Deakin University

Data Structures and Algorithms

OnTrack Submission

Helping your peers

Submitted By:
Negin Pakroohjahromi
s222393187
2024/05/26 07:19

Tutor:
Ziwei Hou

Outcome	Weight
Complexity	♦♦◊◊◊
Implement Solutions	♦♦♦♦♦
Document solutions	♦♦♦♦♦

I created a Distinction practical task as discussed for this task.

May 26, 2024

Practical Task …

(Distinction Task)

Submission deadline: …
Discussion deadline: …

Task Objective

This task aims to practice working with dynamic programming concepts and to implement a specific algorithm to solve the Longest Common Subsequence (LCS) problem. This task has two stages: the first is to implement the LCS algorithm using dynamic programming, and the second is to visualize the LCS matrix.

Background: Longest Common Subsequence (LCS)

The Longest Common Subsequence (LCS) problem is a classic problem in computer science with widespread applications in various fields such as bioinformatics, text comparison, and data analysis. The problem involves finding the longest subsequence common to two given sequences. A subsequence is derived by deleting some or none of the elements from the original sequence without changing the order of the remaining elements. For example, in sequences "AGGTAB" and "GXTXAYB", the LCS is "GTAB". The LCS problem is crucial in areas where sequence comparison is needed, such as DNA sequence analysis in bioinformatics, diff tools in text comparison, and version control systems in software development.

Task Details

Stage 1: Implementing the LCS Algorithm

1. Explore the Provided Program Code:
   Create a new Microsoft Visual Studio project and import the provided LCS.cs file. Alternatively, you can extend an existing project by copying the necessary code from the provided LCS template. This step involves setting up your development environment to work on the LCS algorithm implementation.

2. LCS Functionality:
   Implement the LCS algorithm using dynamic programming. The dynamic programming approach ensures that we solve the problem efficiently by breaking it down into simpler subproblems and storing the results of these subproblems to avoid redundant computations.

   The algorithm should construct a 2D matrix ‘L’ where ‘L[i][j]’ contains the length of the LCS of the sequences ‘X[0...i-1]’ and ‘Y[0...j-1]’. The matrix ‘L’ helps in keeping track of the lengths of the longest common subsequences at each step.

3. ComputeLCSMatrix Method:
   Use the following signature for your function:
   public static int[,] ComputeLCSMatrix (string X, string Y)

   This method should initialize a 2D matrix of size (m+1) x (n+1), where m is the length of sequence X and n is the length of sequence Y. The extra row and column are used to handle the base case where one of the sequences is empty.

   Fill in the matrix according to the following rules:
   If either i or j is 0, then L[i][j] = 0.
   If X[i-1] == Y[j-1], then L[i][j] = L[i-1][j-1] + 1.
   Otherwise, L[i][j] = max(L[i-1][j], L[i][j-1]).

4. GetLCS Method:
   Implement a method to backtrack through the LCS matrix to extract the actual LCS string.

   Use the following signature:
   public static string GetLCS(string X, string Y, int[,] L)

   This method should start from the bottom-right corner of the matrix and trace back the path to construct the LCS string. If X[i-1] == Y[j-1], the character is part of the LCS, and we move diagonally up-left. Otherwise, we move in the direction of the larger value (up or left).

Stage 2: Visualizing the LCS Matrix

5. Implement a method to print the LCS matrix in a readable format. This visualization helps in debugging and understanding how the dynamic programming process works. By seeing the matrix, you can trace the steps the algorithm takes to compute the LCS and identify any potential issues in the computation.

6. PrintLCSMatrix Method:
   Use the following signature for your method:
   public static void PrintLCSMatrix(int[,] L, string X, string Y)

   This method should print the matrix with appropriate labels for the sequences along the top and left edges. Each cell in the matrix should show the value stored at that position, which represents the length of the LCS up to that point.

Testing and Debugging

You will be provided with a Tester.cs and an ISorter.cs template to test your LCS implementation. As you progress with the implementation of this task, you should start using the Tester class to test your code for potential logical issues and runtime errors. This (testing) part of the task is as important as coding. You may wish to extend it with extra test cases to be sure that your solution is checked against other potential mistakes. Before you proceed with searching, make sure that your data is sorted according to the dedicated comparer.

As a Distinction task, it is important to ensure your submission is professional. Ensure you remove debug code, correctly format, use formal naming convention consistently and comment your code. Submissions will not be accepted that are not of a professional quality.

Expected Printout

Appendix A provides an example printout for your program. If you are getting a different printout then your implementation is not ready for submission. Please ensure your program prints out Success for all tests before submission.

Further Notes

- Dynamic Programming:
  Refer to Chapter 15 of the course book “Introduction to Algorithms” by Cormen, Leiserson, Rivest, and Stein. This chapter covers dynamic programming and provides pseudocode for the LCS problem.
  Make sure to understand the dynamic programming approach, as it forms the basis for solving many optimization problems.

Please note:
The solution codes, including LCS.cs, Tester.cs, and ISorter.cs, will be attached at the beginning of the portfolio and this task's chatbot.

Appendix A

This section displays the printout produced by the attached Tester class, specifically by its *Main* method. It is based on our solution. The printout is provided here to help with testing your code for potential logical errors. It demonstrates the correct logic rather than an expected printout in terms of text and alignment.

```
Running LCS Tests:

LCS matrix for X: AGGTAB, Y: GXTXAYB
    G X T X A Y B
  0 0 0 0 0 0 0 0
A 0 0 0 0 0 1 1 1
G 0 1 1 1 1 1 1 1
G 0 1 1 1 1 1 1 1
T 0 1 1 2 2 2 2 2
A 0 1 1 2 2 3 3 3
B 0 1 1 2 2 3 3 4
LCS of AGGTAB and GXTXAYB is GTAB
Expected LCS: GTAB
Test Passed

LCS matrix for X: ABCBDAB, Y: BDCAB
    B D C A B
  0 0 0 0 0 0
A 0 0 0 0 1 1
B 0 1 1 1 1 2
C 0 1 1 2 2 2
B 0 1 1 2 2 3
D 0 1 2 2 2 3
A 0 1 2 2 3 3
B 0 1 2 2 3 4
LCS of ABCBDAB and BDCAB is BCAB
Expected LCS: BCAB
Test Passed

LCS matrix for X: XMJYAUZ, Y: MZJAWXU
    M Z J A W X U
  0 0 0 0 0 0 0 0
X 0 0 0 0 0 0 1 1
M 0 1 1 1 1 1 1 1
J 0 1 1 2 2 2 2 2
Y 0 1 1 2 2 2 2 2
A 0 1 1 2 3 3 3 3
U 0 1 1 2 3 3 3 4
Z 0 1 2 2 3 3 3 4
LCS of XMJYAUZ and MZJAWXU is MJAU
Expected LCS: MJAU
Test Passed

LCS matrix for X: , Y: ABC
    A B C
  0 0 0 0
LCS of  and ABC is
Expected LCS:
Test Passed
```

```
LCS matrix for X: ABC, Y:

  0
A 0
B 0
C 0
LCS of ABC and  is
Expected LCS:
Test Passed

LCS matrix for X: ABC, Y: DEF
    D E F
  0 0 0 0
A 0 0 0 0
B 0 0 0 0
C 0 0 0 0
LCS of ABC and DEF is
Expected LCS:
Test Passed


Running Sorting Tests:
Sorted array: 1, 2, 3, 4, 5, 6, 7, 8
Test Passed

Sorted array: 8, 7, 6, 5, 4, 3, 2, 1
Test Passed

Sorted array: 8, 6, 2, 4, 5, 3, 7, 1
Test Passed

Test Summary:

LCS Tests Passed: 6
LCS Tests Failed: 0

Sorting Tests Passed: 3
Sorting Tests Failed: 0

All tests completed.
```