Dynamic Programming: Longest Common Subsequence

Chapter 15.4
Subsequences

A subsequence is a subcollection of the elements of a sequence, taken in the order they appear in the sequence.

- For instance, if the sequence is ABCDEA, then ACDA is a subsequence (\textit{ABCDEA}), but AEC is not.
- One can also think of a subsequence as the original sequence with some elements crossed out: ABCDEA

It is easiest to think of the sequences as strings, but in reality, any sequences can be used, such as sequences of integers or real numbers or customer records.

- 4 15 24 14 19 33 8 17 has a subsequence 24 19 8 17.
Longest Common Subsequence

The Longest Common Subsequence (LCS) problem is to find a longest sequence that is a subsequence of two given sequences.

For example, suppose we have two DNA strands encoded as

\[ X = \text{ACCGGTTCGAGTGCGCGGAAGCCGGCCGAA} \]

and

\[ Y = \text{GTCGTTTCCGAATGCGCCTTGCTCTGTGTA} \]

The length of the LCS is then a measure of the similarity of the strands.
Longest Common Subsequence

For these strings,

\[ X = \text{ACC} \text{GGGTCGAGTGCGC} \text{GGAAGC} \text{C} \text{GGG} \text{CC} \text{GAA} \]
\[ Y = \text{GTCGTT} \text{C} \text{GGAATGC} \text{CCGTTGCTCTGTAA} \]

an LCS is \( \text{GTCGTCGGAAGCCGGCCGAA} \).

For \( X = \text{SPRINGTIME} \) and \( Y = \text{PIONEER} \), an LCS is \( \text{PINE} \) (\( \text{SPRINGTIME} \) and \( \text{PIONEER} \)).
Structure of an LCS

If X = <x₁x₂...xₘ> is a sequence, let Xᵢ denote the iᵗʰ prefix of X. In notation, Xᵢ = <x₁x₂...xᵢ>.

**Theorem:**

Let X = <x₁x₂...xₘ> and Y = <y₁y₂...yₙ> be sequences, and let Z = <z₁z₂...zₖ> be any LCS of X and Y.

1. If xₘ = yₙ, then zₖ = xₘ = yₙ and Zₖ₋₁ is an LCS of Xₘ₋₁ and Yₙ₋₁.
2. If xₘ ≠ yₙ, then zₖ ≠ xₘ implies that Z is an LCS of Xₘ₋₁ and Y.
3. If xₘ ≠ yₙ, then zₖ ≠ yₙ implies that Z is an LCS of X and Yₙ₋₁.
Structure of an LCS

Proof:

(1) If \( z_k \neq x_m \) we could append \( x_m = y_n \) to \( Z \). →←
If \( Z_{k-1} \) were not an LCS of \( X_{m-1} \) and \( Y_{n-1} \), we could replace it by an LCS \( W \) of \( X_{m-1} \) and \( Y_{n-1} \); then \( Wz_k \) is longer than \( Z \). →←

(2) If \( z_k \neq x_m \) then \( Z \) is a common subsequence of \( X_{m-1} \) and \( Y \). If \( Z \) were not an LCS of \( X_{m-1} \) and \( Y \), we could replace \( Z \) by an LCS \( W \) of \( X_{m-1} \) and \( Y \); then \( W \) (which is longer than \( Z \)) is a (longest) common subsequence of \( X \) and \( Y \). →←

(3) Symmetric to (2).
DP properties of LCS

The theorem shows that LCS has the optimal-substructure property. The LCS of X and Y involves the LCS of prefixes of X and Y.

LCS also has the overlapping subproblems property: In determining an LCS for $X_i$ and $Y_{j-1}$, we may use (case 2) the LCS for $X_{i-1}$ and $Y_{j-1}$. But also, in determining an LCS for $X_{i-1}$ and $Y_j$, we may also use (case 3) the LCS for $X_{i-1}$ and $Y_{j-1}$.

Thus, differing subproblems make use of the same subproblem.
Recursive formulation for the length of an LCS

Consider an LCS of X and Y.

Let $c(i, j)$ be the length of an LCS of sequences $X_i$ and $Y_j$. Then:

$$c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\
\max(c(i, j - 1), c(i - 1, j)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j
\end{cases}$$

The first case is the basis. The second case is case (1) of the theorem. The third case is the combination of cases (2) and (3) of the theorem.
Formulation as code

LCS(X, Y) {
    return c(m, n);
}

c(i, j) {
    if (i=0 or j=0)
        return 0;
    if (X[i] = Y[j])
        return 1 + c(i-1, j-1);
    return max(c(i, j-1), c(i-1, j));
}

Analysis:
let T(s) be time for c(i, j) where s = i+j

T(s) = O(1) if s=0
T(s) = O(1) + max(T(s-2), 2T(s-1)) if not
T(s) ∈ Ω(2^s)
Memoized

LCS(X, Y) {
allocate matrix memo[0..m, 0..n] = ∞;
return c(m, n);
}
c(i, j) {
    if memo[i, j] ≠ ∞
        return memo[i, j];
    if (i=0 or j=0)
        memo[i, j] = 0;
    else if (X[i] = Y[j])
        memo[i, j] = 1 + c(i-1, j-1);
    else
        memo[i, j] = max(c(i, j-1), c(i-1, j));
    return memo[i, j];
}
Memoized analysis

Consider all calls to c(i, j). At most (m+1)(n+1) of them make it past the memo-checking if statement. Each of these has the potential to call c() recursively twice. LCS() calls c() once. Thus there are at most 2(m+1)(n+1) + 1 calls of c(i, j).

The nonrecursive work in c(i, j) is O(1), so the total work in c(i, j) is at most (2(m+1)(n+1) + 1)·O(1) = O(mn).

The nonsubroutine work in LCS() is O(1) + matrix allocation work, which is O(mn) because the entries are initialized to something other than 0.

Thus, the total work is the work in LCS() + work in c() = O(mn) + O(mn) = O(mn).

Exercise: One can avoid the O(mn) matrix allocation work by having the memo[i, j] store 1 + length(LCS(X_i,Y_j)) rather than length(LCS(X_i,Y_j)); this allows initialization of memo[] by 0. Write out the pseudocode for this. (Work out before viewing the next slide.)
Memoized Variant

\[ \text{LCS}(X, Y) \{ \]
\>
\begin{align*}
\text{allocate matrix} \ & \text{memo}[0..m, 0..n] = 0; \\
\text{return} \ & c(m, n); \\
\}
\end{align*}
\[ \]
\[ c(i, j) \{ \\
\text{if memo}[i, j] \neq 0} \]
\>
\begin{align*}
\text{return memo}[i, j] - 1; \\
\end{align*}
\[ \\
\text{if (i=0 or j=0)} \]
\>
\begin{align*}
\text{memo}[i, j] = 0 + 1; \\
\end{align*}
\[ \\
\text{else if (X[i] = Y[j])} \\
\text{memo}[i, j] = 1 + c(i-1, j-1) + 1; \\
\text{else} \]
\>
\begin{align*}
\text{memo}[i, j] = \max(c(i, j-1), c(i-1, j)) + 1; \\
\text{return memo}[i, j] - 1; \\
\end{align*}
\[ \\
\]
\[ // \text{note: written 0+1 rather than 1 for clarity.} \]
\[ // \text{In code, compiler would clean it up.} \]
Dynamic Programming

For dynamic programming, we must find the order in which to compute the memo table entries.

We start with the entries that have $i=0$ or $j=0$.
Dynamic Programming

To compute an entry \( \text{memo}[i, j] \), we may need \( \text{memo}[i-1, j-1], \text{memo}[i-1, j], \) and \( \text{memo}[i, j-1] \).

\[ \text{memo} \]

\[ j \rightarrow \]

\[ i \rightarrow \]

- to compute this
- we need these
Dynamic Programming

A row-major order (or a column-major one) will ensure that we have the entries we need when computing \( \text{memo}[i, j] \).

\[ \text{memo} \quad j \rightarrow \]

- To compute this
- We need these
LCS(X, Y) {
    allocate matrix memo[0..m,0..n] = 0;
    for i = 1 to m
        memo[i, 0] = 0;
    for j = 0 to n
        memo[0, j] = 0;
    for i = 1 to m
        for j=1 to n
            if X[i] = Y[j]
                memo[i, j] = memo[i-1, j-1] + 1;
            else
                memo[i, j] = max(memo[i-1, j], memo[i, j-1]);
    return memo[m, n];
}
Analysis of DP

LCS(X, Y) {
    allocate matrix memo[0..m,0..n] = 0;
    for i = 1 to m
        memo[i, 0] = 0;
    for j = 0 to n
        memo[0, j] = 0;
    for i = 1 to m
        for j=1 to n
            if X[i] = Y[i]
                memo[i, j] = memo[i-1, j-1] + 1;
            else
                memo[i, j] = max(memo[i-1, j], memo[i, j-1]);
    return memo[m, n];
}

O(1)
O(m) iterations
O(1) per iteration
O(mn) iterations
O(1) work per iteration
O(1)
O(m) + O(n) + O(mn) + O(1)
= O(mn)
Textbook notes

LCS on the previous slide is the equivalent of LCS-LENGTH in the text. LCS-LENGTH uses the array c[i, j] instead of memo[i, j] and uses b[i, j] to store the traceback information. The text’s PRINT-LCS is the traceback function.

In the section Improving the code, they note that the traceback information isn’t really needed for optimal-time O(m+n) traceback in this problem.

They also show that one can reduce the memory space to O(min(m, n)) if traceback is not required, by keeping only two rows (or columns, in column-major order) of the table.
Demonstration

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>a</th>
<th>n</th>
<th>k</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tbody>
</table>

a 0
m 0
p 0
u 0
t 0
a 0
t 0
i 0
n 0
Demonstration

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>p</th>
<th>a</th>
<th>n</th>
<th>k</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</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>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

same letter  different letters

© 2020 Shermer
## Demonstration

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>p</th>
<th>a</th>
<th>n</th>
<th>k</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demonstration

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>a</th>
<th>n</th>
<th>k</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<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>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Demonstration

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>p</th>
<th>a</th>
<th>n</th>
<th>k</th>
<th>i</th>
<th>n</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</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>m</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

© 2020 Shermer