ALGORITHMICS - Bachelor of Software Engineering, 2021

LAB GUIDE. SESSION 5

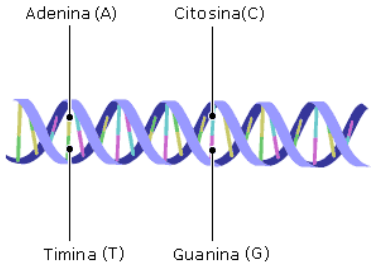
GOALS:

- **Dynamic programming: algorithms applied to bioinformatics**

1. Longest Common Subsequence (LCS)

Bioinformatics is about using computers to solve problems related to biology. Specifically, molecular biology is computer-dependent due to the large amount of information (millions of data) that normally needs to be processed.

A very typical example in bioinformatics is the analysis of DNA strands. Genetic material such as DNA consists of small elements called nucleotides. Nucleotides in DNA are two sequences of adenine (A), cytosine (C), thymine (T) and guanine (C).



One of the previous steps to a multitude of biological assays is to look for the **longest common subsequence** between two DNA sequences, as it will allow the calculation of the similarity between two DNA samples.

An important aspect is that the elements of the subsequence does not need to be consecutive. For example, given this sequence GCCCTAGCG, the GTA string is a subsequence but not a substring of GCCCTAGCG.

As an example, if we have the following DNA sequences: GCCCTAGCG and GCGCAATG, we could get different LCS as GCGCG or GCCAG, depending on the specific implementation of the algorithm.

2. Implementation

A) Direct recursive approach

The algorithm for finding a LCS directly (*findLongestSubseq(S1, S2)*) from two sequences, S1 and S2, can be implemented following the next subtraction recursive scheme:

1. Base case: Whenever S1 or S2 is a zero-length string. Then, returns an empty string.
2. Recursive calls (Sx' is Sx without the right-most character Cx):
    a. L1 = *findLongestSubseq*(S1', S2)
    b. L2 = *findLongestSubseq*(S1, S2')
    c. L3 = *findLongestSubseq*(S1', S2')
3. Combination, the maximum of:

a. L1
b. L2
c. L3 + 1 if C1 is equals to C2, otherwise L3 (being C1 and C2 the rightmost characters of S1 and S2 respectively)

A class called *LCSRec.java* is already provided, the student must implement methods *findLongestSubseq* and *longest* (optional).

B) Dynamic programming

A class called *LCS.java* is already provided, the student is asked to implement methods *fillTable*, *findLongestSubseq* and *longest* (optional).

Here two examples are provided to understand how to construct the table used for dynamic programing. For the previous two substrings, the table should look like the following, considering that each cell in the table has the value of the calculation (according to the algorithm to solve this problem) followed in brackets by the index (i , j) of the position of the cell in the table from which it has been calculated the corresponding value. Note that row 0 and column 0 are always initialized to 0 (0.0), which means that there is no maximum subsequence.

	*	*	G	C	C	C	T	A	G	C	G
*	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)
G	0(0, 0)	1(0, 0)	1(1, 1)	1(2, 1)	1(3, 1)	1(4, 1)	1(5, 1)	1(6, 0)	1(7, 1)	1(8, 0)	
C	0(0, 0)	1(1, 1)	2(1, 1)	2(2, 1)	2(3, 1)	2(4, 2)	2(5, 2)	2(6, 2)	2(7, 1)	2(8, 2)	
G	0(0, 0)	1(1, 2)	2(2, 2)	2(3, 2)	2(4, 2)	2(5, 2)	2(6, 2)	3(6, 2)	3(7, 3)	3(8, 2)	
C	0(0, 0)	1(1, 3)	2(2, 3)	3(2, 3)	3(3, 3)	3(4, 4)	3(5, 4)	3(7, 3)	4(7, 3)	4(8, 4)	
A	0(0, 0)	1(1, 4)	2(2, 4)	3(3, 4)	3(4, 4)	3(5, 4)	4(5, 4)	4(6, 5)	4(8, 4)	4(9, 4)	
A	0(0, 0)	1(1, 5)	2(2, 5)	3(3, 5)	3(4, 5)	3(5, 5)	4(6, 5)	4(7, 5)	4(8, 5)	4(9, 5)	
T	0(0, 0)	1(1, 6)	2(2, 6)	3(3, 6)	3(4, 6)	4(4, 6)	4(6, 6)	4(7, 6)	4(8, 6)	4(9, 6)	
G	0(0, 0)	1(1, 7)	2(2, 7)	3(3, 7)	3(4, 7)	4(5, 7)	4(6, 7)	5(6, 7)	5(7, 8)	5(8, 7)	

Printing maximum subsequence...

GCGCG

You only need to print one of the possible solutions for each case. Another example for GCATGCAT and GAATTCAG sequences would be:

	*	*	G	C	A	T	G	C	A	T
*	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)	0(0, 0)
G	0(0, 0)	1(0, 0)	1(1, 1)	1(2, 1)	1(3, 1)	1(4, 0)	1(5, 1)	1(6, 1)	1(7, 1)	
A	0(0, 0)	1(1, 1)	1(2, 1)	2(2, 1)	2(3, 2)	2(4, 2)	2(5, 2)	2(6, 1)	2(7, 2)	
A	0(0, 0)	1(1, 2)	1(2, 2)	2(3, 2)	2(4, 2)	2(5, 2)	2(6, 2)	3(6, 2)	3(7, 3)	
T	0(0, 0)	1(1, 3)	1(2, 3)	2(3, 3)	3(3, 3)	3(4, 4)	3(5, 4)	3(7, 3)	4(7, 3)	
T	0(0, 0)	1(1, 4)	1(2, 4)	2(3, 4)	3(4, 4)	3(5, 4)	3(6, 4)	3(7, 4)	4(8, 4)	
C	0(0, 0)	1(1, 5)	2(1, 5)	2(3, 5)	3(4, 5)	3(5, 5)	4(5, 5)	4(6, 6)	4(8, 5)	
A	0(0, 0)	1(1, 6)	2(2, 6)	3(2, 6)	3(4, 6)	3(5, 6)	4(6, 6)	5(6, 6)	5(7, 7)	
G	0(0, 0)	1(1, 7)	2(2, 7)	3(3, 7)	3(4, 7)	4(4, 7)	4(6, 7)	5(7, 7)	5(8, 7)	

Printing maximum subsequence...

GATCA

3. Questions

A) Determine theoretically complexities (time, memory space and waste of stack) for both implementations, recursive (approximated) and using programming dynamic.

B) Compute theoretical times and compare them with the experimental measurements.

C) Why large sequences cannot be processed with the recursive implementation? Explain why dynamic programing implementation raises and exception for large sequences.

D) The amount of possible LCS can be more than one, e. g. GCCCTAGCG and GCGCAATG has two GCGCG and GCCAG. Find the code section that determines which subsequence is chosen, modify this code to verify that both solutions can be achieved.

TO DO:

A. Work to be done

- An `algstudent.s5` **package** in your course project. The content of the package should be:
  - `LCSRec.java`. Class for solution with the recursive approach, the student must complete the missing code.
  - `LCS.java`. Class for solution with dynamic programing, the student must complete the missing code.
  - `LCSTest.java`. Class to validate implementations using specific examples.
  - `LCSTimes.java`. Class to measure experimental times.
- A **PDF document** (`session5.pdf`) using the course template. The activities of the document should be the following:
  - **Activity 1. Validation results.**
  - **Activity 2. Experimental time measurements (see tables below).**
  - **Activity 3. Answer to questions in Section 3 (A, B, C and D).**

n	t_dynamic
100	
200	
400	
800	
1600	
3200	
6400	
...	
Until crashes	

n	t_recursive
2	
4	
6	
8	
10	
12	
14	
...	
Until intractable	

B. Delivery method

First, you should include in your Java project a new `algstudent.s5` package with the following content inside it:

- All the requested source files for that package.
- The requested PDF document called `session5.pdf` with the corresponding activities.

Second, push the code and PDF to your project repository, afterwards zip the code and PDF and submit it to the corresponding Campus Virtual task.

Deadlines:

- The deadline is one day before the next lab session of your group.