|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **03 / 04 / 2024** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Dr. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 10**

**Title: Longest Common Subsequence Algorithm**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of Longest Common Subsequence Algorithm.  **Objective:** To compute longest common subsequence for the given two strings. |

|  |
| --- |
| **COs to be achieved:** |
| **CO3:** Develop string matching techniques. |

|  |
| --- |
| **Theory:** |
| Given 2 sequences, X = x1 , ..., xm and Y = y1 , ... , yn , find a subsequence common to both whose length is longest. A subsequence doesn’t have to be consecutive, but it has to be in order.  **New Concepts to be learned:**  String matching algorithm, Dynamic programming approach for LCS, Applications of LCS  **Recursive Formulation:**  Define c[i, j ] = length of LCS of Xi and Y j .  Final answer will be computed with c[m, n].  c[i, j]= 0  if i=0 or j=0.  c[i, j]= c[i − 1, j − 1] + 1  if i,j>0 and xi=yj  c[i, j]= max(c[i − 1, j ], c[i, j − 1])  if i, j > 0 and xi <> yj |

|  |
| --- |
| **Stepwise-Procedure / Algorithm:** |
| Compute length of optimal solution –  **LCS-LENGTH** *( X , Y, m, n)*  **for** *i* ← 1 **to** *m*  **do** *c*[*i,* 0] ← 0  **for** *j* ← 0 **to** *n*  **do** *c*[0*, j* ] ← 0  **for** *i* ← 1 **to** *m*  **do for** *j* ← 1 **to** *n* **do**  **if** *xi* = *y j*  **then** *c*[*i, j* ] ← *c*[*i* − 1*, j* − 1] + 1  *b*[*i, j* ] ← “≈”  **else if** *c*[*i* − 1*, j* ] ≥ *c*[*i, j* − 1]  **then** *c*[*i, j* ] ← *c*[*i* − 1*, j* ]  *{b*[*i, j* ] ← “↑”  **else** *c*[*i, j* ] ← *c*[*i, j* − 1]  *b*[*i, j* ] ← “←”  **return** *c* and *b*  Print the solution –  **PRINT-LCS*(b, X , i, j )***  **if** *i* = 0 or *j* = 0  **then return**  **if** *b*[*i, j* ] = “≈”  **then** PRINT-LCS*(b, X , i* − 1*, j* − 1*)*  print *xi*  **elseif** *b*[*i, j* ] = “↑”  **then** PRINT-LCS*(b, X , i* − 1*, j )*  **else** PRINT-LCS*(b, X , i, j* − 1*)*  Initial call is PRINT-LCS*(b, X , m, n)*.  *b*[*i, j* ] points to table entry whose subproblem we used in solving LCS of *Xi* and *Y j.*  When *b*[*i, j* ] = ≈, we have extended LCS by one character. So longest common subsequence = entries with ≈ in them. |

|  |
| --- |
| **Upload the code / Output:** |
| Code:  Recursive Approach –  def lcs(X, Y, m, n):      if m == 0 or n == 0:          return 0, []      elif X[m - 1] == Y[n - 1]:          length, sequence = lcs(X, Y, m - 1, n - 1)          return length + 1, sequence + [X[m - 1]]      else:          length1, sequence1 = lcs(X, Y, m, n - 1)          length2, sequence2 = lcs(X, Y, m - 1, n)          if length1 > length2:              return length1, sequence1          else:              return length2, sequence2  def main():      print("\nLongest Common Subsequence - Recursive Approach")      X = input("Enter the first sequence: ")      Y = input("Enter the second sequence: ")      lcs\_length, lcs\_sequence = lcs(X, Y, len(X), len(Y))      print("\nLength of Longest Common Subsequence:", lcs\_length)      print("Longest Common Subsequence:", ''.join(lcs\_sequence))  if \_\_name\_\_ == '\_\_main\_\_':      main()  Backtracking Approach –  def lcs(X, Y, m, n):      if m == 0 or n == 0:          return 0      elif X[m - 1] == Y[n - 1]:          return 1 + lcs(X, Y, m - 1, n - 1)      else:          return max(lcs(X, Y, m, n - 1), lcs(X, Y, m - 1, n))  def print\_lcs\_table(X, Y):      m = len(X)      n = len(Y)      table = [[0] \* (n + 1) for \_ in range(m + 1)]      for i in range(1, m + 1):          for j in range(1, n + 1):              if X[i - 1] == Y[j - 1]:                  table[i][j] = table[i - 1][j - 1] + 1              else:                  table[i][j] = max(table[i - 1][j], table[i][j - 1])      return table  def print\_lcs(X, Y, table):      m = len(X)      n = len(Y)      lcs\_length = table[m][n]      lcs\_sequence = []      i, j = m, n      while i > 0 and j > 0:          if X[i - 1] == Y[j - 1]:              lcs\_sequence.append(X[i - 1])              i -= 1              j -= 1          elif table[i - 1][j] > table[i][j - 1]:              i -= 1          else:              j -= 1      lcs\_sequence.reverse()      return lcs\_length, ''.join(lcs\_sequence)  def main():      print("\nLongest Common Subsequence - Backtracking Approach")      X = input("\nEnter the first sequence: ")      Y = input("Enter the second sequence: ")      lcs\_table = print\_lcs\_table(X, Y)      lcs\_length, lcs\_sequence = print\_lcs(X, Y, lcs\_table)      print("\nLCS Table:")      for row in lcs\_table:          print(row)      print("\nLength of Longest Common Subsequence:", lcs\_length)      print("Longest Common Subsequence:", lcs\_sequence)  if \_\_name\_\_ == '\_\_main\_\_':      main()  Code Output:  Recursive Approach –    Backtracking Approach –    Handwritten Solution: |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **Define Longest Common Subsequence Problem. Give Dynamic Programming Solution for the given instance of problem.**  **X = Titanic**  **Y = Maverick**  **To solve the same,**  **1.     Write Recursive formula**  **2.     Solve by Table formulation and compute the answer**  **Compute the Longest Common Subsequence with length.** |

|  |
| --- |
| **Conclusion:** |
| We successfully implemented and learnt one of the string-matching algorithms, Longest Common Subsequence, using different approaches such as recursive and dynamic. |

**Signature of faculty in-charge with Date:**