

Program 2

Due Oct 31 by 11:59pm **Points** 30 **Submitting** a text entry box or a file upload
Available until Oct 31 at 11:59pm

This assignment was locked Oct 31 at 11:59pm.

Longest common subsequence of two strings

(edited: change is in bold-red text)

Purpose

This programming assignments implements the Longest Common Subsequence (LCS) problem. LCS algorithm is applied to many real-world problems including DNA/protein sequence alignment in bioinformatics, and computational linguistics. You will implement the algorithm using iteration and recursion, analyze and compare the two implementations.

Note: Note: When generating log file of iterative program for part2, you can use repl.it, since the school Linux server is so fast, it takes 0 times most of the cases. (edited: 10/26/2020)

The Longest Common Subsequence problem

A subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. Longest common subsequence (LCS) of 2 sequences is a subsequence, with **maximal length**, which is common to both the sequences. Note that it is different from the longest common **substring** problem, which requires to occupy consecutive positions within the original sequences. The *LCS* problem here, does not require to have consecutive positions.

For example,

If the two strings are

AAC**CTT**GG

AC**ACTG**TGA

then the LCS resulting string is

A**ACT**GG

and the LCS length =6

Note that the subsequence is not unique, but the longest length is always the same.

From the above example,

AAC**CTT**GG

AC**ACTG**TGA

The LCS resulting string can be

A**ACT**GG

but the LCS length is again 6

For more information, please read the site


Longest Common Subsequence

([https://en.wikipedia.org/wiki/Longest_common_subsequence_problem#targetText=The%20longest%20common%20subsequence%20\(LCS,\(often%20just%20two%20sequences\)\).](https://en.wikipedia.org/wiki/Longest_common_subsequence_problem#targetText=The%20longest%20common%20subsequence%20(LCS,(often%20just%20two%20sequences)).)) (Wikipedia)

Also, see how to solve the problem using this visualization site. <https://www.cs.usfca.edu/~galles/visualization/DPLCS.html> (<https://www.cs.usfca.edu/~galles/visualization/DPLCS.html>)

Statement of Work

Part 1

Write a program that reads two strings from an [input](#)  file (The first line is X, the second line is Y), compute the longest common subsequence length AND the resulting string.

You will need to write 2 methods

1) return LCS length in iterative function

// return the length of LCS. **C is the 2D matrix**, X, Y are the input strings, m=|X|, n=|Y|

```
int lcs_it(int **C, string X, string Y, int m, int n )
```

2) return LCS resulting string in recursive function

// return the resulting LCS string.

```
string backtrack(int **C, string X, string Y, int m, int n)
```

Hint:

Compute the length of LCS requires to use 2D matrix where the row is |X|+1, column is |Y|+1. And the 2D matrix result should be used to construct the resulting string. Below is how to create a 2D array and how to pass it to the lcs_it function.

The methods will be tested in the main method as the following.

```
===== main =====
```

```
int m=X.size();

int n=Y.size();

int **C=new int*[m+1];
for (int i=0; i<m+1;++i) C[i]=new int [n+1];

int len=lcs_it(C, X, Y, m, n);
```

```
cout<<" LCS is "<<backtrack(C, X, Y, m, n)<<endl;
```

```
=====
```

Part 2

You will write a program that compares iterative LCS and recursive LCS function.

1. Write the following two methods.

1) return LCS length in iterative function

```
int lcs_it_test(string X, string Y, int m, int n )
```

Unlike the function from part 1, you will not reuse the 2D matrix. So, you should not create 2D array in the main. You will use local 2D array in the function.

2) return LCS length in recursive function

```
int lcs_re(string X, string Y, int m, int n)
```

You will implement a method that compute LCS length using recursive method.

Here, you will not call the backtrack to get the LCS. You just need to get the length of LCS.

2. your main function should test both versions of programs. ask the user to input a trial number for each program.

```
Enter trial time for iterative version (less than min(|X|, |Y|))
```

```
Enter trial time for recursive version (less than 20)
```

3. If you implemented correctly, iterative version is fast, but recursive version is very slow. **Therefore, you should not increase the size more than 20 for recursive version.**

For each iteration, you have to increase the string (X, Y) size and compute the LCS length. You should log the execution time as increasing the string size.

Hint:

```
#include <ctime>

int start_s, stop_s;

start_s=clock();

// Execute program

stop_s=clock();

clog << i << "\t" << (stop_s-start_s)/double(CLOCKS_PER_SEC)*1000 << endl;
```

4. Plot the time as increasing the string size from **1 to 800** for iterative version. Estimate the big-O of your program.

5. Plot the time as increasing the string size from **1 to 20** for recursive version. Estimate the big-O of your program.

Hint: when you run the program, if you run with

```
./a.out 2>time.log
```

Then only the size and time is printed to time.log file. You can open it with excel file to plot the graph.

What to Turn in


Clearly state in your code comments any other assumptions you have made. Turn in:

(1) driver.cpp

your driver.cpp should include both Part1 and Part2. That is,

- Return LCS length iteratively with the 2D matrix is passed as a parameter
- Result the LCS string recursively
- Return LCS length iteratively without 2D matrix as a parameter
- Return LCS length recursively
- your main function should perform Part1 and Part2

(2) a separate report in .doc or .docx that must includes:

- (a) The first page should be an output of your Part1. Print the X, and Y, and LCS length and string. (X and Y are provided in the [input](#)  file)
- (b) Plots that show the performance of iterative LCS and recursive LCS. You should have a separate plot for each method.
- (c) Estimated Big O for each algorithm.

Grading Guide and Answers

Check the following grading guide to see how your homework will be graded.

Program 2 Grade Guideline

1. Documentation (10 pts)

Output (The first page)

Correct(2pts) 1 ~ 2 errors(1pt) 3+ errors or no results(0pt)

Performance comparison between two algorithms

All plots are given and Big-O is estimated using 2^n , n^3 , n^2 , $n \log n$, n , and $\log n$ plots. (6 base cases)

Both programs are estimated and comparison results are clearly stated.(8 pts)

No plots: 0 pts

Each of the following cases be taken off 2 points :

- No Big-O estimation or incorrect estimation
- Less than 6 base cases are tested
- No statement for comparison results

2. Correctness (18 pts)

- Compilation errors(0pt)

- Successful Compilation

- + Correct LCS length computing method(iterative with 2D array parameter)(2 pts)
- + Correct LCS string recursively computing method(2 pts)
- + Correct LCS length computing method (iterative without 2D array parameter) (2 pts)
- + Correct LCS length recursively computing method(2 pts)
- + Your program should read two strings of X and Y from the given input file (2 pts)
- + Your driver file test the correctness of each program (2 pts)
- + Your program should ask inputs from the user (2 pts)
- + A new loop is added so that the size can be increased from 1 to 800 to test iterative LCS length (2 pts)
- + A new loop is added so that the size can be increased from 1 to 20 to test recursive LCS length (2 pts)

3. Program Organization (2pts)

You must write a plenty of comments to help the professor or the grader understand your code.

Proper comments

Good (1pts)

Poor/No comments(0pts)

[Coding style](#) (proper indentations, blank lines, variable names, and non-redundant code)

Good (1pts)

Poor(0pts)