Timothy Allec

Name, SID, Date ....

In Class Assignment 6: Longest Common Subsequence

Benjamin Sanders, MS February 22, 2021

1 Introduction

You will need to work individually to complete this assignment. Write your name at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

import java.util.Vector;

....
....
....
....

2 Assignment Description

2.1 Big Picture

This Dynamic Programming algorithm is used to find similarities between sequences of DNA strands, denoted as strings X and Y, below.

2.2 Algorithm Implementation

Implement the following algorithms in Java, using the Vector data structure for any 1-D array, 2-D array (a.k.a. table), or linear algebra purposes.

```
LCS-LENGTH($X, Y$)
1  $m = X.length$
2  $n = Y.length$
3  let $b[1..m, 1..n]$ and $c[0..m, 0..n]$ be new tables
4  for $i = 1$ to $m$
5      $c[i, 0] = 0$
6  for $j = 0$ to $n$
7      $c[0, j] = 0$
8  for $i = 1$ to $m$
9      for $j = 1$ to $n$
10         if $x_i == y_j$
11             $c[i, j] = c[i-1, j-1] + 1$
12             $b[i, j] =$ "$\nwarrow$"
13         elseif $c[i-1, j] \geq c[i, j-1]$
14             $c[i, j] = c[i-1, j]$
15             $b[i, j] =$ "$\uparrow$"
16         else $c[i, j] = c[i, j-1]$
17             $b[i, j] =$ "$\leftarrow$"
18 return $c$ and $b$
```

```
PRINT-LCS($b, X, i, j$)
if $i == 0$ or $j == 0$
    return
if $b[i, j] ==$ "↖"
    PRINT-LCS($b, X, i-1, j-1$)
    print $x_i$
elseif $b[i, j] ==$ "↑"
    PRINT-LCS($b, X, i-1, j$)
else PRINT-LCS($b, X, i, j-1$)
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	←1	↖ 1
2	B	0	↖ 1	←1	←1	↑ 1	↖ 2	←2
3	C	0	↑ 1	↑ 1	↖ 2	←2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	←3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Figure 15.8 The c and b tables computed by LCS-LENGTH on the sequences $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$. The square in row i and column j contains the value of $c[i, j]$ and the appropriate arrow for the value of $b[i, j]$. The entry 4 in $c[7, 6]$—the lower right-hand corner of the table—is the length of an LCS $\langle B, C, B, A \rangle$ of X and Y. For $i, j > 0$, entry $c[i, j]$ depends only on whether $x_i = y_j$ and the values in entries $c[i-1, j]$, $c[i, j-1]$, and $c[i-1, j-1]$, which are computed before $c[i, j]$. To reconstruct the elements of an LCS, follow the $b[i, j]$ arrows from the lower right-hand corner; the sequence is shaded. Each "↖" on the shaded sequence corresponds to an entry (highlighted) for which $x_i = y_j$ is a member of an LCS.

2.3 Time Complexity Analysis

Where m is the number of characters in X and n is the number of characters in Y, analyze the time complexity of the given algorithms with respect to m and n. Write the result of your analysis in big-O notation, i.e. $O(mn)$ in the space below.

O(m * n)

2.4 Space Complexity Analysis

Where m is the number of characters in X and n is the number of characters in Y, analyze the space complexity of the given algorithms with respect to m and n. Write the result of your analysis in big-O notation, i.e. $O(mn)$ in the space below.

O(m * n)

Name, SID, Date ....

2.5 Optimize the Algorithm for a Purpose

Choose an optimization, either in time or in space, for the first given algorithm. Write your intended big-O notation, i.e. $O(1)$, $O(n)$, etc., in the space below, and write N/A in the other space.

- Time Complexity: O(m * n) ....
- Space Complexity: ....

What application would benefit from the purpose of the above optimization? Why? Write two sentences to answer these questions in the space below.

In version control systems like Git, the LCS algorithm is used to identify differences between two versions of a file. The optimization uses a table cTable to store the lengths of common subsequences between the input strings xInput and yInput. The print_LCS method then backtracks through the table to find and print one of the possible longest common subsequences.

2.6 New Algorithm Design and Implementation

In the space below, design an algorithm that achieves the same purpose of the first given algorithm, but includes the optimization you have specified above. Use pseudocode written in a style similar to the given algorithm, and implement it in Java. You may use as many additional pages as necessary for this purpose.

```
import java.util.Vector;

public class optimization {

   public static void print_LCS(Vector<Vector<Integer>> cInput,String xInput, String yInput, int i, int j ){
      if(i==0 || j == 0){
         System.out.println();
         return;
      }

      String direction ="";

      int cCurrent = cInput.get(i).get(j);
      int dValue = cInput.get(i-1).get(j-1);
      int uValue = cInput.get(i-1).get(j);
      int lValue = cInput.get(i).get(j-1);

      Character myXinputChar = xInput.charAt(i-1);
      Character myYinputChar = yInput.charAt(j-1);

      // default: the element to the left
      direction = "l";

      if(myXinputChar.equals(myYinputChar)){
            direction = "d";
         }
      else if(uValue >= lValue){
         direction = "u";
      }
         System.out.println("The direction is: " + direction);
         if(direction.equals("d")){
            print_LCS(cInput, xInput, yInput, i-1, j-1);
            System.out.print(myXinputChar);
         }
         else if(direction.equals("u")){
            print_LCS(cInput, xInput, yInput, i-1, j);
         }
         else if(direction.equals("l")){
            print_LCS(cInput, xInput, yInput, i, j-1);
         }
      }

   public static void main(String[] args) {

         String xInput = "ABCBDAB";
         String yInput = "BDCABA";

         Vector<Vector<Integer>> cTable = new
   Vector<Vector<Integer>>();

         // initiliaze the tables
         for(int i = 0; i <= xInput.length(); i++){
            Vector<Integer> cRow = new Vector<Integer>();
            for(int j = 0; j <= yInput.length(); j++){
               cRow.add(0);
            }
            cTable.add(cRow);
         }

         // consider only C table
         for(int i = 1; i <= xInput.length(); i++){
            for(int j = 1; j <= yInput.length(); j++){
               Character myXinputChar = xInput.charAt(i-1);
               Character myYinputChar = yInput.charAt(j-1);
               if(myXinputChar.equals(myYinputChar)){
                  cTable.get(i).set(j, cTable.get(i-1).get(j-1) +
   1);
               }
               else if(cTable.get(i-1).get(j) >= cTable.get(i).get
   (j-1)){
                  cTable.get(i).set(j, cTable.get(i-1).get(j));
               }
               else{
                  cTable.get(i).set(j, cTable.get(i).get(j-1));
               }
            }
         }

         print_LCS(cTable, xInput, yInput, 0, 0);
      }
}
```

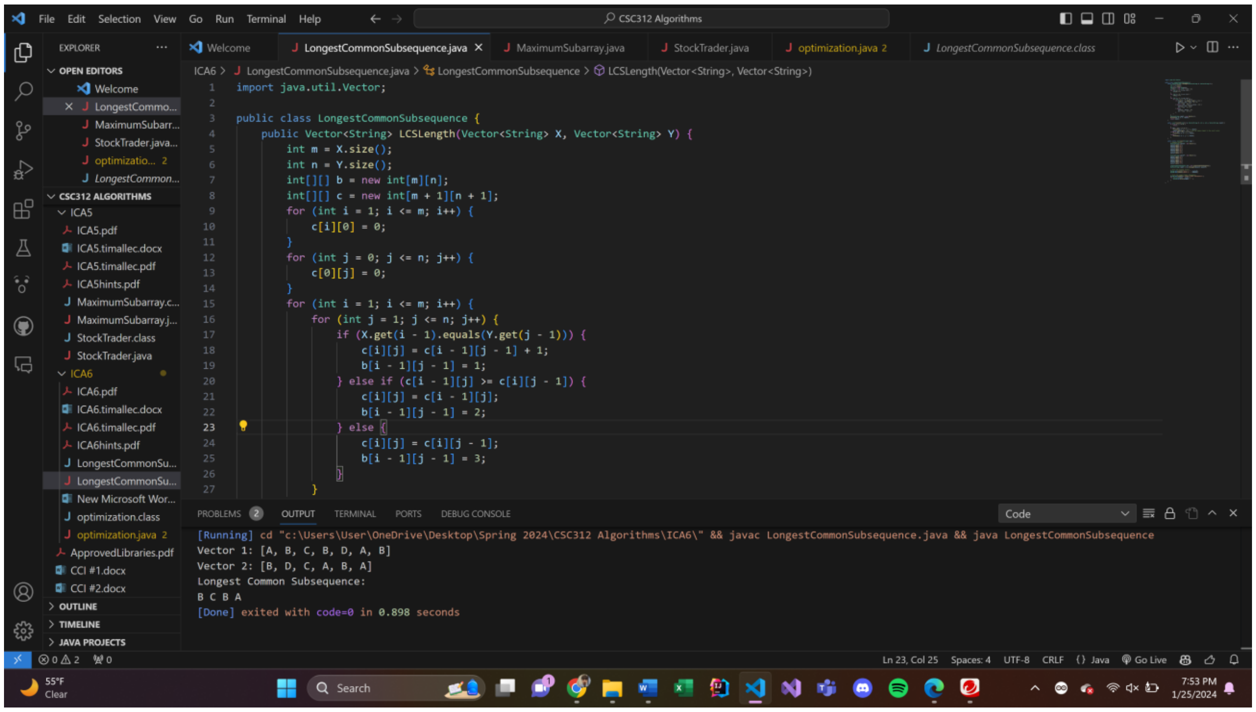
3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.

```java
import java.util.Vector;

public class LongestCommonSubsequence {
    public Vector<String> LCSLength(Vector<String> X, Vector<String> Y) {
        int m = X.size();
        int n = Y.size();
        int[][] b = new int[m][n];
        int[][] c = new int[m + 1][n + 1];
        for (int i = 1; i <= m; i++) {
            c[i][0] = 0;
        }
        for (int j = 0; j <= n; j++) {
            c[0][j] = 0;
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (X.get(i - 1).equals(Y.get(j - 1))) {
                    c[i][j] = c[i - 1][j - 1] + 1;
                    b[i - 1][j - 1] = 1;
                } else if (c[i - 1][j] >= c[i][j - 1]) {
                    c[i][j] = c[i - 1][j];
                    b[i - 1][j - 1] = 2;
                } else {
                    c[i][j] = c[i][j - 1];
                    b[i - 1][j - 1] = 3;
                }
            }
        }
        Vector<String> result = new Vector<>();
        PrintLCS(b, X, m, n, result);
        return result;
    }

    public void PrintLCS(int[][] b, Vector<String> X, int i, int j,
Vector<String> result) {
        if (i == 0 || j == 0) {
            return;
        }
        if (b[i - 1][j - 1] == 1) {
            PrintLCS(b, X, i - 1, j - 1, result);
            result.add(X.get(i - 1)); // Add the common element to the result
vector
        } else if (b[i - 1][j - 1] == 2) {
            PrintLCS(b, X, i - 1, j, result);
        } else {
            PrintLCS(b, X, i, j - 1, result);
        }
    }

    public static void main(String[] args) {
        // Example vectors
        Vector<String> vectorX = new Vector<>();
        vectorX.add("A");
        vectorX.add("B");
        vectorX.add("C");
        vectorX.add("B");
        vectorX.add("D");
        vectorX.add("A");
        vectorX.add("B");

        Vector<String> vectorY = new Vector<>();
        vectorY.add("B");
        vectorY.add("D");
        vectorY.add("C");
        vectorY.add("A");
        vectorY.add("B");
        vectorY.add("A");

        LongestCommonSubsequence lcs = new LongestCommonSubsequence();
        Vector<String> result = lcs.LCSLength(vectorX, vectorY);

        // Print the original vectors
        System.out.println("Vector 1: " + vectorX);
        System.out.println("Vector 2: " + vectorY);

        // Print the Longest Common Subsequence
        System.out.println("Longest Common Subsequence:");
        for (String element : result) {
            System.out.print(element + " ");
        }
    }
}
```

CODE OUTPUT:

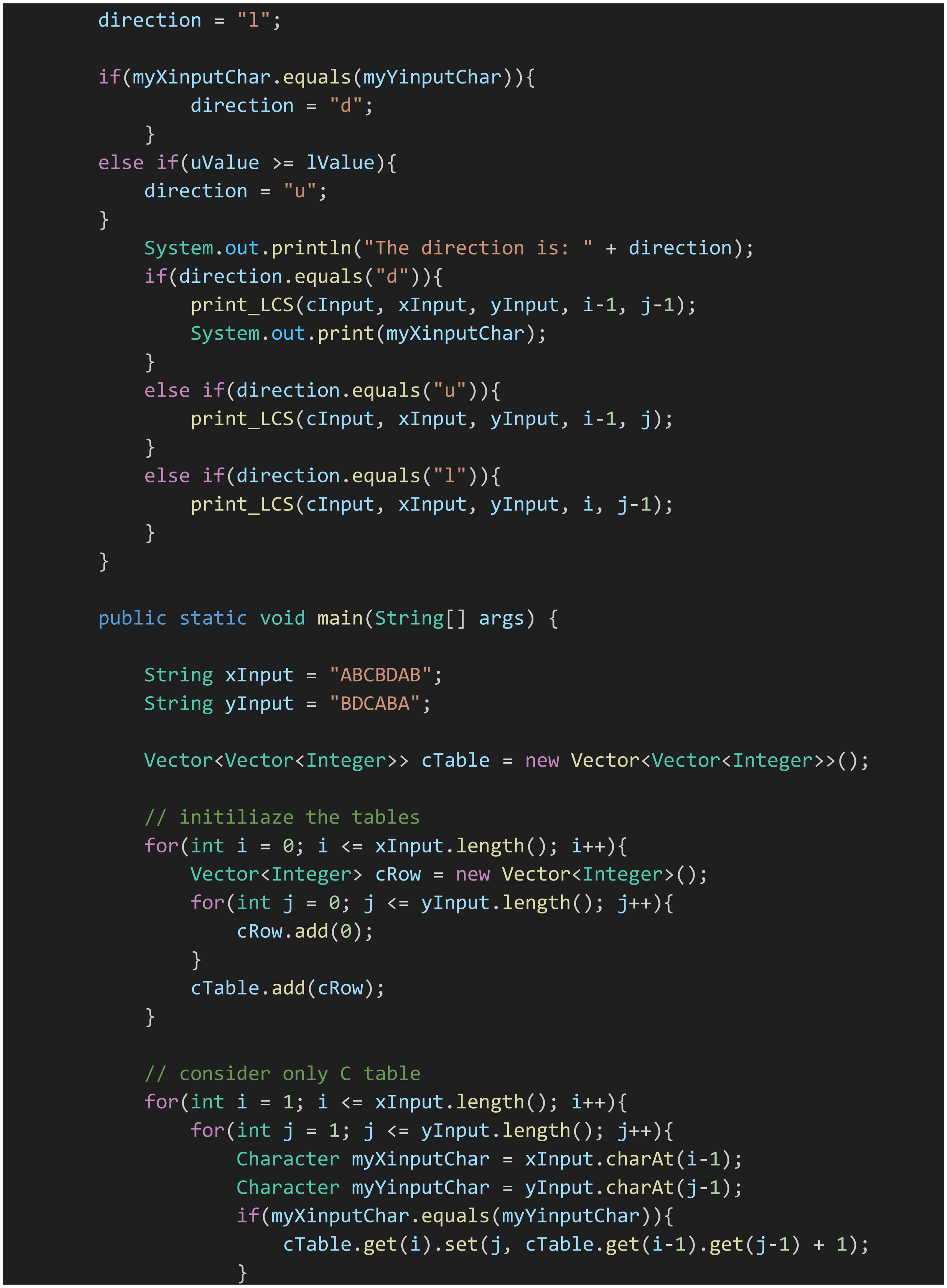


```
import java.util.Vector;

public class LongestCommonSubsequence {
    public Vector<String> LCSLength(Vector<String> X, Vector<String> Y) {
        int m = X.size();
        int n = Y.size();
        int[][] b = new int[m][n];
        int[][] c = new int[m + 1][n + 1];
        for (int i = 1; i <= m; i++) {
            c[i][0] = 0;
        }
        for (int j = 0; j <= n; j++) {
            c[0][j] = 0;
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (X.get(i - 1).equals(Y.get(j - 1))) {
                    c[i][j] = c[i - 1][j - 1] + 1;
                    b[i - 1][j - 1] = 1;
                } else if (c[i - 1][j] >= c[i][j - 1]) {
                    c[i][j] = c[i - 1][j];
                    b[i - 1][j - 1] = 2;
                } else {
                    c[i][j] = c[i][j - 1];
                    b[i - 1][j - 1] = 3;
                }
            }
```

```
[Running] cd "c:\Users\User\OneDrive\Desktop\Spring 2024\CSC312 Algorithms\ICA6\" && javac LongestCommonSubsequence.java && java LongestCommonSubsequence
Vector 1: [A, B, C, B, D, A, B]
Vector 2: [B, D, C, A, B, A]
Longest Common Subsequence:
B C B A
[Done] exited with code=0 in 0.898 seconds
```

OPTIMIZATION:

```
import java.util.Vector;

public class optimization {

    public static void print_LCS(Vector<Vector<Integer>> cInput,String xInput,
String yInput, int i, int j ){
        if(i==0 || j == 0){
            System.out.println();
            return;
        }

        String direction ="";

        int cCurrent = cInput.get(i).get(j);
        int dValue = cInput.get(i-1).get(j-1);
        int uValue = cInput.get(i-1).get(j);
        int lValue = cInput.get(i).get(j-1);

        Character myXinputChar = xInput.charAt(i-1);
        Character myYinputChar = yInput.charAt(j-1);

        // default: the element to the left
```



```
    direction = "l";

    if(myXinputChar.equals(myYinputChar)){
            direction = "d";
        }
    else if(uValue >= lValue){
        direction = "u";
    }
        System.out.println("The direction is: " + direction);
        if(direction.equals("d")){
            print_LCS(cInput, xInput, yInput, i-1, j-1);
            System.out.print(myXinputChar);
        }
        else if(direction.equals("u")){
            print_LCS(cInput, xInput, yInput, i-1, j);
        }
        else if(direction.equals("l")){
            print_LCS(cInput, xInput, yInput, i, j-1);
        }
    }

    public static void main(String[] args) {

        String xInput = "ABCBDAB";
        String yInput = "BDCABA";

        Vector<Vector<Integer>> cTable = new Vector<Vector<Integer>>();

        // initiliaze the tables
        for(int i = 0; i <= xInput.length(); i++){
            Vector<Integer> cRow = new Vector<Integer>();
            for(int j = 0; j <= yInput.length(); j++){
                cRow.add(0);
            }
            cTable.add(cRow);
        }

        // consider only C table
        for(int i = 1; i <= xInput.length(); i++){
            for(int j = 1; j <= yInput.length(); j++){
                Character myXinputChar = xInput.charAt(i-1);
                Character myYinputChar = yInput.charAt(j-1);
                if(myXinputChar.equals(myYinputChar)){
                    cTable.get(i).set(j, cTable.get(i-1).get(j-1) + 1);
                }
```

```
                    else if(cTable.get(i-1).get(j) >= cTable.get(i).get(j-1)){
                        cTable.get(i).set(j, cTable.get(i-1).get(j));
                    }
                    else{
                        cTable.get(i).set(j, cTable.get(i).get(j-1));
                    }
                }
            }

            print_LCS(cTable, xInput, yInput, 0, 0);
        }




    }
```

ALL CODE WORKS AS INTENDED !