

From Divide-and-Conquer ...

- Essence: solve problems by combining the solutions to subproblems.
- Divide-and-conquer algorithms
 - Partition a problem into independent subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem.
 - –What's the potential problem??

 ⇒ Inefficient to solve repeated subproblems
- Therefore, if many independent subproblems are overlapping
 - ⇒ only need to each subproblem just once.
 - ⇒ Dynamic programming

Review Problems

- What is Order Statistics?
 - How to design an efficient Algorithm?
 - Worst-case complexity? Average-case complexity?
 - How to analyze the average-case of RAND-SELECT?
 - What's the key idea to guarantee O(n)? Hint: BFPRT-Select
 - Why groups of 5? Can we use 3?
- Back to Quick Sort
 - How to prove the average-case complexity?
 - Can guarantee O(nlgn) if using BFPRT-Select?

ECM5605(S20) L04-2

Birth of Dynamic Programming

- Richard Bellman, 1952
 - -multistage stochastic decision processes
- "Dynamic Programming" so named because
 - Originally associated with movements (trading) between time and space, thus "dynamic"
 - –"programming" refers to the process of formulating the constraints of a problem
 ⇒ by analogy to "linear programming" and other forms of optimization
- Can be also used in deterministic problems
 ⇒ (recursive) optimization problems

ECM5605(S20) L04-3 ECM5605(S20) L04-4

Dynamic Programming (DP)

- Typically apply to optimization problems:
 - Find a (not "the") solution with the optimal (maximum / minimum) value
- 4 steps in developing a DP algorithm
 - 1. Characterize the structure of an optimal solution.
 - 2. Recursively define the formula of an optimal solution
 - 3. Compute the value of an optimal solution either **bottom-up** in a table (or top-down with caching)
 - 4. Construct an optimal solution from computed information (this step can be omitted if only the value is required)

ECM5605(S20)

When to Use DP

- Characteristics of DP problems:
 - 1. problem can be divided into **stages** (subproblems)
 - 2. each stage has one/more states (substructures)
 - 3. you make a decision at each stage
 - 4. the decision you make affects the state for the next stage
 - 5. there is a *recursive relationship* between the value of the decision at the stage and the previously found optima (principle of optimality)
- **Hopeless configurations:** for an *n*-element set
 - # of permutations: n!
 - # of subsets: 2ⁿ

L04-6 ECM5605(S20)

Longest Common Subsequence

Problem Formulation:

Given 2 sequences, $\mathbf{X} = \langle x_1, ..., x_m \rangle$ and $\mathbf{Y} =$ $\langle y_1, ..., y_n \rangle$, find a common subsequence whose length is maximum.

Example: Subsequence need not be consecutive, but must be in order.

X: springtime

ECM5605(S20)

Y: printing

ncaa tournament north carolina

basketball krzyzewski

printi

ncarna

ke

Naïve Algorithm and Analysis

- Brute-force LCS algorithm: For every subsequence of X, check whether it's a subsequence of Y
- Analysis:
 - -Each subsequence takes O(n) time to check: scan Y for first letter, for second, and so on.
 - -2^m subsequences of X to check.
 - -Worst-case runtime: O(**n2**^m) exponential time
- Read code for the naïve version and watch the number of comparisons

L04-7 ECM5605(S20) L04-8

Towards a Better Algorithm

- Simplification:
 - 1. Look at the *length* of a longest-common subsequence
 - Denote the length for a sequence of S by |S|
 - 2. Extend the algorithm to find the LCS itself
- Strategy: consider prefixes of X and Y
 - Define c[i,j] = |LCS(X[1..i],Y[1..j])|
 - Then c[m,n] = |LCS(X,Y)|

ECM5605(S20) L04-

Step 1: Optimal Substructure of LCS (cont'd)

Theorem

Let $Z = \langle z_1, \dots, z_k \rangle$ be any LCS of X and Y. 1 If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1} . 2 If $x_m \neq y_n$, then either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y. 3 or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1} .

Proof: (case 2: $x_m \neq y_n$, and $z_k \neq x_m$)

Since Z does not end in x_m ,

- 1) Z is a common subsequence of X_{m-1} and Y, and
- 2) there is no *longer* CS of X_{m-1} and Y, or Z would not be an LCS

Step 1: Optimal Substructure of LCS

Theorem

Let $Z = \langle z_1, \ldots, z_k \rangle$ be any LCS of X and Y. 1 If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1} . 2 If $x_m \neq y_n$, then either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y. 3 or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1} .

Proof: (case 1: if $x_m = y_n$)

Any sequence **Z**' that does not end in $x_m = y_n$ can be made longer by adding $x_m = y_n$ to the end. Therefore,

- 1) longest common subsequence (LCS) Z must end in $x_m = y_n$.
- 2) Z_{k-1} is a common subsequence of X_{m-1} and Y_{n-1} , and
- 3) there is no *longer* CS of X_{m-1} and Y_{n-1} , or Z_{k-1} would not be an LCS.

ECM5605(S20) L04-10

Step 2: Recursive Formula for Solution

Theorem

Let $Z = \langle z_1, \ldots, z_k \rangle$ be any LCS of X and Y. 1 If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1} 2 If $x_m \neq y_n$, then either $z_k \neq x_m$ and Z is an LCS of X_{m-1} and Y3 or $z_k \neq y_n$ and Z is an LCS of X and Y_{n-1}



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

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?

$$LCS(X, Y) = BCB$$

$$X = AB$$
 C B

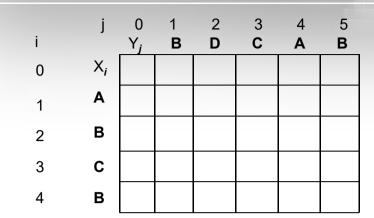
$$Y = BDCAB$$

ECM5605(S20)

L04-13

L04-15

LCS Example (I)



$$X = ABCB; m = |X| = 4;$$

 $Y = BDCAB; n = |Y| = 5;$ \Rightarrow Allocate 5x6 Matrix **C**

ECM5605(\$20) L04-14

LCS Example (1)

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

if
$$(x_i == y_j)$$
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

L04-16

ABCB BDCAB	LCS Example (3)										
i	j	0 Yj	1 B	2	30	4 A	5 B				
0	Xi	0	0	0	0	0	0				
1	(A)	0	0	0	0						
2	В	0									
3	С	0									
4	В	0									

if
$$(x_i == y_j)$$
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

ECM5605(S20) L04-17

ABCB BDCAB	l	LCS Example (4)										
i	j	0 Yj	1 B	2 D	3 C	4 (A)	5 B					
0	Xi	0	0	0	0 、	0	0					
1	(A)	0	0	0	0	1						
2	В	0										
3	С	0										
4	В	0										
if $(x_i == y_j)$ $c[i,j] = c[i-1,j-1] + 1$ else $c[i,j] = max(c[i-1,j], c[i,j-1])$												

ECM5605(S20) L04-18

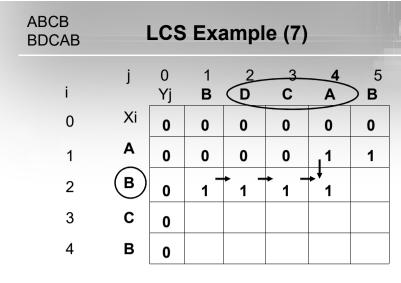
ABCB BDCAB	LCS Example (5)									
i	j	0 Yj	1 B	2 D	3 C	4 A	(B)			
0	Xi	0	0	0	0	0	0			
1	A	0	0	0	0	1 -	+ 1			
2	В	0								
3	С	0								
4	В	0								

ECM5605(S20)

if (
$$x_i == y_j$$
) $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

if
$$(x_i == y_j)$$
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

ECM5605(S20) L04-20



if
$$(x_i == y_j)$$
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

ECM5605(S20)

ABCB BDCAB		LCS Example (8)										
	j	0	1	2	3 C	4	_5					
		Yj	В	D	С	Α	(B)					
0	Xi	0	0	0	0	0	0					
1	Α	0	0	0	0	1 ,	1					
2	В	0	1	1	1	1	2					
3	С	0										
4	В	0										
if $(x_i == y_j)$ $c[i,j] = c[i-1,j-1] + 1$ else $c[i,j] = max(c[i-1,j], c[i,j-1])$												

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

ECM5605(S20) L04-22

ABCB BDCAB	LCS Example (10)									
i	j	0 Yj	B		3 C	4 A	5 B			
0	Xi	0	0	0	0	0	0			
1	Α	0	0	0	0	1	1			
2	В	0	. 1	.1	1	1	2			
3	(c)	0	† ₁ -	→ 1						
4	В	0								

ECM5605(S20)

if (
$$x_i == y_j$$
) $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

L04-23

if
$$(x_i == y_j)$$
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

L04-24 ECM5605(S20)

ABCB BDCAB	L	.cs	Exa	mple	e (12)		F
i	j	0 Yj	1 B	2 D	3 C	A	5 B	
0	Xi	0	0	0	0	0	0	
1	Α	0	0	0	0	1	1	
2	В	0	1	1	1	1	,2	
3	C	0	1	1	2 -	→ ₂ –	→ 2	
4	В	0						

if (
$$x_i == y_j$$
) $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

ECM5605(S20) L04-25

ABCB BDCAB	L	.cs	Exa	mple	e (13)		
	i	0	1	2	3	4	5	
i	•	Yj	B	D	C	A	В	
0	Xi	0	0	0	0	0	0	
1	A	0	0	0	0	1	1	
2	В	0	1	1	1	1	2	
3	C	0 、	1	1	2	2	2	
4	B	0	1					
	if (x _i else	== y _j) c[i, c[i,	.j] = c .j] = n	[i-1,j- nax(c	1] + 1 [i-1,j],	c[i,j-	1]

ECM5605(S20) L04-26

ABCB BDCAB	L	LCS Example (14)										
i	j	0 Yj	1 B	2	<u>з</u> С	4 A	5 B					
0	Xi	0	0	0	0	0	0					
1	Α	0	0	0	0	1	1					
2	В	0	1	1	1	1	2					
3	C	0	1	,1	_2	2	2					
4	В	0	1 -	→ [†] 1 -	+ [†] 2 -	† 2						

ECM5605(S20)

if (
$$x_i == y_j$$
) $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j],c[i,j-1])$

L04-27

$$\begin{array}{c} if \ (\ x_i == y_j \) \ c[i,j] = c[i\text{-}1,j\text{-}1] + 1 \\ else \ c[i,j] = max(\ c[i\text{-}1,j],\ c[i,j\text{-}1] \) \end{array}$$

LCS Algorithm Runtime

- LCS algorithm calculates the values of each entry of the array c[m,n]
- So what is the runtime to compute **C**?

$O(m \times n)$

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

ECM5605(S20) L04-29

Step 4: Construct a LCS (cont'd)

Remember that

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

- So we can start from c[m,n] and go backwards
- Whenever c[i,j] = c[i-1, j-1]+1, 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

Step 4: Construct a LCS

- So far, we have just found the *length* of LCS, but not a 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 either 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:



ECM5605(S20)

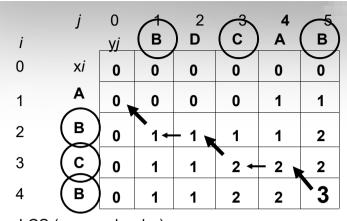
For example, here c[i,j] = c[i-1,j-1] + 1 = 2+1=3

ECM5605(S20) L04-30

Example of Constructing a LCS

	j	0	1	2	3	4	5
i		уј	В	D	С	Α	В
0	Хİ	0		0	0	0	0
1	Α	0,	0	0	0	1	1
2	В	0	1	- 1 _K	1	1	2
3	С	0	1	1	(2)+	- 2	2
4	В	0	1	1	2	2	(3)

Example of Constructing a LCS (cont'd)



LCS (reversed order):

в с в

LCS (straight order):

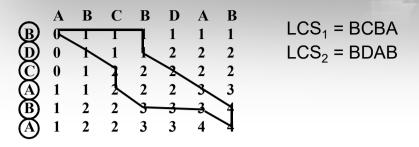
ВСВ

(this string turned out to be a palindrome)

ECM5605(S20)

_04-33

Analysis DP on LCS Problem

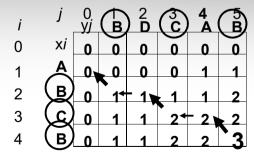


- Space complexity: $O(m \times n)$
- Time complexity:
 - -Computing the table: time = $O(m \times n)$
 - -Constructing an LCS: time = O(m+n)

ECM5605(\$20) L04-34

Optimizing Space in LCS

Recap memory usage



- –Do we really need a *m*×*n* table to store the LCS values? Any improvement?
- -Implement your space-optimized version

Summary

- What are common/different points between Divide-and-Conquer and DP?
- 2 Hallmarks of dynamic programming:
 - Optimal substructure: an optimal solution to a problem (instance) contains optimal solutions to subproblems
 - Overlapping subproblems: a recursive solution contains a "small" number of distinct subproblems repeated many times
- What is longest common sequence problem?
 - How to use DP to solve this problem
 - Space complexity? Time complexity??

ECM5605(S20) L04-35 ECM5605(S20) L04-36

Ex: Longest Increasing Subsequence

- The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.
- For example, length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80 } is 6 $-LIS = \{10, 22, 33, 50, 60, 80\}$

DP for LIS

• Let arr be the input array and L(i) be the length of the LIS ending at index i $\Rightarrow arr[i]$ is the last element of the LIS

■ Then, *L*(*i*) can be recursively written as:

$$L(i) = \begin{cases} 1 + \max(L(j)), & \text{if } 0 < j < i \text{ and } arr[j] < arr[i] \\ 1, & \text{if no such } j \text{ exists} \end{cases}$$

- Complexity is ???
- Refer to LIS v1.cpp and develop your DP solution in C/C++

ECM5605(S20) L04-37 ECM5605(S20) L04-38