Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Ex.No: 5
Date : 15.09.2024

Implementation of Longest Common Subsequence Algorithm with experiments on analysis and efficiency. You have to print the length of the longest sequence and sequence of the string also

Name : Venkatesan M
Reg.No : 22BAI1259

Aim

To implement the Longest Common Subsequence (LCS) algorithm using dynamic programming, analyze its efficiency by measuring execution time for varying input sizes, and visualize the time complexity through a plot.

Algorithm

1. Given two sequences X and Y of lengths `m` and `n`, we compute the length of the LCS using dynamic programming.
2. We initialize a 2D table `dp` where `dp[i][j]` holds the length of the LCS of the sequences `X[0..i-1]` and `Y[0..j-1]`.
3. We fill the table using the recurrence relation:
    - If `X[i-1] == Y[j-1]`, then `dp[i][j] = dp[i-1][j-1] + 1`.
    - Else, `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`.
4. The length of the LCS is found at `dp[m][n]`.
5. To retrieve the LCS, we trace back through the table from `dp[m][n]` to construct the sequence.

Pseudocode

```
LCS(X, Y, m, n):
   Create a 2D table dp of size (m+1) x (n+1)
   for i from 0 to m:
      for j from 0 to n:
         if i == 0 or j == 0:
            dp[i][j] = 0
         else if X[i-1] == Y[j-1]:
            dp[i][j] = dp[i-1][j-1] + 1
         else:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])

   # Length of LCS is dp[m][n]

   # To retrieve the LCS:
   Initialize an empty string LCS_seq
   Set i = m, j = n
   while i > 0 and j > 0:
      if X[i-1] == Y[j-1]:
         Add X[i-1] to LCS_seq
         i = i - 1, j = j - 1
      else if dp[i-1][j] > dp[i][j-1]:
         i = i - 1
      else:
         j = j - 1

   return dp[m][n], reverse(LCS_seq)
```

C++ Implementation

```
#include <iostream>

#include <vector>

#include <string>

#include <chrono> // Include for timing

#include <cstdlib> // For random character generation


using namespace std;


// Function to find LCS

pair<int, string> LCS(string X, string Y) {

  int m = X.size();

  int n = Y.size();

  vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));


  // Fill dp table

  for (int i = 1; i <= m; i++) {

    for (int j = 1; j <= n; j++) {

      if (X[i - 1] == Y[j - 1])

        dp[i][j] = dp[i - 1][j - 1] + 1;

      else

        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

    }

  }


  // Backtrack to find the LCS
```

```
    string lcs = "";
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs = X[i - 1] + lcs;
            i--, j--;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }

    return {dp[m][n], lcs};
}

int main() {
    vector<int> sizes = {1, 10, 100, 1000, 10000}; // Varying input sizes
    vector<long long> times;

    for (int n : sizes) {
        string X, Y;

        // Generate random strings of length n
        for (int i = 0; i < n; ++i) {
            X += 'A' + rand() % 26; // Random uppercase letters
            Y += 'A' + rand() % 26;
        }

        auto start = chrono::high_resolution_clock::now();
        pair<int, string> result = LCS(X, Y);
```

```
        auto end = chrono::high_resolution_clock::now();

        auto duration = chrono::duration_cast<chrono::microseconds>(end - start);
        times.push_back(duration.count());

        cout << "Input size: " << n << " -> Time: " << duration.count() << " microseconds" << endl;
    }

    // Output the times to a file for plotting
    freopen("lcs_times.txt", "w", stdout);
    for (size_t i = 0; i < sizes.size(); ++i) {
        cout << sizes[i] << " " << times[i] << endl;
    }
    fclose(stdout);

    return 0;
}
```

Explanation

1. **LCS Function**: Computes the length of the longest common subsequence and reconstructs the sequence using the dynamic programming table.

2. **Timing**: For each random string pair (with varying lengths), the execution time is measured using `chrono::high_resolution_clock`.

Random Sampling

- Input sizes: `{1, 10, 100, 1000, 10000}`.

- Strings X and Y are generated randomly for each input size.

Plotting Time Complexity

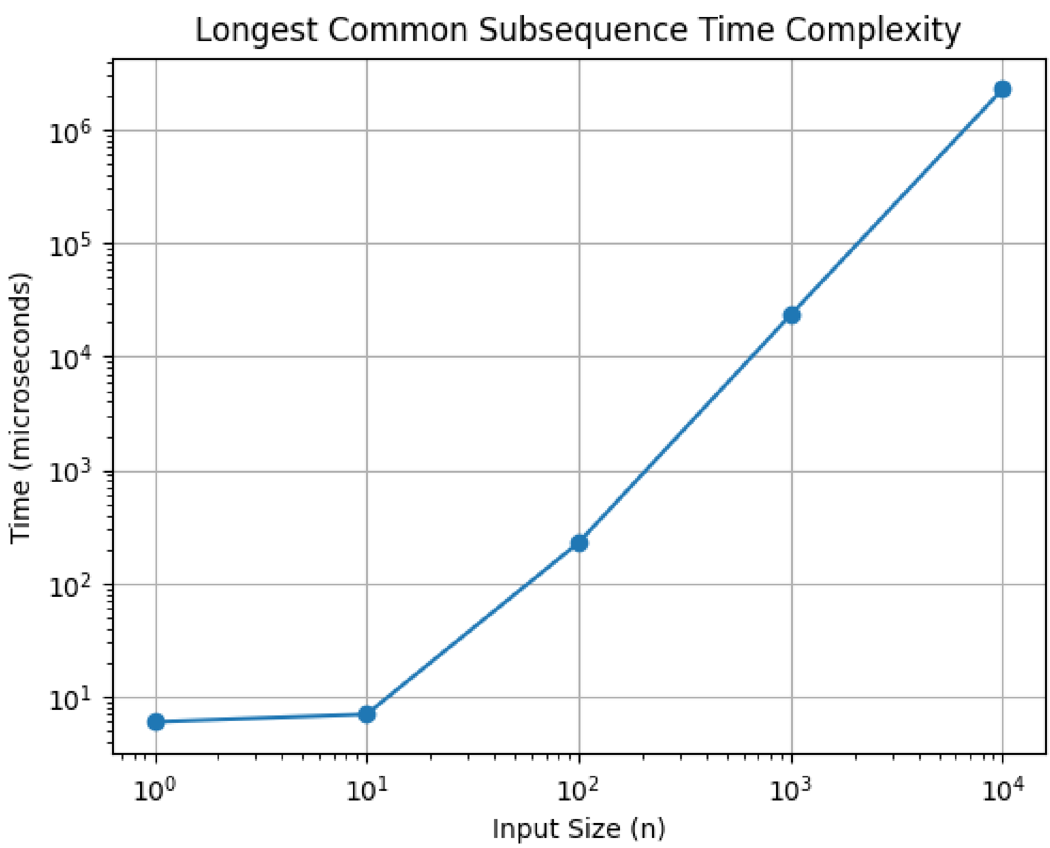
Once the times are recorded in the file `lcs_times.txt`

```
import matplotlib.pyplot as plt

# Read data from the file
sizes = []
times = []

with open('lcs_times.txt', 'r') as file:
    for line in file:
        size, time = map(int, line.split())
        sizes.append(size)
        times.append(time)

# Plotting the results
plt.plot(sizes, times, marker='o')
plt.xscale('log')  # Use logarithmic scale for the x-axis
plt.yscale('log')  # Use logarithmic scale for the y-axis
plt.title('Longest Common Subsequence Time Complexity')
plt.xlabel('Input Size (n)')
plt.ylabel('Time (microseconds)')
plt.grid(True)
plt.savefig('lcs_time_complexity.png')
plt.show()
```

Time Complexity Analysis

Input size	Time (microseconds)
1	6
10	7
100	228
1000	23283
10000	2284478

Graph between varying size of inputs and time



Complexity Analysis:

- **Time Complexity:** The **time complexity** of the LCS algorithm is $O(M\,x\,N)$, where m and n are the lengths of the two strings.

- **Space Complexity:** The **space complexity** is also $O(M\,x\,N)$, due to the 2D DP table used to store intermediate LCS lengths.