Dynamic Programming

Longest Common Subsequence

Dr. G P Gupta

Longest Common Subsequence

What is Subsequences?

- Suppose you have a sequence $X = <x_1,x_2,\ldots,x_m>$ of elements over a finite set S.

- A sequence $Z = <z_1,z_2,\ldots,z_k>$ over S is called a subsequence of X **if and only** if it can be obtained from X by deleting elements.
- Put differently, there exist indices $i_1<i_2<\ldots<i_k$ such that

$$z_a = x_{i_a}$$

for all a in the range 1<= a <= k.

What is Subsequences? Cont..

A subsequence of a string S, is a set of characters that appear in left-to-right order, but not necessarily consecutively.

Example

ACTTGCG

- *ACT* , *ATTC* , *T* , *ACTTGC* are all subsequences.
- *TTA* is not a subequence

What is Common Subsequences ?

- Suppose that X and Y are two sequences over a set S.

- We say that ***Z is a common subsequence*** of X and Y if and only if
- Z is a subsequence of X
- Z is a subsequence of Y

What is Longest common subsequence ?

- **Subsequence:**
  - A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements.
- **Longest common subsequence:**
  - Longest common subsequence (*LCS*) of 2 sequences is a subsequence, with maximal length, which is common to both the sequences.

What is Longest common subsequence ?

A common subequence of two strings is a subsequence that appears in both strings. A longest common subequence is a common subsequence of maximal length.

Example

$$S_1 = AAACCGTGAGTTATTCGTTCTAGAA$$
$$S_2 = CACCCCTAAGGTACCTTTGGTTC$$

The Longest Common Subsequence Problem

Given two sequences X and Y over a set S, the longest common subsequence problem asks to find a common subsequence of X and Y that is of maximal length.

Longest Common Subsequence

- Biologists need to ***measure how similar strands of DNA are*** to determine how closely related an organism is to another.
- ***considering DNA as strings of letters*** **A,C,G,T** and then comparing similarities in the strings.
- Formally , researchers ***look at common subsequences in the strings.***

- **Example :** X = AGTCAACGTT, Y=GTTCGACTGTG
- Both **S** = AGTG and **S'**=GTCACGT are subsequences
- **How to do find these efficiently?**

What is Longest common subsequence ?

$$S_1 = AAACCGTGAGTTATTCGTTCTAGAA$$
$$S_2 = CACCCCTAAGGTACCTTTGGTTC$$

LCS is

$$ACCTAGTACTTTG$$

Has applications in many areas including biology.

Brute Force solution

- 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"

Dynamic Programming

Let us try to develop a dynamic programming solution to the LCS problem.

i^{th} prefix of X

- Let $X = < x_1, x_2, \ldots, x_m >$ be a sequence.
- **i^{th} prefix of X :**

  We denote by X_i the sequence $X_i = < x_1, x_2, \ldots, x_i >$

  and call it the **i^{th} prefix of X.**
- **For example:**
- if X = <A; B; C; B; D; A; B>, then

  4^{th} prefix: X4 = <A; B; C; B> and
- X0 is the empty sequence.

LCS Notation

Let X and Y be sequences.

LCS(X, Y) represent :

– the set of longest common subsequences of X and Y.

Optimal Substructure

- Let $X = < x_1, x_2, \ldots, x_m >$

and $Y = < y_1, y_2, \ldots, y_n >$ be two sequences.

- Let $Z = < z_1, z_2, \ldots, z_k >$ is any LCS of X and Y.

a) **Case1:** If $x_m = y_n$ then certainly $x_m = y_n = z_k$ and Z_{k-1} is in $LCS(X_{m-1}, Y_{n-1})$

Optimal Substructure cont..

Let $X = < x_1, x_2, \ldots, x_m >$
and $Y = < y_1, y_2, \ldots, y_n >$ be two sequences.
Let $Z = < z_1, z_2, \ldots, z_k >$ is any LCS of X and Y.

b) **Case2:** If $x_m \neq y_n$ then $x_m \neq z_k$ implies that **Z is in $LCS(X_{m-1}, Y)$**

c) **Case3:** If $x_m \neq y_n$ then $y_n \neq z_k$ implies that **Z is in $LCS(X, Y_{n-1})$**

Overlapping Subproblems

- If $x_m = y_n$ then we solve the subproblem to find an element in $LCS(X_{m-1}, Y_{n-1})$ and append x_m
- If $x_m \neq y_n$ then we solve the two subproblems of finding elements in **$LCS(X_{m-1}, Y_{n-1})$** and **$LCS(X_{m-1}, Y_{n-1})$** and choose the longer one.

Recursive Solution

Let X and Y be sequences.
Let **c[i,j]** be ***the length of an LCS*** of the sequences Xi and Yj

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

optimal substructure of the LCS problem

- The optimal substructure of the LCS problem gives the recursive formula

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases} \quad (15.9)$$

Dynamic Programming Solution

•To compute length of an element in LCS(X,Y) with X of length m and Y of length n,

we do the following:

- Initialize first row and first column of the array c with 0.
- **Calculate:** c[1,j] for 1 <= j <= n,
  c[2,j] for 1 <= j <= n
  …
- Return **c[m, n]**
- **Complexity O(mn).**

Dynamic Programming Solution cont..

- **How can we get an actual longest common subsequence?**
- Store in addition to the array c an array b pointing to the optimal subproblem chosen when computing c[i,j].

Example

LCS Example-1

- Consider the two sequences
- X = <A, B, C, B, A>
- Y = <B, D, C, A, B>

LCS Example-1

$$x_i = y_j \Rightarrow c[i,j] = c[i-1, j-1] + 1 \quad \nwarrow$$

$$x_i \neq y_j \Rightarrow c[i-1, j] \geqslant c[i, j-1]$$
$$c[i,j] = c[i-1, j] \quad \uparrow$$

$$c[i-1, j] < c[i, j-1]$$
$$c[i,j] = c[i, j-1] \quad \leftarrow$$

LCS Example-1

	j	→	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**B**
↓	x_i	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 ↖
5	**A**	0	1 ↑	1 ↑	2 ↑	3 ↖	3 ↑

Thus the optimal LCS length is *c[m,n]* = 3.

LCS Algorithm

- **Computing the length of an LCS**
  LCS-LENGTH (X, Y)
  - stores the c[i,j] values in a table c[0…m,0…n], and it computes the entries in row-major order.
- **Constructing an LCS**
  - **PRINT-LCS(b, X, i, j)**
  - maintains the table **b[1…m; 1…n]** to help us construct an optimal solution
    b[i,j] **points to the table entry corresponding to the optimal subproblem solution chosen when computing c[i,j].**

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i - 1, j - 1] + 1
            b[i, j] = "↖"
        elseif c[i - 1, j] ≥ c[i, j - 1]
            c[i, j] = c[i - 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j - 1]
            b[i, j] = "←"
return c and b
```

PRINT-LCS

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i - 1, j - 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i - 1, j)
else PRINT-LCS(b, X, i, j - 1)
```

Analysis

- since each table entry takes O(1) time to compute.
- The running time of the procedure is **Θ (mn).**

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)

i	j	0	1	2	3	4	5
		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[6,5]

LCS Example (1)

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

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

LCS Example (2)

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**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for j = 0 to n c[0,j] = 0

LCS Example (3)

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** ↑				
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

case i=1 and j=1
A != B
but, c[0,1]>=c[1,0]
so c[1,1] = c[0,1], and b[1,1] = ↑

LCS Example (4)

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** ↑			
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

case i=1 and j=2
A != D
but, c[0,2]>=c[1,1]
so c[1,2] = c[0,2], and b[1,2] = ↑

LCS Example (5)

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** ↑		
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

case i=1 and j=3
A != C
but, c[0,3]>=c[1,2]
so c[1,3] = c[0,3], and b[1,3] = ↑

LCS Example (6)

	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					

case i=1 and j=4
A = A
so c[1,4] = c[0,2]+1, and b[1,4] = ↖

LCS Example (7)

	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					
3	C	0					
4	B	0					

case i=1 and j=5
A != B
this time c[0,5]<c[1,4]
so c[1,5] = c[1, 4], and b[1,5] = ←

LCS Example (8)

	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					

case i=2 and j=1
B = B
so c[2, 1] = c[1, 0]+1, and b[2, 1] = ↖

LCS Example (9)

	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 ←			
3	C	0					
4	B	0					

case i=2 and j=2
B != D
and c[1, 2] < c[2, 1]
so c[2, 2] = c[2, 1] and b[2, 2] = ←

LCS Example (10)

	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 ←		
3	C	0					
4	B	0					

case i=2 and j=3
B != D
and c[1, 3] < c[2, 2]
so c[2, 3] = c[2, 2] and b[2, 3] = ←

LCS Example (11)

	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 ↑	
3	C	0					
4	B	0					

case i=2 and j=4
B != A
and c[1, 4] = c[2, 3]
so c[2, 4] = c[1, 4] and b[2, 2] = ↑

LCS Example (12)

	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					
4	B	0					

case i=2 and j=5
B = B
so c[2, 5] = c[1, 4]+1 and b[2, 5] = ↖

LCS Example (13)

	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 ↑				
4	B	0					

case i=3 and j=1
C != B
and c[2, 1] > c[3,0]
so c[3, 1] = c[2, 1] and b[3, 1] = ↑

LCS Example (14)

	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					

case i=3 and j= 2
C != D
and c[2, 2] = c[3, 1]
so c[3, 2] = c[2, 2] and b[3, 2] = ↑

LCS Example (15)

	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					

case i=3 and j= 3
C = C
so c[3, 3] = c[2, 2]+1 and b[3, 3] = ↖

LCS Example (16)

	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 ←	
4	B	0					

case i=3 and j= 4
C != A
c[2, 4] < c[3, 3]
so c[3, 4] = c[3, 3] and b[3, 3] =

LCS Example (17)

	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					

case i=3 and j= 5
C != B
c[2, 5] = c[3, 4]
so c[3, 5] = c[2, 5] and b[3, 5] = ↑

LCS Example (18)

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↖				

case i=4 and j=1
B = B
so c[4, 1] = c[3, 0]+1 and b[4, 1] = ↖

LCS Example (19)

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↖	1↑			

case i=4 and j=2
B != D
c[3, 2] = c[4, 1]
so c[4, 2] = c[3, 2] and b[4, 2] = ↑

LCS Example (20)

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↖	1↑	2↑		

case i=4 and j= 3
B != C
c[3, 3] > c[4, 2]
so c[4, 3] = c[3, 3] and b[4, 3] = ↑

LCS Example (21)

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↖	1↑	2↑	2↑	

case i=4 and j=4
B != A
c[3, 4] = c[4, 3]
so c[4, 4] = c[3, 4] and b[3, 5] = ↑

LCS Example (22)

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↖	1↑	2↑	2↑	3↖

case i=4 and j=5
B= B
so c[4, 5] = c[3, 4]+1 and b[4, 5] = ↖

LCS Algorithm Running Time

- LCS algorithm calculates the values of each entry of the array c[m,n]
- So the running time is clearly O(mn) as each entry is done in 3 steps.
- Now how to get at the solution?
- We use the arrows we created to guide us.
- We simply follow arrows back to base case 0

Finding LCS

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**	**1**	**2**	**2**	**3**

Finding LCS (2)

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**	**1**	**2**	**2**	**3**

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

LCS (straight order): **B C B**
(this string turned out to be a palindrome)

```
LCS-Length(X, Y)
m = length(X),  n = length(Y)
for i = 1 to m
     do c[i, 0] = 0
for j = 0 to n
      do c[0, j] = 0
for i = 1 to m
        do for j = 1 to n
              do if ( x_i = = y_j )
                    then c[i, j] = c[i - 1, j - 1] + 1
                           b[i, j] = "←↑ "
              else if c[i - 1, j]>=c[i,  j - 1]
                           then  c[i, j] = c[i - 1, j]
                                 b[i, j] = "↑"
                           else c[i, j] = c[i, j - 1]
                                 b[i, j] = "←"
return c and b
```

End