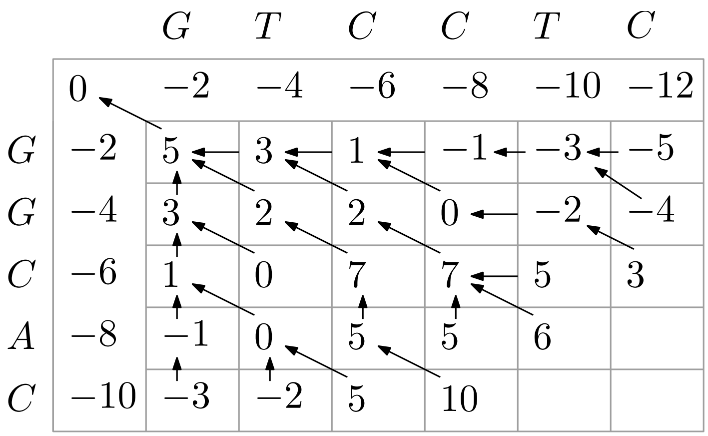
```
$X' \leftarrow$‘ ’
$Y' \leftarrow$‘ ’
$i \leftarrow m$
$j \leftarrow n$
while $i > 0$ or $j > 0$ do
    if $backtrack[i, j] =$‘match’ then
        $X' \leftarrow a_i + X'$
        $Y' \leftarrow b_j + Y'$
    else if $backtrack[i, j] =$’gap in X’ then
        $X' \leftarrow$‘-’$+X'$
        $Y' \leftarrow b_j + Y'$
    else
        $X' \leftarrow a_i + X'$
        $Y' \leftarrow$‘-’$+Y'$
return $X', Y', gsa[m, n]$
```

Example

Let $X =$ `GGCAC` and $Y =$ `GTCCTC`. Let

$$score(x, y) = \begin{cases} 5 & \text{if } x = y \\ -1 & \text{if } x \neq y, \text{ but} \quad \text{both } x, y \in \{A, T\} \text{ or} \\ & \qquad\qquad\qquad\quad \text{both } x, y \in \{G, C\} \\ -3 & \text{otherwise} \end{cases}$$

with a gap penalty of -2



		G	T	C	C	T	C
	0	−2	−4	−6	−8	−10	−12
G	−2	5	3	1	−1	−3	−5
G	−4	3	2	2	0	−2	−4
C	−6	1	0	7	7	5	3
A	−8	−1	0	5	5	6	
C	−10	−3	−2	5	10		