COMP 355
Advanced Algorithms
Dynamic Programming:
Knapsack & LCS
Section 6.4 (KT)

Rhodes College

1

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*.

Rhodes College

2

Knapsack Problem

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.

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Ex: { 3, 4 } has value 40.

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

3

Dynamic Programming: False Start

Def. OPT(i) = max profit subset of items 1, …, i.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 }
- Case 2: 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!

4

Dynamic Programming: Adding a New Variable

Def. OPT(i, W) = max profit subset of items 1, …, i with weight limit W.

- Case 1: OPT does not select item i.
  - OPT selects best of { 1, 2, …, i-1 } using weight limit W
- Case 2: OPT selects item i.
  - new weight limit = W – w_i
  - OPT selects best of { 1, 2, …, i–1 } using this new weight limit

$$OPT(i,W) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,W) & \text{if } w_i > W \\ \max\{OPT(i-1,W), v_i + OPT(i-1, W-w_i)\} & \text{otherwise} \end{cases}$$

5

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]
```

6

Knapsack Algorithm

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	34	40

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

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Problem: Running Time

Running time. Θ(n W).

- 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.

8

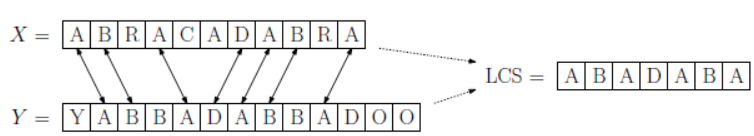
Using DP to compare strings

- 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.

9

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 $\langle i_1, i_2, \ldots, i_k \rangle$ ($1 \le i_1 < i_2 < \ldots < i_k \le n$) such that $Z = \langle x_{i1}, x_{i2}, \ldots, x_{ik} \rangle$.
For example, let X = <ABRACADABRA> and let Z = <AADAA>, 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.

10

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: $O(n\ 2^m)$

Rhodes College

11

DP Formulation for LCS

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, $X_i = \langle x_1, \ldots, x_i \rangle$. X_0 is the empty sequence.

Let 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, lcs(5, 6) = 3.

Rhodes College

12

DP Formulation for LCS

Base Case: lcs(i, 0) = lcs(j, 0) = 0.

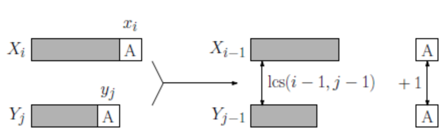
Last characters match: Suppose $x_i = y_j$. For example: Let X_i = <ABCA> and let Y_j = <DACA>. Since both end in 'A', it is easy to see that the LCS must also end in 'A'.

Last characters do not match: Suppose that x_i != y_j . x_i and y_j cannot both be in the LCS. 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).

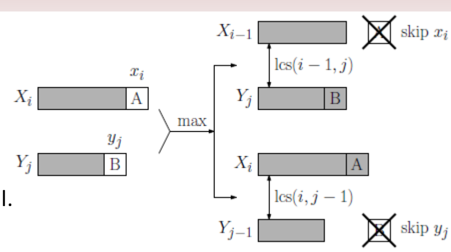
- Option 1: (x_i is not in the LCS) - the LCS of X_i and Y_j is the LCS of X_i−1 and Y_j , which is given by lcs(i – 1, j).
- Option 2: (y_j is not in the LCS) - the LCS of X_i and Y_j−1, which is given by lcs(i, j – 1).

13

Recursive Formulation of LCS



LCS of two strings whose last characters are equal.
if (x_i = y_j) then lcs(i, j) = lcs(i – 1, j – 1) + 1



The possible cases in the DP formulation of LCS.
if (x_i != y_j) then lcs(i, j) = max(lcs(i – 1, j), lcs(i, j – 1))

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

14

Memoized Implemention

Memoized Longest Common Subsequence

```
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 already computed - 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).

Bottom-up implementation

Bottom-up Longest Common Subsequence

```
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) {                     // fill rest of table
        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)

Rhodes College
—1848—

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

ABCB
BDCAB

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[5,4]

LCS Example

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      lcs[i,0] = 0
for j = 1 to n      lcs[0,j] = 0
```

19

LCS Example

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 )
        lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

20

LCS Example

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

```
if ( Xi == Yj )
     lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

21

LCS Example

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

```
if ( Xi == Yj )
     lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

22

LCS Example

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

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

23

LCS Example

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

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

24

LCS Example

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	
3	C		0					
4	B		0					

```
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] )
```

25

LCS Example

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					
4	B		0					

```
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] )
```

26

LCS Example

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			
4	B		0					

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

27

LCS Example

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		
4	B		0					

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

28

LCS Example

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 ( Xi == Yj )
        lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

29

LCS Example

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 ( Xi == Yj )
        lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

30

LCS Example

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	1	2	2	

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

31

LCS Example

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	1	2	2	3

```
if ( Xi == Yj )
      lcs[i,j] = lcs[i-1,j-1] + 1
else lcs[i,j] = max( lcs[i-1,j], lcs[i,j-1] )
```

32

Adding Hints to Reconstruct LCS

add$_{XY}$: Add $x_i (= y_j)$ to the LCS ('↖') and continue with $\text{lcs}[i-1, j-1]$

skip$_X$: Do not include x_i to the LCS ('↑') and continue with $\text{lcs}[i-1, j]$

skip$_Y$: Do not include y_j to the LCS ('←') and continue with $\text{lcs}[i, j-1]$

Bottom-up Longest Common Subsequence with Hints

```
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[i,0] = 0;  h[i,0] = skipX }
    for (j = 0 to n) { lcs[0,j] = 0;  h[0,j] = 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
}
```

33

Extracting the LCS

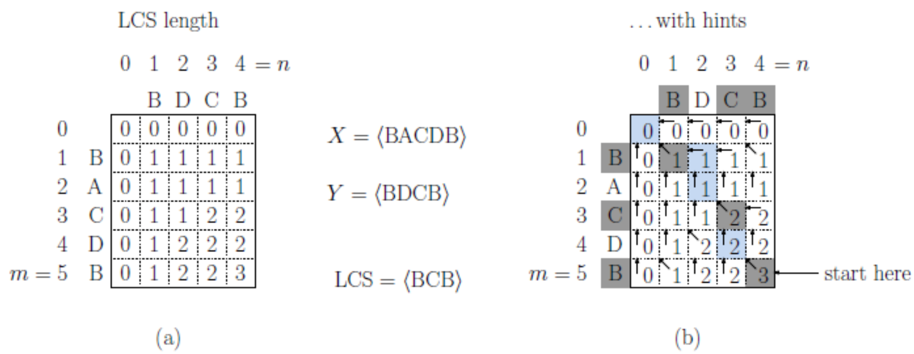
Extracting the LCS using the Hints

```
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
}
```

Rhodes College

34

LCS Example



Contents of the lcs array for the input sequences X = <BACDB> and Y = <BCDB>. The numeric table entries are the values of lcs[i, j] and the arrow entries are used in the extraction of the sequence.