Longest common Substring Algorithm

Abstract/interpretation :
Given two sequences, the traditional longest common subsequence (LCS) problem is to obtain the common subsequence with the maximum number of matches, without considering the continuity of the matched characters. However, in many applications, the matching results with higher continuity are more meaningful than the sparse ones, even if the number of matched characters is a little lower.
I will show that the optimal solution of the LCS can be determined in O(n 2) time, where n denotes the longer length of the two input sequences.
The topic of evaluating how similar one string or one sequence to another is, such as the longest common subsequence [3, 4, 7,10, 15, 17] or the edit distance [2, 14], has been studied for several decades.
Given two sequences X and Y , the longest common subsequence (LCS) the problem asks for the longest sequence that is a subsequence of both X and Y . Meanwhile, the edit distance problem is to find the minimum number of operations (to insert, delete or substitute one character each time) required for modifying one array to another. However, to my knowledge, the continuity of matched integers has not yet been emphasized in previous methods.

What is the algorithm used for?
For the purpose of this report I have made some research on why we need such an algorithm. The reason that captivated my attention were:

- Plagiary detection,

For plagiary detection, long and continuous matches infer that many sentences were copied from one article to the other. Therefore, it is interesting and necessary to devise a comparison scheme that considers the continuity of matches. For this purpose, many tools online for this purpose like Turnitin is implemented using the flexible longest common subsequence (FLCS) problem.

- DNA comparison:

In the textbook it can be noted that this algorithm can be used for biological applications to compare the DNA of two (or more) different organisms.

- And many other interesting applications.

Time Complexity:

For the general case of an arbitrary number of input sequences, when the number of sequences is constant, the problem is solvable in polynomial time by dynamic programming.

The running time of the algorithm is given to be in normal time O(2^n)

For the case of two sequences of n and m elements, the running time of the dynamic programming approach becomes O(n × m). For an arbitrary number of input sequences, the dynamic programming approach gives a solution in

The LCS, in the worst case, the number of common subsequences is exponential in the lengths of the inputs, so the algorithmic complexity must be at least exponential.

The LCS problem has an optimal substructure: the problem can be broken down into smaller, simpler subproblems, which can, in turn, be broken down into simpler subproblems, and so on, until, finally, the solution becomes trivial. LCS in particular has overlapping subproblems: the solutions to high-level subproblems often reuse solutions to lower level subproblems.

Like in the previous assignment about Rod cutting, these two properties are amenable to dynamic programming approaches, in which subproblem solutions are memoized, that is, the solutions of subproblems are saved for reuse.

Code explanation:

LCS(X, Y,n,m) is a function that computes a longest subsequence common to X and Y.

While the Print(X,Y,Z,n,m) is another function that prints the results,

The Driver program ask to input the two values for the arrays X and Y respectively, and proceed to randomly fill those with numbers between 0~10, first calls the LCS function and then lastly print the result.

Results:

Comparison of different size arrays:

```
n: 8

m: 8

X=[ 3, 9, 0, 5, 2, 2, 7, 3]
Y=[ 7, 9, 3, 8, 0, 2, 4, 8]

Z=[ 9, 0, 2]

Time for lcs and print: 64 microseconds
```

```
n: 9

m: 9

X=[ 9, 0, 5, 2, 2, 7, 3, 7, 9]
Y=[ 7, 9, 3, 8, 0, 2, 4, 8, 3]

Z=[ 9, 0, 2, 3]

Time for lcs and print: 66 microseconds
```