# CSCI 2270
# Data Structures and Algorithms
# Lecture 13—some more algorithms; finish binary search, start longest common subsequence

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 128 or ECCS 112
Wed 9:30am-11:30am
Thurs 10:00am-11:00am

## Administrivia

- First midterm exam is in grading
- I have to shift 0.5 office hour from Thursday to Wednesday permanently
- Read Recursion (chapter 4); concentrate on factorial and binary search sections for now
- Interview grading slots are posted
  - Find a slot and please don’t miss the appointment. We don’t promise you we can make a second date for this.

# Recursion

Recursive factorial is not so high performance…

- Recursive calls cost time
- Recursive functions with local variables cost *lots of space*

But other recursive methods are very useful.  Examples:

- Binary search of sorted array
- Solution to Towers of Hanoi problem
- Quicksort/heapsort/mergesort to sort arrays
- Processing fractals: Koch snowflake, Sierpinski triangle…

## Longest common subsequence

If we have 2 strings, how similar are they?

What if we answer by counting all the letters?

Is “man bites dog” equal to “dog bites man”?

What if we take order into account?

BANANA vs. NANA

This algorithm is used to compute genetic similarity (BLAST)

# Longest common subsequence

This is a problem for a 2D array!

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Note: Array has one extra row and column, filled with 0s

# Making the array

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

```
string a = “BANANA”;
string b = “NANA”;
int** lcs_array = new int*[length(b) + 1];
for (int k = 0; k < length(b) + 1; ++k)
        lcs_array[k] = new int[length(a) + 1];
```

# Computing the LCS

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Now we fill in the table according to this rule:

If there’s a cell in the array at row i and column j, and the letters of string b[i-1] and string a[j-1] match, we take the maximum of three numbers:

the number in row (i - 1), column j; or

the number in row i, column (j - 1); or

1 + the number in row (i - 1), column (j - 1).

# Computing the LCS

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

If there's a cell in the array at row i and column j, and the letters of string b[i-1] and string a[j-1] do not match, we take the maximum of two numbers:

the number in row (i - 1), column j; or

the number in row i, column (j - 1).

# First row

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | B | A | N | A | N | A |
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Compute first unknown cell, lcs_array[1][1]

Note that this cell isn't a match (b[0] == N, a[0] == B)

So we take the max of the cell to its left (lcs_array[1][0]) and the cell just above it (lcs_array[0][1]). That's still 0.

# First row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Compute next unknown cell, lcs_array[1][2]

Note that this cell isn't a match (b[0] == N, a[1] == A)

So we take the max of the cell to its left (lcs_array[1][0]) and the cell just above it (lcs_array[0][1]). That's still 0.

# First row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Compute lcs_array[1][3]

Note that this cell is a match (b[0] == N, a[2] == N)

So we give this cell the maximum value of the cell to its left (lcs_array[1][2]), the cell just above it (lcs_array[0][3]) and 1+the cell above and to the left of it (lcs_array[0][2]). That's 1.

# End of first row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | ? | ? | ? | ? | ? | ? |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Computing the rest of the first unknown row…

# Next row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| N | 0 | ? | ? | ? | ? | ? | ? |
| A | 0 | ? | ? | ? | ? | ? | ? |

Computing the second unknown row…

## Third row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| N | 0 | 0 | 1 | 2 | 2 | 3 | 3 |
| A | 0 | ? | ? | ? | ? | ? | ? |

Computing the third unknown row…

# Last row

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| N | 0 | 0 | 1 | 2 | 2 | 3 | 3 |
| A | 0 | 0 | 1 | 2 | 3 | 3 | 4 |

Computing the last unknown row… bottom right corner has the matching score.

# Finished alignment

| - | - | B | A | N | A | N | A |
|---|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 0 | 1 | 1 | 2 | 2 | 2 |
| N | 0 | 0 | 1 | 2 | 2 | 3 | 3 |
| A | 0 | 0 | 1 | 2 | 3 | 3 | 4 |

Backtracing the match is harder:

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| B | A | N | A | N | A |
| - | - | N | A | N | A |