COMP 355
Advanced Algorithms
Dynamic Programming:
Knapsack &
Longest Common Subsequence
Section 6.4 (KT), Section 15.4 (CLRS)
Algorithmic Paradigms

Greed. Build up a solution incrementally, myopically optimizing some local criterion.

Divide-and-conquer. Break up a problem into two sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

Dynamic programming. Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.
Dynamic Programming: Generating Subproblems

How to generate the solutions to these subproblems?

- **Top-Down**: A solution to a DP problem is expressed recursively. This approach applies recursion directly to solve the problem. However, due to the overlapping nature of the subproblems, the same recursive call is often made many times. An approach, called memoization, records the results of recursive calls, so that subsequent calls to a previously solved subproblem are handled by table look-up.

- **Bottom-up**: Although the problem is formulated recursively, the solution is built iteratively by combining the solutions to small subproblems to obtain the solution to larger subproblems. The results are stored in a table.
Knapsack Problem

There are two versions of the problem:
(1) “0-1 knapsack problem” and
(2) “Fractional knapsack problem”

(1) Items are indivisible; you either take an item or not. Solved with *dynamic programming*
(2) Items are divisible: you can take any fraction of an item. Solved with a *greedy algorithm*. 
Knapsack problem.

- Given \( n \) objects and a "knapsack."
- Item \( i \) weighs \( w_i > 0 \) kilograms and has value \( v_i > 0 \).
- Knapsack has capacity of \( W \) kilograms.
- Goal: fill knapsack so as to maximize total value.

Ex: \( \{3, 4\} \) has value 40.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

\( W = 11 \)

Greedy: repeatedly add item with maximum ratio \( v_i / w_i \).
Ex: \( \{5, 2, 1\} \) achieves only value = 35 \( \Rightarrow \) greedy not optimal.
Dynamic Programming: False Start

Def. $\text{OPT}(i) = \text{max profit subset of items } 1, ..., i.$

- Case 1: $\text{OPT}$ does not select item $i$.
  - $\text{OPT}$ selects best of $\{1, 2, ..., i-1\}$

- Case 2: $\text{OPT}$ selects item $i$.
  - Accepting item $i$ does not immediately imply that we will have to reject other items
  - Without knowing what other items were selected before $i$, we don't even know if we have enough room for $i$

Conclusion. Need more sub-problems!
Dynamic Programming: Adding a New Variable

Def. \( \text{OPT}(i, w) = \text{max profit subset of items } 1, \ldots, i \text{ with weight limit } w \).

• Case 1: \( \text{OPT} \) does not select item \( i \).
  – \( \text{OPT} \) selects best of \( \{1, 2, \ldots, i-1\} \) using weight limit \( w \)

• Case 2: \( \text{OPT} \) selects item \( i \).
  – new weight limit = \( w - w_i \)
  – \( \text{OPT} \) selects best of \( \{1, 2, \ldots, i-1\} \) using this new weight limit

\[
\text{OPT}(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
\text{OPT}(i-1, w) & \text{if } w_i > w \\
\max \{ \text{OPT}(i-1, w), v_i + \text{OPT}(i-1, w-w_i) \} & \text{otherwise}
\end{cases}
\]
Knapsack Problem: Bottom-Up

Knapsack. Fill up an n-by-W array.

Input: n, w₁,...,wₙ, v₁,...,vₙ

for w = 0 to W
  M[0, w] = 0

for i = 1 to n
  for w = 1 to W
    if (wᵢ > w)
      M[i, w] = M[i-1, w]
    else
      M[i, w] = max {M[i-1, w], vᵢ + M[i-1, w-wᵢ]}

return M[n, W]
Knapsack Algorithm

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

OPT: \{ 4, 3 \}
value = 22 + 18 = 40

W = 11
Knapsack Problem: Running Time

Running time. $\Theta(nW)$.

- Not polynomial in input size!
- "Pseudo-polynomial."
- Decision version of Knapsack is NP-complete.

Knapsack approximation algorithm. There exists a polynomial algorithm that produces a feasible solution that has value within 0.01% of optimum.
Using DP to compare strings

• Determining the degree of similarity between two strings
  – Applications in computational biology (sequence alignment)
  – Applications in document processing and retrieval

• One common measure of similarity between two strings is the lengths of their longest common subsequence.
Longest Common Subsequence (LCS)

Given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Z = \langle z_1, z_2, \ldots, z_k \rangle$, we say that $Z$ is a subsequence of $X$ if there is a strictly increasing sequence of $k$ indices $\langle i_1, i_2, \ldots, i_k \rangle$ ($1 \leq i_1 < i_2 < \ldots < i_k \leq n$) such that $Z = \langle x_{i_1}, x_{i_2}, \ldots, x_{i_k} \rangle$. For example, let $X = \langle ABRACADABRA \rangle$ and let $Z = \langle AADAA \rangle$, then $Z$ is a subsequence of $X$.

**LCS Problem:** Given two sequences $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ determine the length of their longest common subsequence, and more generally the sequence itself.
Brute Force for LCS

Brute Force: compare each subsequence of X with the symbols in Y

If $|X| = m$, $|Y| = n$, then there are $2^m$ subsequences of x; we must compare each with Y (n comparisons)

Running time: $O(n \ 2^m)$
Notice that the LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.

**Subproblems:** “find LCS of pairs of *prefixes* of X and Y”

A prefix of a sequence is just an initial string of values, $X_i = <x_1, \ldots, x_i>$. $X_0$ is the empty sequence.

Let $\text{lcs}(i, j)$ denote the length of the longest common subsequence of $X_i$ and $Y_j$.

Example: $X_5 = <ABRAC>$ and $Y_6 = <YABBAD>$. Their longest common subsequence is $<ABA>$. Thus, $\text{lcs}(5, 6) = 3$. 
**DP Formulation for LCS**

**Base Case:** If either sequence is empty, then the longest common subsequence is empty. Therefore, $\text{lcs}(i, 0) = \text{lcs}(j, 0) = 0$.

**Last characters match:** Suppose $x_i = y_j$. For example: Let $X = \langle\text{ABCA}\rangle$ and let $Y = \langle\text{DACA}\rangle$. Since both end in ‘A’, it is easy to see that the LCS must also end in ‘A’.

**Last characters do not match:** Suppose that $x_i \neq y_j$. In this case $x_i$ and $y_j$ cannot both be in the LCS (since they would have to be the last character of the LCS). Thus either $x_i$ is not part of the LCS, or $y_j$ is not part of the LCS (and possibly both are not part of the LCS).

- **Option 1:** ($x_i$ is not in the LCS) Since we know that $x_i$ is out, we can infer that the LCS of $X_i$ and $Y_j$ is the LCS of $X_{i-1}$ and $Y_j$, which is given by $\text{lcs}(i - 1, j)$.
- **Option 2:** ($y_j$ is not in the LCS) Since $y_j$ is out, we can infer that the LCS of $X_i$ and $Y_j$ is the LCS of $X_i$ and $Y_{j-1}$, which is given by $\text{lcs}(i, j - 1)$.
LCS of two strings whose last characters are equal.
if \( x_i = y_j \) then \( \text{lcs}(i, j) = \text{lcs}(i - 1, j - 1) + 1 \)

The possible cases in the DP formulation of LCS.
if \( x_i \neq y_j \) then \( \text{lcs}(i, j) = \max(\text{lcs}(i - 1, j), \text{lcs}(i, j - 1)) \)

\[
\text{lcs}(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0, \\
\text{lcs}(i - 1, j - 1) + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\
\max(\text{lcs}(i - 1, j), \text{lcs}(i, j - 1)) & \text{if } i, j > 0 \text{ and } x_i \neq y_j.
\end{cases}
\]
Memoized Implementation

```c
memoized-lcs(i, j) {
    if (lcs[i, j] has not yet been computed) {
        if (i == 0 || j == 0) // basis case
            lcs[i, j] = 0
        else if (x[i] == y[j]) // last characters match
            lcs[i, j] = memoized-lcs(i-1, j-1) + 1
        else // last chars don’t match
            lcs[i, j] = max(memoized-lcs(i-1, j), memoized-lcs(i, j-1))
    }
    return lcs[i, j] // return stored value
}
```

- The running time of the memoized version is $O(mn)$.
- Observe that there are $m+1$ possible values for $i$, and $n + 1$ possible values for $j$.
- Each time we call `memoized-lcs(i, j)`, if it has already been computed then it returns in $O(1)$ time.
- Each call to `memoized-lcs(i, j)` generates a constant number of additional calls.
- Therefore, the time needed to compute the initial value of any entry is $O(1)$, and all subsequent calls with the same arguments is $O(1)$.
- Total running time is equal to the number of entries computed, which is $O((m + 1)(n + 1)) = O(mn)$. 

Rhodes College
Bottom-up implementation

```java
bottom-up-lcs() {
    lcs = new array [0..m, 0..n]
    for (i = 0 to m) lcs[i,0] = 0  // basis cases
    for (j = 0 to n) lcs[0,j] = 0
    for (i = 1 to m) {
        for (j = 1 to n) {
            if (x[i] == y[j])  // take x[i] (= y[j]) for LCS
                lcs[i,j] = lcs[i-1, j-1] + 1
            else
                lcs[i,j] = max(lcs[i-1, j], lcs[i, j-1])
        }
    }
    return lcs[m, n]  // final lcs length
}
```

- Running time: $O(mn)$
- Space: $O(mn)$
We’ll see how LCS algorithm works on the following example:

- X = ABCB
- Y = BDCAB

What is the Longest Common Subsequence of X and Y?

\[ \text{LCS}(X, Y) = BCB \]

\[
\begin{align*}
X &= A \ B \ C \ B \\
Y &= B \ D \ C \ A \ B
\end{align*}
\]
LCS Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Xi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

X = ABCB; \( m = |X| = 4 \)

Y = BDCAB; \( n = |Y| = 5 \)

Allocate array \( c[5,4] \)
### LCS Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Xi</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>X0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Base Cases**

- For $i = 1$ to $m$  
  \[
  \text{lc}[i, 0] = 0 
  \]
- For $j = 1$ to $n$  
  \[
  \text{lc}[0, j] = 0 
  \]
### LCS Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example Strings:**

```
ABC
BDCAB
```

**Algorithm:**

\[ \text{lcs}[i,j] = \begin{cases} 
\text{lcs}[i-1,j-1] + 1 & \text{if } X_i = Y_j \\
\max( \text{lcs}[i-1,j], \text{lcs}[i,j-1] ) & \text{otherwise}
\end{cases} \]

---

2/26/2014

COMP 355: Advanced Algorithms
Spring 2014
LCS Example

<table>
<thead>
<tr>
<th></th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Xi</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( X_i == Y_j )
\[
lcs[i,j] = lcs[i-1,j-1] + 1
\]
else \[
lcs[i,j] = \max( lcs[i-1,j], lcs[i,j-1] )
\]
LCS Example

\[
\begin{array}{cccccc}
  & 0 & 1 & 2 & 3 & 4 & 5 \\
 0 & \text{Xi} & Y_j & B & D & C & A & B \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 1 & \text{A} & \text{B} \\
 2 & \text{B} & 0 & 0 & 0 & 0 & 0 & 0 \\
 3 & \text{C} & 0 & 0 & 0 & 0 & 0 & 0 \\
 4 & \text{B} & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

if \( X_i == Y_j \)

\[
lcs[i,j] = lcs[i-1,j-1] + 1
\]
else

\[
lcs[i,j] = \max( lcs[i-1,j], lcs[i,j-1] )
\]
### LCS Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Xi</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{if } (X_i == Y_j) \\
\text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1 \\
\text{else } \text{lcs}[i,j] = \text{max}( \text{lcs}[i-1,j], \text{lcs}[i,j-1])
\]
## LCS Example

### Table

<table>
<thead>
<tr>
<th></th>
<th>(0)</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</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>
</tr>
<tr>
<td>(1)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>(2)</td>
<td>(0)</td>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>(0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Code

If \( X_i == Y_j \)

\[
\text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1
\]

Else \( \text{lcs}[i,j] = \max( \text{lcs}[i-1,j], \text{lcs}[i,j-1] ) \)

---

**COMP 355: Advanced Algorithms**  
**Spring 2014**
### LCS Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Xi</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</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>
</tr>
</tbody>
</table>

**if** ( \( X_i == Y_j \))

\[
lcs[i,j] = lcs[i-1,j-1] + 1
\]

**else** \( lcs[i,j] = \max( lcs[i-1,j], lcs[i,j-1] ) \)
LCS Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Xi</td>
<td>Yj</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if ( X_i == Y_j )
\[ \text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1 \]
else \[ \text{lcs}[i,j] = \max( \text{lcs}[i-1,j], \text{lcs}[i,j-1] ) \]
LCS Example

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
i & & & & & & \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 1 \\
\hline
2 & 0 & 1 & 1 & 1 & 1 & 2 \\
\hline
3 & 0 & 1 & 1 & & & \\
\hline
4 & 0 & & & & & \\
\hline
\end{array}
\]

\[\text{if } ( X_i == Y_j ) \]
\[\text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1 \]
\[\text{else } \text{lcs}[i,j] = \max( \text{lcs}[i-1,j], \text{lcs}[i,j-1] ) \]
LCS Example

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>Xj</td>
<td>Yj</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if \( X_i == Y_j \)
\[
lcs[i,j] = lcs[i-1,j-1] + 1
\]
else \( lcs[i,j] = \max( lcs[i-1,j], lcs[i,j-1] ) \)
### LCS Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Xi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>B</td>
<td>DC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

**Example Table**

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm**

\[
\text{if } ( X_i == Y_j ) \\
\text{lcs[i,j] = lcs[i-1,j-1] + 1} \\
\text{else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )}
\]
### LCS Example

**LCS Algorithm**:

1. Initialize the table with zeros.
2. For each character in the first string `X_i`, compare it with each character in the second string `Y_j`.
3. If `X_i` equals `Y_j`, then `lcs[i,j] = lcs[i-1,j-1] + 1`.
4. If `X_i` does not equal `Y_j`, then `lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )`.

#### Example:

Given strings `X = ABCB` and `Y = BDCAB`, the LCS matrix is:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The LCS of `ABC` and `BDCAB` is `ABC`, with a length of 3.

**Equation**:

\[
lcs[i,j] = \begin{cases} 
  lcs[i-1,j-1] + 1 & \text{if } X_i = Y_j \\
  \max( lcs[i-1,j], lcs[i,j-1] ) & \text{otherwise}
\end{cases}
\]
### LCS Example

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>X_i</th>
<th>Y_j</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- **if** \( X_i == Y_j \)
  - \( \text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1 \)
- **else** \( \text{lcs}[i,j] = \max(\text{lcs}[i-1,j], \text{lcs}[i,j-1]) \)
### LCS Example

**Input Sequences:**
- \( X = ABCB \)
- \( Y = BDCABA \)

**LCS Table:**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Recurrence Relation:**

\[
\text{if } (X_i == Y_j) \\
\text{lcs}[i,j] = \text{lcs}[i-1,j-1] + 1 \\
\text{else } \text{lcs}[i,j] = \text{max}( \text{lcs}[i-1,j], \text{lcs}[i,j-1] )
\]
Adding Hints to Reconstruct LCS

\[ \text{add}_{XY}: \text{Add } x_i (= y_j) \text{ to the LCS (‘\:\\doce’ in Fig. 44(b)) and continue with } \text{lcs}[i - 1, j - 1] \]
\[ \text{skip}_X: \text{Do not include } x_i \text{ to the LCS (‘↑’ in Fig. 44(b)) and continue with } \text{lcs}[i - 1, j] \]
\[ \text{skip}_Y: \text{Do not include } y_j \text{ to the LCS (‘←’ in Fig. 44(b)) and continue with } \text{lcs}[i, j - 1] \]

---

Bottom-up Longest Common Subsequence with Hints

```java
bottom-up-lcs-with-hints() {
    lcs = new array [0..m, 0..n]    // stores lcs lengths
    h = new array [0..m, 0..n]      // stores hints
    for (i = 0 to m) { lcs[0,0] = 0; h[0,0] = skipX }
    for (j = 0 to n) { lcs[0,0] = 0; h[0,0] = skipY }
    for (i = 1 to m) {
        for (j = 1 to n) {
            if (x[i] == y[j])
                { lcs[i,j] = lcs[i-1, j-1] + 1; h[i,j] = addXY }
            else if (lcs[i-1, j] >= lcs[i, j-1])
                { lcs[i,j] = lcs[i-1, j]; h[i,j] = skipX }
            else
                { lcs[i,j] = lcs[i, j-1]; h[i,j] = skipY }
        }
    }
    return lcs[m, n]                // final lcs length
}
```
Extracting the LCS

```c
get-lcs-sequence() {
    LCS = new empty character sequence
    i = m; j = n  // start at lower right
    while(i != 0 or j != 0)  // loop until upper left
        switch h[i,j]
            case addXY:  // add x[i] (= y[j])
                prepend x[i] (or equivalently y[j]) to front of LCS
                i--; j--; break
            case skipX: i--; break  // skip x[i]
            case skipY: j--; break  // skip y[j]
    return LCS
}
```
LCS Example

Contents of the lcs array for the input sequences $X = \langle BACDB \rangle$ and $Y = \langle BCDB \rangle$. The numeric table entries are the values of $lcs[i, j]$ and the arrow entries are used in the extraction of the sequence.
The Chips game: Dynamic programming activity

This is a game between two players. It starts with two piles of chips, ten chips per pile. On each turn, a player may take one chip from one pile, or two chips, one from each pile. If one pile is empty, you can only take one chip from the other pile. The player who takes the last chip or chips wins the game.

1. Set up and play the game with another person. Try to win! Play a few more times. Keep track of whether the first player wins or the first player loses (and what moves the players made).

2. Would you rather be the first player or not? Does it matter? Why or why not?

While you are playing, think about how you could use a DP algorithm to determine the answers to the above questions.
Next Time

• More Dynamic Programming Algorithms!