DEEMED TO BE UNIVERSITY

Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES 23CS03HF

Topic:

Longest Common Subsequence

AIM OF THE SESSION

To familiarize students with the concept of Longest Common Subsequence.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate :- Longest Common Subsequence.
2. Describe :- Use of Dynamic programming to find longest common subsequence.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- Longest Common Subsequence.
2. Describe :- Learn to formulate the recurrence relation for the LCS problem..
3. Summarize:- LCS problem identifies the longest sequence common to two strings in the same relative order using dynamic programming.

Longest Common Subsequence

Application: comparison of two DNA strings

Ex: X= {A B C B D A B }, Y= {B D C A B A}

Longest Common Subsequence:

X = A **B**   **C**   **B** D **A** B

Y =   **B** D **C** A **B**   **A**

Brute force algorithm would 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)
- So the running time of the brute-force algorithm is $O(n\,2^m)$
- 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”

- First we'll find the length of LCS. Later we'll modify the algorithm to find LCS itself.
- Define X_i, Y_j to be the prefixes of X and Y of length *i* and *j* respectively
- Define *c[i,j]* to be the length of LCS of X_i and Y_j
- Then the length of LCS of X and Y will be *c[m,n]*

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i]=y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- We start with *i = j = 0* (empty substrings of x and y)
- Since X_0 and Y_0 are empty strings, their LCS is always empty (i.e. *c[0,0] = 0*)
- LCS of empty string and any other string is empty, so for every i and j: *c[0, j] = c[i,0] = 0*

$$c[i,j]=\begin{cases} c[i-1,j-1]+1 & \text{if } x[i]=y[j], \\ \max(c[i,j-1],c[i-1,j]) & \text{otherwise} \end{cases}$$

- When we calculate *c[i,j],* we consider two cases:
- **First case:** *x[i]=y[j]*: one more symbol in strings X and Y matches, so the length of LCS X_i and Y_j equals to the length of LCS of smaller strings X_{i-1} and Y_{i-1} , plus 1

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- **Second case:** *x[i] != y[j]*
- As symbols don't match, our solution is not improved, and the length of LCS(X_i , Y_j) is the same as before (i.e. maximum of LCS(X_i, Y_{j-1}) and LCS(X_{i-1},Y_j)

Why not just take the length of LCS(X_{i-1}, Y_{j-1}) ?

LCS Length Algorithm

```
LCS-Length(X, Y)
1. m = length(X)  // get the # of symbols in X
2. n  = length(Y) // get the # of symbols in Y
3. for i = 1 to m c[i,0] = 0  // special case: $Y_0$
4. for j = 1 to n  c[0,j] = 0  // special case: $X_0$
5. for i = 1 to m              // for all $X_i$
6.   for j = 1 to n                // for all $Y_j$
7.         if ( $X_i$ == $Y_j$ )
8.               c[i,j] = c[i-1,j-1] + 1
9.         else c[i,j] = max( c[i-1,j], c[i,j-1] )
10. return c[m,n]   // return LCS length for X and Y
```

LCS Example

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?

LCS(X, Y) = BCB

X = A **B** **C** **B**

Y = **B** D **C** A **B**

LCS Example (0)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5
Allocate array c[5,4]

LCS Example (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for i = 1 to m   c[i,0] = 0
for j = 1 to n   c[0,j] = 0

LCS Example (2)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0				
2	B	0					
3	C	0					
4	B	0					

```
if ( $X_i == Y_j$ )
        $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (3)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (4)

ABCB

BDCAB

		j	0	1	2	3	4	5
i			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	
2	B		0					
3	C		0					
4	B		0					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (5)

ABCB

BDCAB

j		0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1** $\rightarrow$	**1**
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

```
if ( $X_i$ == $Y_j$ )
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max( c[i-1,j], c[i,j-1] )
```

LCS Example (6)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1				
3	C	0					
4	B	0					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (7)

ABCB

BDCAB

		0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	→ 1	→ 1	→ ↓ 1	
3	C	0					
4	B	0					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (8)

ABCB

BDCAB

i	j	0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**					
4	**B**	**0**					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (10)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	↓ 1 →	↓ 1			
4	B	0					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (11)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**		
4	**B**	**0**					

if ($X_i == Y_j$)
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (12)

ABCB

BDCAB

i	j	0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	→ **2**	↓ → **2**
4	**B**	**0**					

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (13)

ABCB

BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1				

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Example (14)

ABCB

BDCAB

i	j	0	1	2	3	4	5
		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	$\rightarrow$ $\downarrow$ **1**	$\downarrow$ **2**	$\rightarrow$ $\downarrow$ **2**	

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max( c[i-1,j], c[i,j-1] )$
```

LCS Example (15)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**

```
if ( $X_i == Y_j$ )
    $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max( c[i-1,j], c[i,j-1] )$
```

LCS Algorithm Running Time

- LCS algorithm calculates the values of each entry of the array c[m,n]
- So what is the running time?

O(m*n)

since each c[i,j] is calculated in constant time, and there are m*n elements in the array

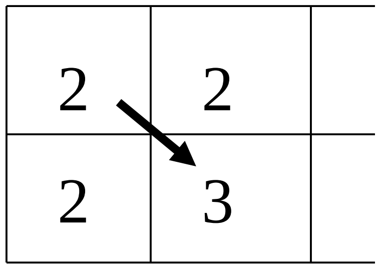
How to find actual LCS

- So far, we have just found the *length* of LCS, but not LCS itself.
- We want to modify this algorithm to make it output Longest Common Subsequence of X and Y

Each *c[i,j]* depends on *c[i-1,j]* and *c[i,j-1]*

or *c[i-1, j-1]*

For each c[i,j] we can say how it was acquired:



For example, here

c[i,j] = c[i-1,j-1] +1 = 2+1=3

How to find actual LCS - continued

- Remember that

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i]=y[j], \\ \max(c[i,j-1], c[i-1,j]) & \text{otherwise} \end{cases}$$

- So we can start from *c[m,n]* and go backwards
- Look first to see if 2nd case above was true
- If not, then *c[i,j] = c[i-1, j-1]+1*, so remember *x[i]* (because *x[i]* is a part of LCS)
- When i=0 or j=0 (i.e. we reached the beginning), output remembered letters in reverse order

Algorithm to find actual LCS

- Here's a recursive algorithm to do this:

```
LCS_print(x, m, n, c) {
    if (c[m][n] == c[m-1][n]) // go up?
        LCS_print(x, m-1, n, c);
    else if (c[m][n] == c[m][n-1] // go left?
        LCS_print(x, m, n-1, c);
    else { // it was a match!
        LCS_print(x, m-1, n-1, c);
        print(x[m]); // print after recursive call
     }
}
```

Finding LCS

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1** ↖	**1** ←	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2** ↖	**2** ←	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3** ↖

Finding LCS (2)

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	↖ **1** ←	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	↖ **2** ←	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	↖ **3**

LCS (reversed order): **B C B**

LCS (straight order): **B C B**

(this string turned out to be a palindrome)

SUMMARY

The **Longest Common Subsequence (LCS)** problem involves finding the longest sequence of characters that appear in the same relative order in two given strings, without needing to be contiguous. LCS is widely used in **bioinformatics**, **text comparison**, and **data analysis**.

SELF-ASSESSMENT QUESTIONS

1. What is the length of the LCS for X="AGGTAB"X = "AGGTAB"X="AGGTAB" and Y="GXTXAYB"Y = "GXTXAYB"Y="GXTXAYB"?

A. 2

B. 4

C. 5

D. 6

2. In the context of LCS, a subsequence is defined as:

A. A contiguous subset of characters

B. A sequence derived by deleting characters without rearranging others

C. A rearrangement of characters to form another string

D. A subsequence of maximum length

TERMINAL QUESTIONS

1. Given two strings X="ABCBDAB"X = "ABCBDAB"X="ABCBDAB" and Y="BDCAB"Y = "BDCAB"Y="BDCAB", describe the steps involved in constructing the LCS table and backtracking to find the LCS.
2. Describe the dynamic programming approach to solve the LCS problem. How do you construct the DP table, and what is the recurrence relation used?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.
2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.
3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.
4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=
2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

DEEMED TO BE UNIVERSITY

THANK YOU