Mini Project

Submitted by

PRANAV MATHILAKATH
[RA2111030010078]. KESHAV
JAYAKRISHNAN [RA2111030010064]
ANANYA MADIREDDY
[RA2111030010071]

Under the Guidance of

Mrs. Lavanya V

Assistant Professor, Department of Networking and Communications

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE ENGINEERING

with specialization in CYBERSECURITY

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603203

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203
Chengalpattu District

BONAFIDE CERTIFICATE

Certified to be the bonafide work done by

Keshav Jayakrishnan (RA2111030010064)

Pranav Mathilakath (RA2111030010078)

Ananya Madireddy (RA2111030010071)

of II Year/IV Sem B.Tech Degree Course in the **Practical Course – 18CSC204J - Design and Analysis of Algorithms** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY,** Kattankulathur during the academic year 2022 – 2023.

SIGNATURE

Faculty In-Charge
Mrs. Lavanya V
Assistant Professor
Department of Networking and Communications
SRM Institute of Science and Technology

SIGNATURE

HEAD OF THE DEPARTMENT
Dr. Annapurani Panaiyappan. K
Professor and Head,
Department of Networking and Communications
SRM Institute of Science and Technology

LONGEST COMMON SUBSEQUENCE

Team Members	**Reg No:**
Keshav Jayakrishnan	RA2111030010064
Pranav Mathilakath	RA2111030010078
Ananya M	RA2111030010071

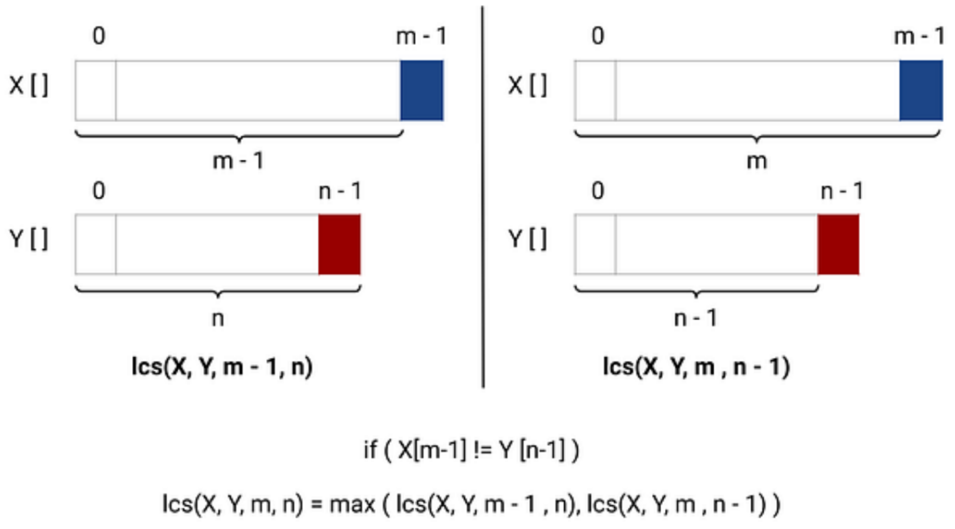
Introduction:

- Involves finding the longest subsequence that is common to two given sequences.
- LCS has numerous applications, including bioinformatics, text comparison, plagiarism detection, and version control.

Different Algorithmic approaches in LCS

Brute force algorithm:

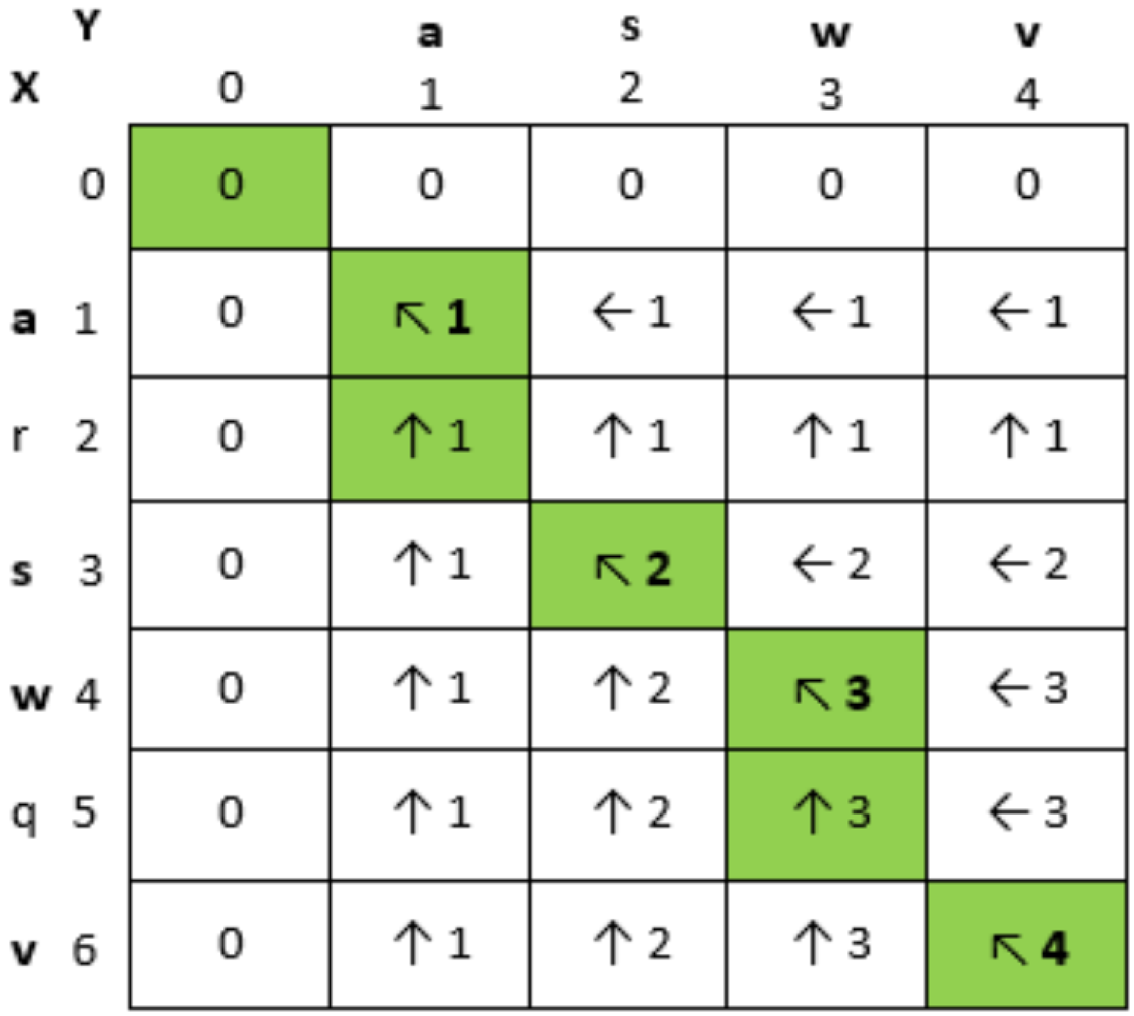
- The brute force algorithm is the most basic method for solving the LCS problem.
- It involves generating all possible subsequences of the two input sequences and checking if they are common to both sequences.
- The brute force algorithm has a time complexity of O(2^n) and a space complexity of O(n), where n is the length of the longer sequence.
- Although the brute force algorithm is simple to implement, it is not efficient for large input sizes.



Brute Force Method

Dynamic Programming:

- The dynamic programming algorithm for LCS is a more efficient method for solving the problem.
- It involves breaking down the input sequences into smaller subproblems and storing the solutions to these subproblems in a table.
- The dynamic programming algorithm has a time complexity of O(mn), where m and n are the lengths of the input sequences, and a space complexity of O(mn).
- The dynamic programming algorithm is widely used for solving the LCS problem due to its efficiency.



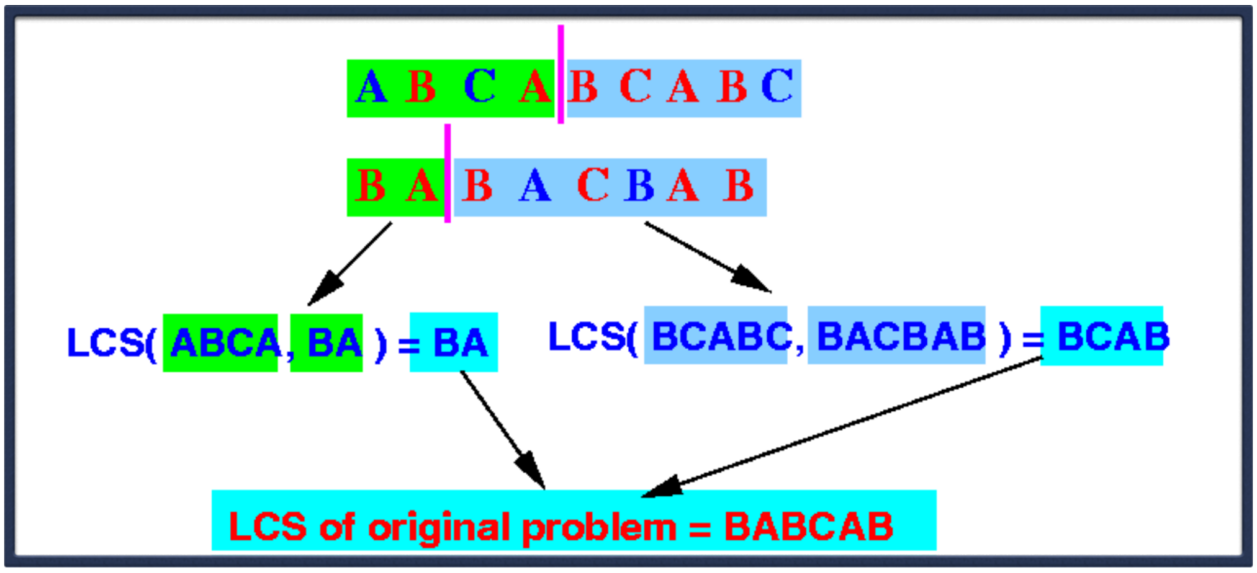
	Y		a	s	w	v
X		0	1	2	3	4
	0	0	0	0	0	0
a	1	0	↖**1**	←1	←1	←1
r	2	0	↑1	↑1	↑1	↑1
s	3	0	↑1	↖**2**	←2	←2
w	4	0	↑1	↑2	↖**3**	←3
q	5	0	↑1	↑2	↑3	←3
v	6	0	↑1	↑2	↑3	↖**4**

Dynamic programming

Divide and Conquer:

- The divide and conquer algorithm for LCS involves dividing the input sequences into smaller subproblems, solving these subproblems recursively, and then combining the solutions to obtain the LCS of the original sequences.
- The divide and conquer algorithm has a time complexity of O(n^2logn) and a space complexity of O(n^2).

- Although the divide and conquer algorithm is not as efficient as the dynamic programming algorithm, it is useful for solving other problems that involve finding common subsequences.



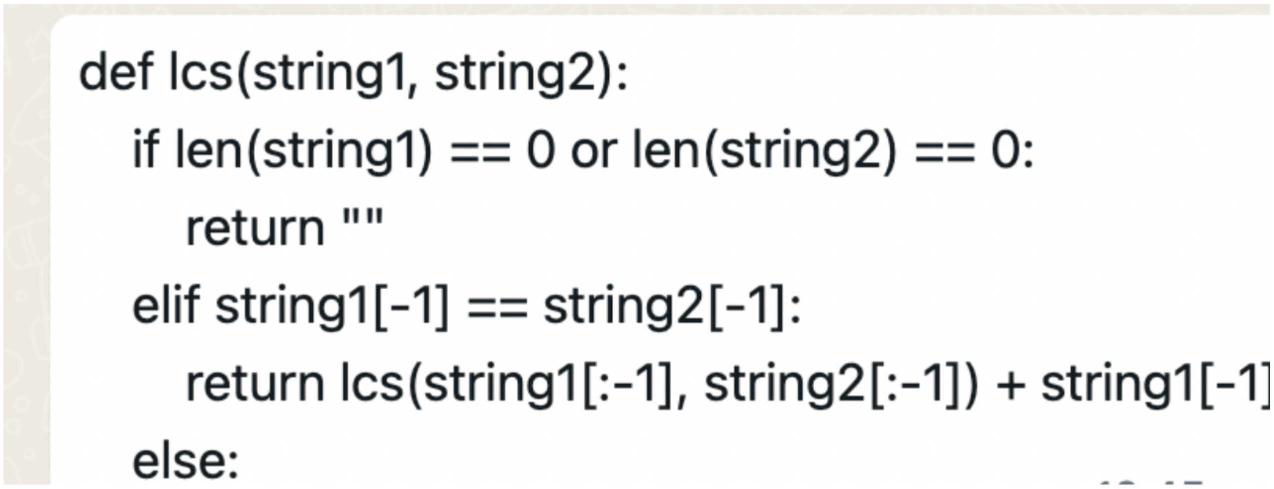
Divide and Conquer

Real Time Applications:

- Bioinformatics: LCS is used to compare DNA sequences and identify similarities between them. This information is used to understand the evolution of species, study genetic disorders, and develop new drugs.
- Text Comparison: LCS is used to compare text files and identify differences between them. This information is used in version control systems, plagiarism detection software, and document management systems.
- Image Processing: LCS is used to compare images and identify similar regions in them. This information is used in image registration, object detection, and image segmentation.
- Speech Recognition: LCS is used to compare spoken words and identify the words that are common to different utterances. This information is used in speech recognition systems to improve their accuracy.

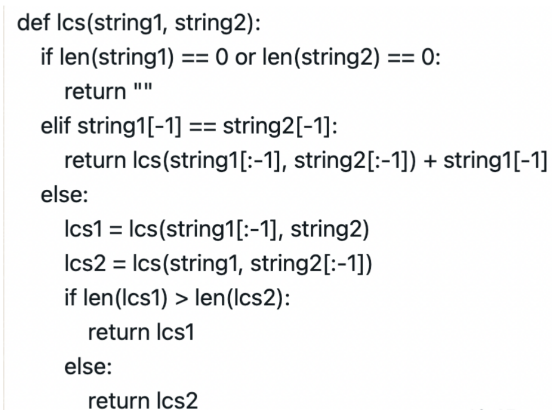
Comparison of Algorithms:

After comparing the brute force, dynamic programming, and divide and conquer algorithms based on their time and space complexity. We can conclude that the Dynamic programming algorithm is the most efficient algorithm for solving the LCS problem, with a time complexity of O(mn) and a space complexity of O(mn). However, the divide and conquer algorithm is useful for solving other problems that involve finding common subsequences.
The Brute force algorithm is the least efficient algorithm and is only suitable for small input sizes.



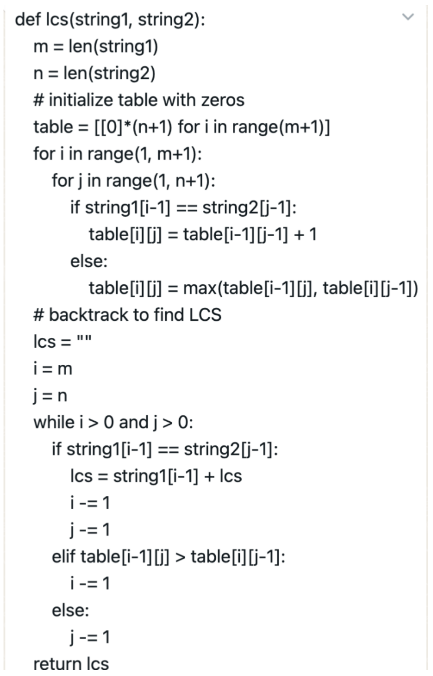
```
def lcs(string1, string2):
    if len(string1) == 0 or len(string2) == 0:
        return ""
    elif string1[-1] == string2[-1]:
        return lcs(string1[:-1], string2[:-1]) + string1[-1]
    else:
```

Brute Force Method



```
def lcs(string1, string2):
    if len(string1) == 0 or len(string2) == 0:
        return ""
    elif string1[-1] == string2[-1]:
        return lcs(string1[:-1], string2[:-1]) + string1[-1]
    else:
        lcs1 = lcs(string1[:-1], string2)
        lcs2 = lcs(string1, string2[:-1])
        if len(lcs1) > len(lcs2):
            return lcs1
        else:
            return lcs2
```

Dynamic Programming



```
def lcs(string1, string2):
    m = len(string1)
    n = len(string2)
    # initialize table with zeros
    table = [[0]*(n+1) for i in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if string1[i-1] == string2[j-1]:
                table[i][j] = table[i-1][j-1] + 1
            else:
                table[i][j] = max(table[i-1][j], table[i][j-1])
    # backtrack to find LCS
    lcs = ""
    i = m
    j = n
    while i > 0 and j > 0:
        if string1[i-1] == string2[j-1]:
            lcs = string1[i-1] + lcs
            i -= 1
            j -= 1
        elif table[i-1][j] > table[i][j-1]:
            i -= 1
        else:
            j -= 1
    return lcs
```

Divide and Conquer

Source Code:

Brute force method

```
#include <stdio.h>
#include <string.h>

int max(int a, int b) {
   return (a > b) ? a : b;
}

int lcs_brute_force(char *X, char *Y, int m, int n) {
   if (m == 0 || n == 0) {
      return 0;
   } else if (X[m-1] == Y[n-1]) {
      return 1 + lcs_brute_force(X, Y, m-1, n-1);
   } else {
      return max(lcs_brute_force(X, Y, m-1, n), lcs_brute_force(X, Y, m, n-1));
   }
}

int main() {
   char X[] = "AGGTAB";
   char Y[] = "GXTXAYB";

   int m = strlen(X);
   int n = strlen(Y);

   printf("Length of LCS is %d\n", lcs_brute_force(X, Y, m, n));

   return 0;
}
```

Output:

```
Length of LCS is 4
```

Dynamic Programming

```
#include<stdio.h>
```

```
#include<stdlib.h>
#include<string.h>

int max(int a, int b)
{
   return (a > b) ? a : b;
}

int longestCommonSubsequence(char *s1, char *s2, int m, int n)
{
   int dp[m+1][n+1];
   int i, j;

   // Initializing the dp table
   for(i=0;i<=m;i++)
   {
      for(j=0;j<=n;j++)
      {
         if(i==0 || j==0)
            dp[i][j] = 0;
         else if(s1[i-1] == s2[j-1])
            dp[i][j] = dp[i-1][j-1] + 1;
         else
            dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      }
   }

   return dp[m][n];
}

int main()
{
   char s1[] = "abcdgh";
   char s2[] = "aedfhr";
   int m = strlen(s1);
   int n = strlen(s2);

   printf("Length of longest common subsequence: %d", longestCommonSubsequence(s1, s2, m,
n));
```

```
    return 0;
}
```

Output:

```
Enter the first string: AGGTAB
Enter the second string: GXTXAYB
The length of the longest common subsequence is: 4
```

Divide and Conquer

```
#include <stdio.h>
#include <stdlib.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}

int longestSubsequence(int arr[], int left, int right) {
    if (left == right) {
        return 1;
    }
    if (arr[left] == arr[right] && left + 1 == right) {
        return 2;
    }
    if (arr[left] == arr[right]) {
        return longestSubsequence(arr, left + 1, right - 1) + 2;
    }
    return max(longestSubsequence(arr, left, right - 1),
               longestSubsequence(arr, left + 1, right));
}

int main() {
    int arr[] = {1, 4, 5, 6, 7, 8, 9, 9, 9, 10, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    int longest = longestSubsequence(arr, 0, n - 1);
    printf("The length of the longest subsequence is %d\n", longest);
    return 0;
}
```

Output:

```
Length of longest subsequence: 4
```

Conclusion

We discussed three different algorithms for solving the LCS problem: brute force, dynamic programming, and divide and conquer.
 We compared the time and space complexity of these algorithms and discussed their pros and cons. We also highlighted the numerous applications of LCS in computer science and industry. LCS is a fundamental problem in computer science, and efficient algorithms for solving it are crucial for many applications.