

DYNAMIC PROGRAMMING

David Kauchak  
CS 140 – Fall 2024

1

### Admin

- Assignment 3
- Assignment 4

2

### Algorithmic "techniques"

**Iterative/incremental:** solve problem of size  $n$  by first solving problem of size  $n-1$ .

**Divide-and-conquer:** divide problem into independent subproblems. Solve each subproblem independently. Combine solutions to subproblem to create solution to the original problem.

5

### Dynamic programming

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to subproblems

**AND**  
the subproblems are overlapping

6

### Fibonacci: a first attempt

```

FIBONACCI(n)
1  if n = 1 or n = 2
2     return 1
3  else
4     return FIBONACCI(n - 1) + FIBONACCI(n - 2)
    
```

7

### Running time

```

FIBONACCI(n)
1  if n = 1 or n = 2
2     return 1
3  else
4     return FIBONACCI(n - 1) + FIBONACCI(n - 2)
    
```

Each call creates two recursive calls

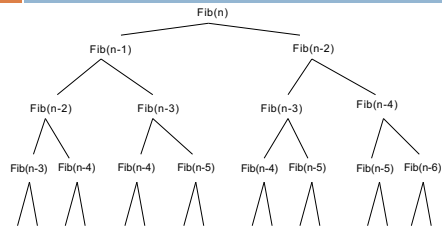
Each call reduces the size of the problem by 1 or 2:  
 $T(n) = T(n - 1) + T(n - 2) + O(1)$

Creates a full binary of depth n (where the shortest leaf is at depth  $n/2$ )

$O(2^n)$

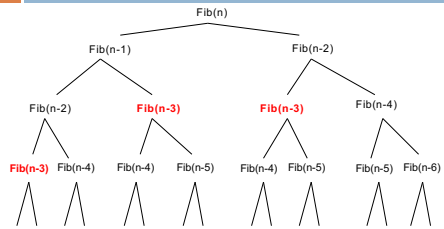
8

### Can we do better?



9

### A lot of repeated work!



10

### Identifying a dynamic programming problem

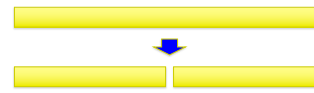
The solution can be defined with respect to solutions to subproblems

The subproblems created are *overlapping*, that is we see the same subproblems repeated

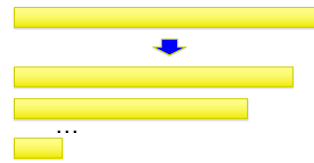
11

### Overlapping sub-problems

divide and conquer



dynamic programming



12

### Dynamic programming: steps

**recursive definition:** use this to recursively define the value of an optimal solution

**DP solution:** describe the dynamic programming table:

- size, initial values, order in which it's filled in, location of solution

**Analysis:** analyze space requirements, running time

14

### Recursive definition

Define a function and *clearly* define the inputs to the function

The function definition should be *recursive* with respect to *multiple subproblems*

- pretend like you have a working function, but it only works on smaller problems

Key: subproblems will be *overlapping*, i.e., inputs to subproblems will not be disjoint

17

## Recursive definition

Fibonacci:

$$F(n) = ?$$

18

## Recursive definition

Fibonacci:

$$F(n) = F(n-1) + F(n-2)$$

19

## DP solution

The recursive solution will generally be top-down, i.e., working from larger problems to smaller

DP solution:

- work bottom-up, from the smallest versions of the problem to the largest
- store the answers to subproblems in a table (often an array or matrix)
- to build bigger problems, lookup solutions in the table to subproblems

20

## DP solution

$$F(n) = F(n-1) + F(n-2)$$

What are the smallest possible values (subproblems)?

To calculate  $F(n)$ , what are all the subproblems we need to calculate? This is the "table".

How should we fill in the table?

21

## DP solution

$$F(n) = F(n-1) + F(n-2)$$

What are the smallest possible values (subproblems)?  $F(1) = 1, F(2) = 1$

To calculate  $F(n)$ , what are all the subproblems we need to calculate? This is the "table".  $F(1) \dots F(n-1)$

How should we fill in the table?  $F(1) \rightarrow F(n)$

22

## DP solution

```

FIBONACCI-DP(n)
1 fib[1] ← 1
2 fib[2] ← 1
3 for i ← 3 to n
4   fib[i] ← fib[i-1] + fib[i-2]
5 return fib[n]

```

Store the intermediary values in an array (*fib*)

23

## Analysis

```

FIBONACCI-DP(n)
1 fib[1] ← 1
2 fib[2] ← 1
3 for i ← 3 to n
4   fib[i] ← fib[i-1] + fib[i-2]
5 return fib[n]

```

Space requirements?

Running time?

24

## Analysis

```

FIBONACCI-DP(n)
1 fib[1] ← 1
2 fib[2] ← 1
3 for i ← 3 to n
4   fib[i] ← fib[i-1] + fib[i-2]
5 return fib[n]

```

Space requirements:  $\mathcal{O}(n)$

Running time:  $\mathcal{O}(n)$

25

### Counting binary search trees

How many unique binary search trees can be created using the numbers 1 through n?

26

### Step 1: What is the subproblem?

Assume we have a working function (call it T) that can give us the answer to smaller subproblems

How can we use the answer from this to answer our question?

How many options for the root are there?

27

### Subproblems

How many trees have i as the root?

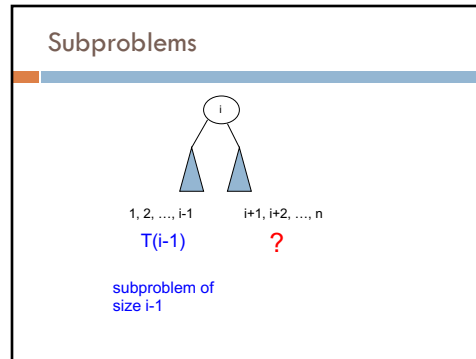
28

### Subproblems

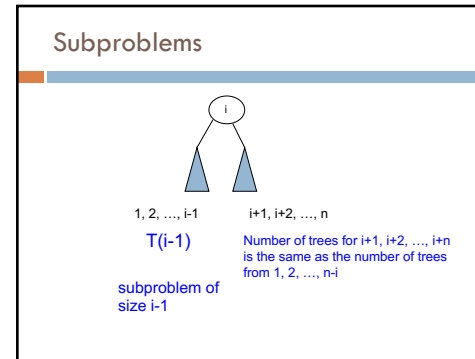
1, 2, ..., i-1      i+1, i+2, ..., n

?

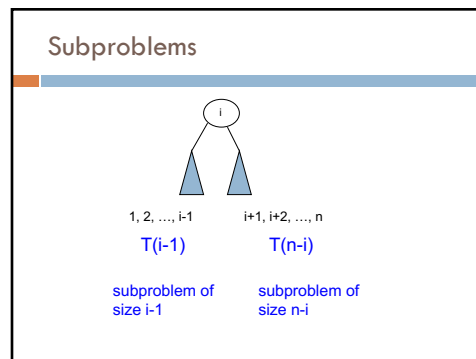
29



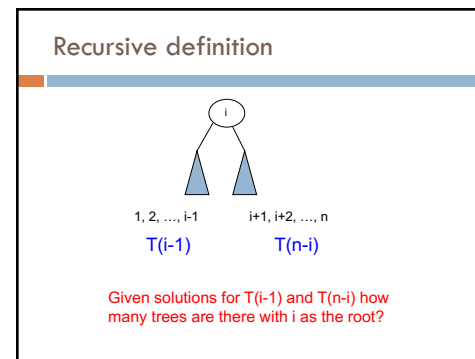
30



31



32



35

### Recursive definition

1, 2, ..., i-1      i+1, i+2, ..., n

$T(i-1)$        $T(n-i)$

Trees with  $i$  as root =  $T(i-1) * T(n-i)$

36

### Recursive definition

Trees with  $i$  as root =  $T(i-1) * T(n-i)$

How many trees total?

37

### Recursive definition

Trees with  $i$  as root =  $T(i-1) * T(n-i)$

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

38

### A recursive implementation

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

```

BST-COUNT(n)
1  if n = 0
2     return 1
3  else
4     sum = 0
5     for i ← 1 to n
6         sum ← sum + BST-COUNT(i-1) * BST-COUNT(n-i)
7  return sum
    
```

Like with Fibonacci, we're repeating a lot of work

39



## DP solution (from the bottom-up)

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

What are the smallest possible subproblems?

To calculate  $T(n)$ , what are all the subproblems we need to calculate? This is the "table".

How should we fill in the table?

40

## DP solution (from the bottom-up)

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

What are the smallest possible subproblems?

$T(0)=1, T(1) = 1$

Why do we need  $T(0)$  and why is it 1?

41

## DP solution (from the bottom-up)

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

What are the smallest possible subproblems?

$T(0)=1, T(1) = 1$



Need to think carefully about base cases/edge cases

42

## DP solution (from the bottom-up)

$$T(n) = \sum_{i=1}^n T(i-1) * T(n-i)$$

What are the smallest possible subproblems?

$T(0)=1, T(1) = 1$

To calculate  $T(n)$ , what are all the subproblems we need to calculate? This is the "table".  $T(0) \dots T(n-1)$

How should we fill in the table?  $T(0) \rightarrow T(n)$

43

### DP solution (from the bottom-up)

```

BST-COUNT(n)
1 if n = 0
2   return 1
3 else
4   sum = 0
5   for i ← 1 to n
6     sum ← sum + BST-COUNT(i - 1) * BST-COUNT(n - i)
7   return sum

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i - 1] * c[k - i]
7   return c[n]
    
```

44

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i - 1] * c[k - i]
7   return c[n]
    
```

Fill in the first 4 values

0 1 2 3 4 5 ... n

45

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i - 1] * c[k - i]
7   return c[n]
    
```

1 1  
0 1 2 3 4 5 ... n

46

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i - 1] * c[k - i]
7   return c[n]
    
```

$c[0]*c[1] + c[1]*c[0]$   
1 1 ↓  
0 1 2 3 4 5 ... n

47

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i-1] * c[k-i]
7 return c[n]
    
```

$c[0] * c[1] + c[1] * c[0]$

1	1	↓					
0	1	2	3	4	5	...	n

48

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i-1] * c[k-i]
7 return c[n]
    
```

$c[0] * c[1] + c[1] * c[2]$

1	1	2					
0	1	2	3	4	5	...	n

49

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i-1] * c[k-i]
7 return c[n]
    
```

$c[0] * c[2] + c[1] * c[1] + c[2] * c[0]$

1	1	2	↓				
0	1	2	3	4	5	...	n

50

```

BST-COUNT-DP(n)
1 c[0] = 1
2 c[1] = 1
3 for k ← 2 to n
4   c[k] ← 0
5   for i ← 1 to k
6     c[k] ← c[k] + c[i-1] * c[k-i]
7 return c[n]
    
```

$c[0] * c[2] + c[1] * c[1] + c[2] * c[0]$

1	1	2	5				
0	1	2	3	4	5	...	n

51

```

BST-COUNT-DP(n)
1  c[0] = 1
2  c[1] = 1
3  for k ← 2 to n
4      c[k] ← 0
5      for i ← 1 to k
6          c[k] ← c[k] + c[i-1] * c[k-i]
7  return c[n]

```

```

1 1 2 5 ...
0 1 2 3 4 5 ... n

```

52

### 3: Analysis

```

BST-COUNT-DP(n)
1  c[0] = 1
2  c[1] = 1
3  for k ← 2 to n
4      c[k] ← 0
5      for i ← 1 to k
6          c[k] ← c[k] + c[i-1] * c[k-i]
7  return c[n]

```

Space requirements?

Running time?

53

### 3: Analysis

```

BST-COUNT-DP(n)
1  c[0] = 1
2  c[1] = 1
3  for k ← 2 to n
4      c[k] ← 0
5      for i ← 1 to k
6          c[k] ← c[k] + c[i-1] * c[k-i]
7  return c[n]

```

Space requirements:  $\Theta(n)$

Running time:  $\Theta(n^2)$

54

### Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where

$$1 \leq i_1 < i_2 < \dots < i_k \leq n$$

$X = \text{A B A C D A B A B}$

$\text{ABA?}$

55

### Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = \underline{A}BACDABAB$

ABA

56

### Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = ABACDABAB$

ACA?

57

### Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = ABACDABAB$

ACA

58

### Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = ABACDABAB$

DCA?

59

## Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = \text{ABACDABAB}$

~~DSA~~

60

## Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = \text{ABACDABAB}$

AADAA?

61

## Subsequences

For a sequence  $X = x_1, x_2, \dots, x_n$ , a subsequence is a subset of the sequence defined by a set of increasing indices  $(i_1, i_2, \dots, i_k)$  where  $1 \leq i_1 < i_2 < \dots < i_k \leq n$

$X = \underline{\text{A}}\underline{\text{B}}\underline{\text{A}}\underline{\text{C}}\underline{\text{D}}\underline{\text{A}}\underline{\text{B}}\underline{\text{A}}\underline{\text{B}}$

AADAA

62

## Longest common subsequence (LCS)

Given two sequences  $X$  and  $Y$ , a **common subsequence** is a subsequence that occurs in both  $X$  and  $Y$

Given two sequences  $X = x_1, x_2, \dots, x_n$  and  $Y = y_1, y_2, \dots, y_n$

What is the longest common subsequence?

63

### LCS problem

Given two sequences X and Y, a **common subsequence** is a subsequence that occurs in both X and Y

Given two sequences  $X = x_1, x_2, \dots, x_n$  and  $Y = y_1, y_2, \dots, y_n$

*What is the longest common subsequence?*

$X = \text{A B C B D A B}$

$Y = \text{B D C A B A}$

64

### LCS problem

Given two sequences X and Y, a **common subsequence** is a subsequence that occurs in both X and Y

Given two sequences  $X = x_1, x_2, \dots, x_n$  and  $Y = y_1, y_2, \dots, y_n$

*What is the longest common subsequence?*

$X = \text{A B C B D A B}$

$Y = \text{B D C A B A}$

65

### Recursive solution

$X = \text{A B C B D A B}$

$Y = \text{B D C A B A}$

Assume you have a solver for smaller problems

68

### Recursive solution

$X = \text{A B C B D A ?}$

$Y = \text{B D C A B ?}$

Is the last character part of the LCS?

69

Recursive solution

X = ABCBDA?

Y = BDCAB?

Two cases: either the characters are the same or they're different

70

Recursive solution

X = ABCBDA A

Y = BDCABA A

LCS

The characters are part of the LCS

What is the recursive relationship?

If they're the same

$$LCS(X, Y) = LCS(X_{1..n-1}, Y_{1..m-1}) + x_n$$

71

Recursive solution

X = ABCBDAB

Y = BDCABA

LCS

If they're different

$$LCS(X, Y) = LCS(X_{1..n-1}, Y)$$

72

Recursive solution

X = ABCBDAB

Y = BDCABA

LCS

If they're different

$$LCS(X, Y) = LCS(X, Y_{1..m-1})$$

73



### Recursive solution

X = ABCBDAB  
 Y = BDCABA

X = ABCBDAB  
 Y = BDCABA

If they're different

?

74

### Recursive solution

X = ABCBDAB  
 Y = BDCABA

$$LCS(X, Y) = \begin{cases} 1 + LCS(X_{1..n-1}, Y_{1..m-1}) & \text{if } x_n = y_m \\ \max(LCS(X_{1..n-1}, Y), LCS(X, Y_{1..m-1})) & \text{otherwise} \end{cases}$$

*(for now, let's just worry about counting the length of the LCS)*

75

### DP solution

$$LCS(X, Y) = \begin{cases} 1 + LCS(X_{1..n-1}, Y_{1..m-1}) & \text{if } x_n = y_m \\ \max(LCS(X_{1..n-1}, Y), LCS(X, Y_{1..m-1})) & \text{otherwise} \end{cases}$$

What types of subproblem solutions do we need to store?

LCS(X<sub>1..j</sub>, Y<sub>1..k</sub>)

two different indices

76

### DP solution

$$LCS(X, Y) = \begin{cases} 1 + LCS(X_{1..n-1}, Y_{1..m-1}) & \text{if } x_n = y_m \\ \max(LCS(X_{1..n-1}, Y), LCS(X, Y_{1..m-1})) & \text{otherwise} \end{cases}$$

What types of subproblem solutions do we need to store?

LCS(X<sub>1..j</sub>, Y<sub>1..k</sub>)

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

77

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j	0	1	2	3	4	5	6
		i	y <sub>j</sub>	B	D	C	A	B	A
0	x <sub>i</sub>								
1	A								
2	B								
3	C								
4	B								
5	D								
6	A								
7	B								

For Fibonacci and tree counting, we had to initialize some entries in the array. Any here?

78

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j	0	1	2	3	4	5	6
		i	y <sub>j</sub>	B	D	C	A	B	A
0	x <sub>i</sub>		0	0	0	0	0	0	0
1	A		0						
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

Need to initialize values within 1 smaller in either dimension.

79

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j	0	1	2	3	4	5	6
		i	y <sub>j</sub>	B	D	C	A	B	A
0	x <sub>i</sub>		0	0	0	0	0	0	0
1	A		0						
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

How should we fill in the table?

80

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j	0	1	2	3	4	5	6
		i	y <sub>j</sub>	B	D	C	A	B	A
0	x <sub>i</sub>		0	0	0	0	0	0	0
1	A		0						
2	B		0						
3	C		0			?			
4	B		0						
5	D		0						
6	A		0						
7	B		0						

To fill in an entry, we may need to look:

- up one
- left one
- diagonal up and left

Just need to make sure these exist

81

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	?					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

LCS(A, B)

82

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

83

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	?		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

LCS(A, BDCA)

84

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	0	1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

LCS(A, BDCA)

85

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	?	
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCAB)

86

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCAB)

87

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	3	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Where's the final answer?

88

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i		y <sub>j</sub> BDCABA						
0	x <sub>i</sub>	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	3	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

Space requirements?  
Running time?

89

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

	j	0	1	2	3	4	5	6
i	$y_j$	B	D	C	A	B	A	
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	4	4
7	B	0	1	2	2	3	4	4

Space requirements:  $O(nm)$

Running time:  $O(nm)$

90

### The algorithm

```

LCS-LENGTH(X, Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0, 0] ← 0
4  for i ← 1 to m
5      c[i, 0] ← 0
6  for j ← 1 to n
7      c[0, j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if  $x_i = y_j$ 
11              $c[i, j] ← 1 + c[i-1, j-1]$ 
12         elseif  $c[i-1, j] > c[i, j-1]$ 
13              $c[i, j] ← c[i-1, j]$ 
14         else
15              $c[i, j] ← c[i, j-1]$ 
16  return c[m, n]
```

91

### The algorithm

```

LCS-LENGTH(X, Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0, 0] ← 0
4  for i ← 1 to m
5      c[i, 0] ← 0
6  for j ← 1 to n
7      c[0, j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if  $x_i = y_j$ 
11              $c[i, j] ← 1 + c[i-1, j-1]$ 
12         elseif  $c[i-1, j] > c[i, j-1]$ 
13              $c[i, j] ← c[i-1, j]$ 
14         else
15              $c[i, j] ← c[i, j-1]$ 
16  return c[m, n]
```

Base case initialization

92

### The algorithm

```

LCS-LENGTH(X, Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0, 0] ← 0
4  for i ← 1 to m
5      c[i, 0] ← 0
6  for j ← 1 to n
7      c[0, j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if  $x_i = y_j$ 
11              $c[i, j] ← 1 + c[i-1, j-1]$ 
12         elseif  $c[i-1, j] > c[i, j-1]$ 
13              $c[i, j] ← c[i-1, j]$ 
14         else
15              $c[i, j] ← c[i, j-1]$ 
16  return c[m, n]
```

Fill in the matrix

93

## The algorithm

```

LCS-LENGTH(X,Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0,0] ← 0
4  for i ← 1 to m
5      c[i,0] ← 0
6  for j ← 1 to n
7      c[0,j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if xi = yj
11             c[i,j] ← 1 + c[i-1,j-1]
12         elseif c[i-1,j] > c[i,j-1]
13             c[i,j] ← c[i-1,j]
14         else
15             c[i,j] ← c[i,j-1]
16  return c[m,n]

```

94

## The algorithm

```

LCS-LENGTH(X,Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0,0] ← 0
4  for i ← 1 to m
5      c[i,0] ← 0
6  for j ← 1 to n
7      c[0,j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if xi = yj
11             c[i,j] ← 1 + c[i-1,j-1]
12         elseif c[i-1,j] > c[i,j-1]
13             c[i,j] ← c[i-1,j]
14         else
15             c[i,j] ← c[i,j-1]
16  return c[m,n]

```

95

## The algorithm

```

LCS-LENGTH(X,Y)
1  m ← length[X]
2  n ← length[Y]
3  c[0,0] ← 0
4  for i ← 1 to m
5      c[i,0] ← 0
6  for j ← 1 to n
7      c[0,j] ← 0
8  for i ← 1 to m
9      for j ← 1 to n
10         if xi = yj
11             c[i,j] ← 1 + c[i-1,j-1]
12         elseif c[i-1,j] > c[i,j-1]
13             c[i,j] ← c[i-1,j]
14         else
15             c[i,j] ← c[i,j-1]
16  return c[m,n]

```

96

## Keeping track of the solution

Our LCS algorithm only calculated the length of the LCS between X and Y

What if we wanted to know the actual sequence?

97

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B D C A B A						
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	?	
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCAB)

98

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B D C A B A						
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCAB)

99

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B D C A B A						
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	?
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCABA)

100

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B D C A B A						
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0						
6	A	0						
7	B	0						

LCS(ABCB, BDCABA)

101

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B	D	C	A	B	A	B
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

How do we generate the solution from this?

102

$$LCS[i, j] = \begin{cases} 1 + LCS[i-1, j-1] & \text{if } x_i = y_j \\ \max(LCS[i-1, j], LCS[i, j-1]) & \text{otherwise} \end{cases}$$

		j						
		0	1	2	3	4	5	6
i	$y_j$	B	D	C	A	B	A	B
0	$x_i$	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

We can follow the arrows to generate the solution

BCBA

103