LONGEST COMMON SUBSEQUENCE

A MINI PROJECT REPORT

Submitted by

SOURABH SHARMA [RA2111042010007]

D KAMALESH REDDY [RA2111042010019]

D BHEEMESWARA CHOWDARI [RA2111042010020]

for the course 18CSC361J – Design and Analysis of Algorithms

Under the guidance of

Dr. P. C. Karthik Sir

(Associate Professor, Department of Data Science and Business Systems)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING AND BUSINESS SYSTEMS

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

OCTOBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur – 603203.

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

BONAFIDE CERTIFICATE

Reg. No:

R	A	2	1	1	1	0	4	2	0	1	0	0	7

Certified that this is the bonafide record of work done by **SOURABH SHARMA** of V semester B.Tech. COMPUTER SCIENCE WITH BUSINESS SYSTEMS during the academic year 2023-2024 in the 18CSC361J – Design and Analysis of Algorithms Laboratory.

---------- ----------

Dr. P. C. Karthik **Dr. M. Lakshmi**

Faculty - Incharge **HOD/DSBS**

Submitted for the practical examination held on_____ at SRM Institute of Science and Technology, Kattankulathur, Chennai-603203.

---------- ----------

Examiner-1 Examiner-2

ACKNOWLEDGEMENT

We express our humble gratitude to Dr. C. Muthamizhchelvan, Vice Chancellor (I/C), SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Dr. Revathi Venkataraman, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support.

We wish to thank Dr. M. Lakshmi, Professor & Head, Department of Computer Science and Business Systems, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor Dr E. Sasikala, Assistant Professor, Department of Computer Science and Business Systems, SRM Institute of Science and Technology, for her great support at all the stages of project work. We register our immeasurable thanks to our Faculty Advisors, Dr. Jeba Sonia, and Dr. Mercy Thomas, Assistant Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, Dr. P. C. Karthik, Assistant Professor, Department of Computer Science and Business Systems, SRM Institute of Science and Technology, for providing us an opportunity to pursue our project under his mentorship. He provided the freedom and support to explore the research topics of our interest. His passion for solving real problems and making a difference in the world has always been inspiring.

We sincerely thank staff and students of the Computer Science and Business Systems Department, SRM Institute of Science and Technology, for their help during my research.

Finally, we would like to thank my parents, our family members and our friends for their unconditional love, constant support, and encouragement.

CONTENT

S. No.	Index	Page No.
1.	Abstract	5
2.	Topic: Longest Common Subsequence	6
3.	Dynamic Programming for LCS I. Algorithm II. Illustration III. Code IV. Output	7-11
4.	Usage for LCS Program	12
5.	Recursion for LCS I. Algorithm II. Illustration III. Code IV. Recurrence Relation	13-16
6.	Real Life Application	17
7.	References	18

1. Abstract

The Longest Common Subsequence (LCS) problem is a fundamental challenge in computer science and plays a crucial role in various applications, including bioinformatics, text comparison, and data analysis. Given two sequences, the LCS is defined as the longest sequence of elements that appears in the same order in both input sequences. This abstract provides an overview of the LCS problem, its significance, and various algorithms used to solve it.

The LCS problem has numerous practical applications, such as DNA sequence alignment, plagiarism detection, version control systems, and natural language processing. Solving this problem efficiently is critical for improving the performance of these applications. This abstract discusses some of the common algorithms used to find the LCS, including dynamic programming, memorization, and more recent advances like the Hirschberg algorithm.

This abstract serves as a concise introduction to the Longest Common Subsequence problem, illustrating its broad scope and significance in various fields of study and application. Researchers and practitioners interested in algorithm design, data analysis, and information retrieval will find the LCS problem to be a compelling area of study and innovation.

2. Topic: LONGEST COMMON SUBSEQUENCE (LCS)

The Longest Common Subsequence (LCS) problem is a classic Computational challenge in the field of computer science and data analysis. The problem can be defined as follows:

Given two input sequences, typically represented as strings or arrays, the goal is to find the longest subsequence that appears in the same order in both input sequences. A subsequence is a sequence that can be derived from another sequence by deleting zero or more elements without changing the order of the remaining elements. The LCS does not require elements to appear consecutively in the original sequences.

The LCS problem continues to be an active area of research, with ongoing efforts to develop faster and more memory-efficient algorithms to address its computational demands.

3. Dynamic Programming for LCS:

Algorithm:

We can use the following steps to implement the dynamic programming approach for LCS.

- Create a 2D array **dp[][]** with rows and columns equal to the length of each input string plus 1 [the number of rows indicates the indicesof **S1** and the columns indicate the indices of **S2**].
- Initialize the first row and column of the dp array to 0.
- Iterate through the rows of the dp array, starting from 1 (say using iterator **i**).
- For each **i**, iterate all the columns from **j = 1 to n**:
- If **S1[i-1]** is equal to **S2[j-1]**, set the current element of the dp arrayto the value of the element to (**dp[i-1] [j-1] + 1**).
- Else, set the current element of the dp array to the maximum valueof **dp[i-1] [j]** and **dp[i][j-1]**.
- After the nested loops, the last element of the dp array will containthe length of the LCS.

See the below illustration for a better understanding:

Illustration:

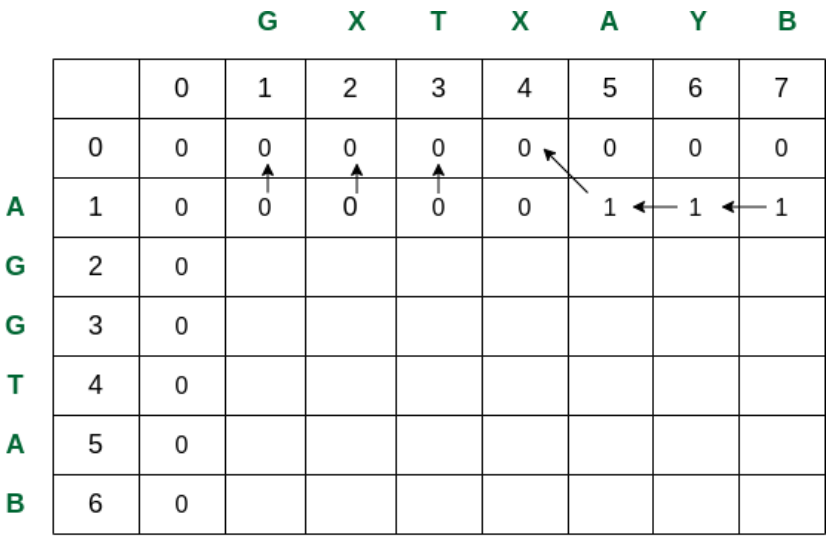
*Say the strings are **S1 = "AGGTAB"** and **S2 = "GXTXAYB"**.*

***First step:** Initially create a 2D matrix (say dp[][]) of size 8 x 7 whose first row and first column are filled with 0.*

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0							
G	2	0							
G	3	0							
T	4	0							
A	5	0							
B	6	0							

Creating dp table and filling 0 in 0th row and column

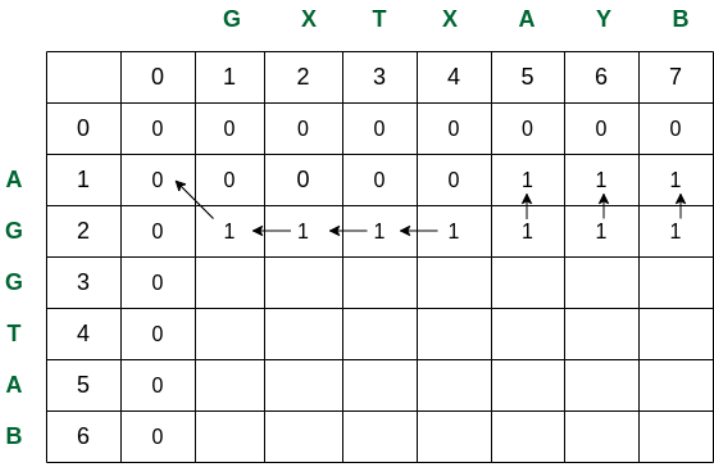
Second step: *Traverse for i = 1. When j becomes 5, S1[0] and S2[4] are equal. So the dp[][] is updated. For the other elements take the maximum ofdp[i-1][j] and dp[i][j-1]. (In this case, if both values are equal, we have used arrows to the previous rows).*



Updating dp table for row 1

Filling the row no 1

Third step: *While traversed for i = 2, S1[1] and S2[0] are the same (both are 'G'). So the dp value in that cell is updated. Rest of the elements are updated as per the conditions.*



Updating dp table for row 2

Filling the row no. 2

Fourth step: *For i = 3, S1[2] and S2[0] are again same. The updations are as follows.*

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0							
A	5	0							
B	6	0							

Updating dp table for row 3

Filling row no. 3

Fifth step: *For i = 4, we can see that S1[3] and S2[2] are same. So dp[4][3] updated as dp[3][2] + 1 = 2.*

			G	X	T	X	A	Y	B
		0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0							
B	6	0							

Updating dp table for row 4

Filling row 4

Sixth step: *Here we can see that for i = 5 and j = 5 the values of S1[4] and S2[4] are same (i.e., both are 'A'). So dp[5][5] is updated accordingly and becomes 3.*

		G	X	T	X	A	Y	B
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0	1	1	2	2	3	3	3
B	6	0							

Updating dp table for row 5

Filling row 5

Final step: *For i = 6, see the last characters of both strings are same (they are 'B'). Therefore the value of dp[6][7] becomes 4.*

		G	X	T	X	A	Y	B
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
A	1	0	0	0	0	0	1	1	1
G	2	0	1	1	1	1	1	1	1
G	3	0	1	1	1	1	1	1	1
T	4	0	1	1	2	2	2	2	2
A	5	0	1	1	2	2	3	3	3
B	6	0	1	1	2	2	3	3	4

Updating dp table for row 6

Filling the final row

So, we get the maximum length of common subsequence as ***4****.*

CODE:

```
// Dynamic Programming C implementation of LCS problem
#include <bits/stdc++.h>
using namespace std;
// Returns length of LCS for X[0..m-1], Y[0..n-1]
int lcs (string X, string Y, int m, int n)
{
    // Initializing a matrix of size (m+1) *(n+1)
    int L [m + 1] [n + 1];
    // Following steps build L[m+1] [n+1] in bottom up
    // fashion. Note that L[i][j] contains length of LCS of
    // X[0..i-1] and Y[0..j-1]
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)L[i][j] = 0;

            else if (X[i - 1] == Y[j - 1])
                L[i][j] = L [i - 1] [j - 1] + 1;
            else
                L[i][j] = max (L [i - 1] [j], L[i] [j – 1];
        }
       }
    // L[m][n] contains length of LCS for X[0..n-1]
    // and Y[0..m-1]return L[m][n];
}
// Driver code
int main()
{
    string S1 = "AGGTAB"; string S2 = "GXTXAYB";int  m =  S1.size();
    int n = S2.size();
    cout << "Length of LCS is " << lcs (S1, S2, m, n);
    return 0;
}
```

Output

Length of LCS is 4

Usage of LCS Problem:

It is a classic computer science problem, the basis of diff (a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

Recurrence Relation:

Let Xand Ybe the two strings we want to find the length of the LCS of,and X_i be the substring <X0,X1,...,Xi>and Y_j be the

substring <Y0,Y1,...,Yj>. X_i and Y_j are called *prefixes* of the

strings Xand Y. Further, let c[i,j]be the length of the optimal LCS betweenthe strings X_i and Y_j. Then we can have the recursive relation:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1][j-1] + 1 & \text{if } i,j > 0 \text{ and } x_i = y_i \\ max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Time Complexity: O (m * n) which is much better than the worst-case time complexity of Naive Recursive implementation.

Auxiliary Space: O (m * n) because the algorithm uses an array of size(m+1) *(n+1) to store the length of the common substrings.

Note: The longest common subsequence problem can be efficiently solved using dynamic programming, which involves breaking it down into smaller subproblems and buildingup a solution using the solutions to those subproblems. The algorithm has many practical applications in fields such as bioinformatics and text processing.

Recursion for LCS:

Algorithm:

Follow the below steps to implement the idea:

- Create a recursive function [say **lcs ()**].
- Check the relation between the last characters of the strings that arenot yet processed.
- Depending on the relation call the next recursive function asmentioned above.
- Return the length of the LCS received as the answer.

Illustration:

1. Let's consider two sequences, X and Y, of length m and n that both end in the same element.

To find their LCS, shorten each sequence by removing the last element, find the LCS of the shortened sequences, and that LCS append the removed element.

So, we can say that.

LCS (X [1…m], Y [1…n]) = LCS (X [1…m-1], Y [1…n-1]) + X[m] if X[m] = Y[n]

2. Now suppose that the two sequences does not end in the same symbol.

Then the LCS of X and Y is the longer of the two sequences LCS (X [1…m-1], Y

[1…n]) and LCS (X [1…m], Y [1…n-1]). To understand this property, let's considerthe two following sequences:

X: ABCBDAB (n elements)

Y: BDCABA (m elements)

The LCS of these two sequences either ends with B (the last element of the sequence X) or does not.

Case 1: If LCS ends with B, then it cannot end with A, and we can remove A from the sequence Y, and the problem reduces to LCS(X[1…m], Y[1…n-1]).

Case 2: If LCS does not end with B, then we can remove B from sequence X and the problem reduces to LCS (X [1…m-1], Y [1…n]).

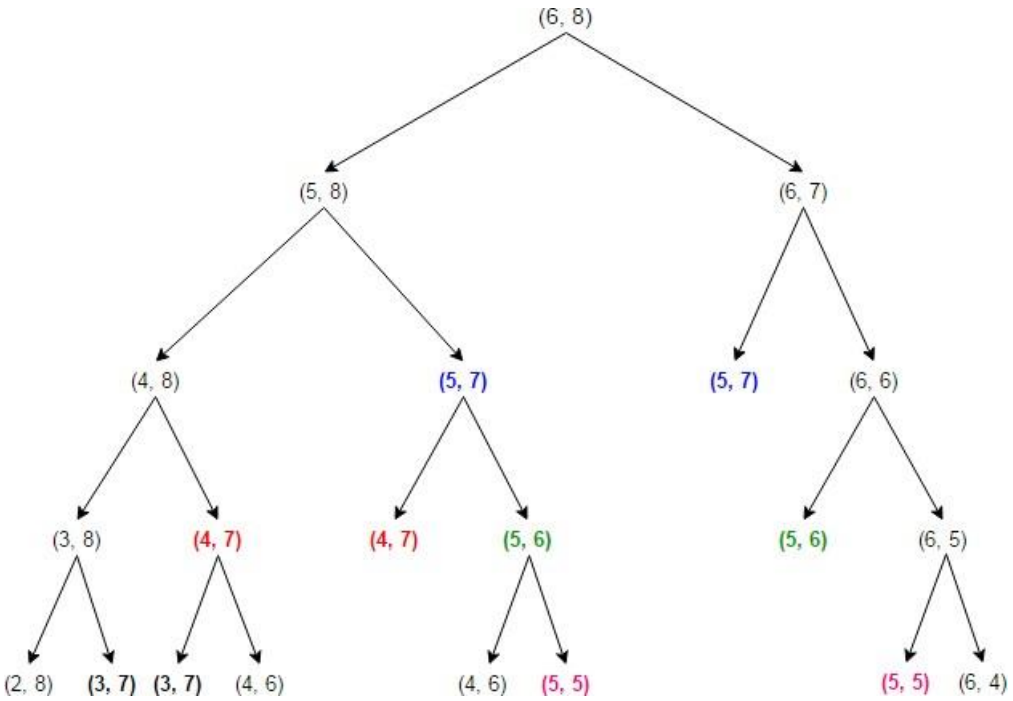
LCS (ABCBDAB, BDCABA) = maximum (LCS (ABCBDA, BDCABA), LCS (ABCBDAB, BDCAB))LCS (ABCBDA, BDCABA) = LCS (ABCBD, BDCAB) + A

LCS (ABCBDAB, BDCAB) = LCS (ABCBDA, BDCA) + B

LCS (ABCBD, BDCAB) = maximum (LCS (ABCB, BDCAB), LCS (ABCBD, BDCA))LCS (ABCBDA, BDCA) = LCS (ABCBD, BDC) + A

And so on…

Let's consider the recursion tree for two sequences of length 6 and 8 whose LCS is 0.



CODE:

```
#include <iostream>

#include <string>

using namespace std;


// Function to find the length of the longest common subsequence of
// sequences `X[0…m-1]` and `Y[0…n-1]`
int LCSLength(string X, string Y, int m, int n)
{
  // return if the end of either sequence is reached
  if (m == 0 || n == 0) {
    return 0;
  }
  // if the last character of `X` and `Y` matches
  if (X[m - 1] == Y[n - 1]) {
    return LCSLength(X, Y, m - 1, n - 1) + 1;
  }


  // otherwise, if the last character of `X` and `Y` don't match
  return max(LCSLength(X, Y, m, n - 1), LCSLength(X, Y, m - 1, n));
}


int main()
{
  string X = "ABCBDAB", Y = "BDCABA";

  cout << "The length of the LCS is " << LCSLength(X, Y, X.length(),
      Y.length());


  return 0;
}
```

Output:

The length of the LCS is 4.

Time Complexity: $O(2^{\wedge n})$

Auxiliary Space: O (1)

Recurrence Relation:

The recurrence relation would be:

T(n,m) = T(n-1,m-1)+O(1), if (X[m-1] = Y[n-1])

or

T(n-1,m)+T(n,m-1)+O(1), otherwise

We would have to consider the worst case scenario, which would be:

T(n,m) = T(n-1,m)+T(n,m-1)+O(1)

throughout. Which would boil down to:

T(n,m) <= 2^(n-1) T(0,m) + ... , if m<n (longest branch of height n)or

2^(m-1) T(n,0) + ... , if n<m (longest branch of height m)

Here if the longest branch is of length k, we get an upper limit if assume all other branches are of height k as well. Since both T(0,k) and T(k,0) are constants, we have

T(n,m) = O(2^(max(n,m)))Or

T(n,m) = O(2^n)

if n and m are equal.

Conclusion:

The recursive approach to solving the Longest Common Subsequence problem can beimproved with memorization to reduce its time complexity to O(2^n). However, for very long sequences, the exponential worst-case time complexity of the basic recursive approach may still be impractical. Other approaches, such as dynamic programming or the Hirschberg's algorithm, may be more suitable for larger inputs.

7. **Real Life Applications**

The Longest Common Subsequence (LCS) has a wide range of practical applications across different domains. Here are some of the key applications for LCS:

1. **Bioinformatics**: LCS is extensively used in bioinformatics to compare and analyze DNA, RNA, and protein sequences. It helps identify similarities between genetic sequences, which can provide insights into evolutionary relationships, genetic mutations, and functional similarities between different species. LCS algorithms are crucial for sequence alignment in tasks like genome comparison and identifying conserved regions.

2. **Text Comparison and Plagiarism Detection**: In natural language processing and document analysis, LCS is used to compare text documents, articles, or code files. It is instrumental in detecting plagiarism by finding common subsequences in texts. This is valuable for academic institutions, content publishers, and plagiarism detection services.

3. **Version Control Systems**: In software development, version control systems like Git use LCS to track changes between different versions of source code. It helps identify additions, deletions, and modifications in code files, making it easier to merge changes made by different developers and maintain a clean project history.

4. **Data Diff and Merge Tools**: Data differencing and merging tools leverage LCS algorithms to identify changes in structured data, such as databases, XML files, and configuration files. This is essential for managing and reconciling changes in data sources.

5**. Image Comparison and Video Processing**: LCS can also be applied to image and video analysis, where it helps find common patterns and similarities in images or video frames. This is useful for tasks like image recognition, video summarization, and video content retrieval.

6. **Natural Language Processing**: In NLP applications, LCS can be used for tasks like spell correction, sentence alignment, and identifying similar sentence pairs in translation tasks. It aids in measuring the similarity between text strings.

8. **References**

1. The longest common Closest Point Algorithm D. Chetverikov, D. Svirko, and D. Stepanov Computer and Automation Institute Budapest, Kendeu.13-17, H-Ill1 Hungary 2.

2.The Closest Point Algorithm Pave1 Krsek Center for Applied Cybernetics FEE, Czech echnical University

3. A Fast Iterative Point Algorithm for Support Vector Machine Classifier Design S. S. Keerthi, S. K. Shevade, C. karthikeya, and K. R. K. Murthy

4. Robust Euclidean alignment of 3D point sets: the trimmed iterative closest point algorithm

5.longest common subsequence Michael Ian Shamost and Dan Hoey Department of Computer Science, Yale University

6.Modified Sort Algorithm for Large Scale Data Sets Marcin Woźniak, Zbigniew Marszałek, Marcin Gabryel & Robert K. Nowicki

7.Performance analysis of merge sort algorithms Joella Lobo Dept of Electronics and Telecommunication, Farmagudi, Goa Goa college of engineering

8.Optimization And Automation Of bioinformatics: An Overview (10 Pt) Meryeme Hafidi, Mohamed Benaddy , Salahddine Kri