VTOP: LAB ASSIGNMENT 2

REG NO: 23BKT0079

NAME: Shivam Kumar Jaiswal

SLOT: L37+L38

CLASS ID: 3993

SUBJECT CODE AND NAME: BCSE204P, Design and Analysis of Algorithm

VIT - Vellore

Name: SHIVAM KUMAR JAISWAL .
Email: shivamkumar.jaiswal2023@vitstudent.ac.in
Roll no: 23BKT0079
Phone: 9999999999
Branch: VETRISELVI T_DSA
Department: admin
Batch: VL2024250503993
Degree: admin

Scan to verify results

BCSE204P_Design and Analysis of Algorithms Lab_VL2024250503993

VIT V_DAA_Week 3_LCS_COD_Medium

Attempt : 1
Total Mark : 10
Marks Obtained : 10

1. Problem Statement Emma, a software developer, is working on a plagiarism detection tool for an educational platform. The platform receives large text documents from students, and Emma needs to compare the submitted papers to check for similarities. She decides to implement an algorithm that fnds the Longest Common Subsequence (LCS) between two documents, which are represented as strings. To achieve this, Emma needs a program that can effciently calculate the length of the LCS for these large document strings. Help Emma by writing a program that calculates the length of the LCS between two given document strings

Answer:

```
#include <stdio.h>

#include <string.h>

#defne MAX 1005

int dp[MAX][MAX];

void fndLCS(char *s1, char *s2, int len1, int len2, char lcs[MAX][MAX], int *count,

char *temp, int pos) {

 if (len1 == 0 || len2 == 0) {
```

```
 temp[pos] = '\0';
 for (int i = 0; i < *count; i++)
 if (strcmp(lcs[i], temp) == 0) return;
 strcpy(lcs[*count], temp);
 (*count)++;
 return;
 }
 if (s1[len1 - 1] == s2[len2 - 1]) {
 temp[pos] = s1[len1 - 1];
 fndLCS(s1, s2, len1 - 1, len2 - 1, lcs, count, temp, pos + 1);
 } else {
 if (dp[len1 - 1][len2] >= dp[len1][len2 - 1])
 fndLCS(s1, s2, len1 - 1, len2, lcs, count, temp, pos);
 if (dp[len1][len2 - 1] >= dp[len1 - 1][len2])
 fndLCS(s1, s2, len1, len2 - 1, lcs, count, temp, pos);
 }
}
void reverseStr(char *str) {
 int n = strlen(str);
 for (int i = 0; i < n / 2; i++) {
 char temp = str[i];
 str[i] = str[n - i - 1];
 str[n - i - 1] = temp;
 }
}
int main() {
 char s1[MAX], s2[MAX], lcs[MAX][MAX], temp[MAX];
 fgets(s1, MAX, stdin);
```

```
 fgets(s2, MAX, stdin);
 s1[strcspn(s1, "\n")] = '\0;
s2[strcspn(s2, "\n")] = '\0';
 int len1 = strlen(s1), len2 = strlen(s2);
 for (int i = 0; i <= len1; i++)
 for (int j = 0; j <= len2; 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] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];
 int count = 0;
 fndLCS(s1, s2, len1, len2, lcs, &count, temp, 0);
 for (int i = 0; i < count; i++) {
 reverseStr(lcs[i]);
 printf("%s\n", lcs[i]);
 }
 printf("The length of the Longest Common Subsequence is: %d\n", dp[len1]
[len2]);
 return 0;
}
```

Status : Correct Marks : 10/10

VIT - Vellore

Name: SHIVAM KUMAR JAISWAL .
Email: shivamkumar.jaiswal2023@vitstudent.ac.in
Roll no: 23BKT0079
Phone: 9999999999
Branch: VETRISELVI T_DSA
Department: admin
Batch: VL2024250503993
Degree: admin

Scan to verify results

BCSE204P_Design and Analysis of Algorithms Lab_VL2024250503993

VIT V_DAA_Week 3_LCS_CY

Attempt : 1
Total Mark : 20
Marks Obtained : 20

1. Problem Statement

Given two strings s and t, the task is to determine the length of their Longest Common Subsequence (LCS). A subsequence of a string is a sequence derived from the string by deleting some or no characters without changing the order of the remaining characters. The LCS is the longest subsequence that is common to both strings.

Example

Input:

s = "abcde"

t = "ace"

Output:

3

Explanation:

The LCS is "ace", which has a length of 3.

Answer

```
// You are using GCC
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int fndLCSLength(string& s1, string& s2) {
 int m = s1.length();
 int n = s2.length();

 // Create DP table
 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(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 LCS length
 return dp[m][n];
```

```
}
int main() {
 string s1, s2;
 getline(cin, s1);
 getline(cin, s2);

 cout << fndLCSLength(s1, s2) << endl;
return 0;
}
```

Status : Correct Marks : 10/10

2. Problem Statement

Dr. Sophia, a genetic researcher, is studying the DNA sequences of two species and aims to identify the Longest Common Subsequence (LCS) between their strands. The DNA sequences are represented as strings of characters (`A`, `C`, `G`, and `T`), and the LCS is the longest sequence that appears in both DNA strands in the same order, though not necessarily consecutively.

By fnding the LCS, Dr. Sophia can uncover the longest shared genetic segment between the species, which will provide valuable insights into their evolutionary similarities and differences.

Answer

```
#include <stdio.h>
#include <string.h>
#defne MAX 1005
int dp[MAX][MAX];
void fndLCS(char *s1, char *s2, int len1, int len2, char lcs[MAX][MAX], int *count,
char *temp, int pos) {
```

```
 if (len1 == 0 || len2 == 0) {
 temp[pos] = '\0';
 for (int i = 0; i < *count; i++)
 if (strcmp(lcs[i], temp) == 0) return;
 strcpy(lcs[*count], temp);
 (*count)++;
 return;
 }
 if (s1[len1 - 1] == s2[len2 - 1]) {
 temp[pos] = s1[len1 - 1];
 fndLCS(s1, s2, len1 - 1, len2 - 1, lcs, count, temp, pos + 1);
}
else {
 if (dp[len1 - 1][len2] >= dp[len1][len2 - 1])
 fndLCS(s1, s2, len1 - 1, len2, lcs, count, temp, pos);
 if (dp[len1][len2 - 1] >= dp[len1 - 1][len2])
 fndLCS(s1, s2, len1, len2 - 1, lcs, count, temp, pos);
 }
}
void reverseStr(char *str) {
 int n = strlen(str);
 for (int i = 0; i < n / 2; i++) {
 char temp = str[i];
 str[i] = str[n - i - 1];
 str[n - i - 1] = temp;
 }
}
int main() {
```

```
 char s1[MAX], s2[MAX], lcs[MAX][MAX], temp[MAX];
 fgets(s1, MAX, stdin);
 fgets(s2, MAX, stdin);
 s1[strcspn(s1, "\n")] = '\0';
 s2[strcspn(s2, "\n")] = '\0';
 int len1 = strlen(s1), len2 = strlen(s2);
 for (int i = 0; i <= len1; i++)
 for (int j = 0; j <= len2; 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] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] : dp[i][j - 1];
 int count = 0;
 fndLCS(s1, s2, len1, len2, lcs, &count, temp, 0);
 printf("All possible unique LCS combinations:\n");
 for (int i = 0; i < count; i++) {
 reverseStr(lcs[i]);
 printf("%s\n", lcs[i]);
 }
 printf("Length of LCS: %d\n", dp[len1][len2]);
return 0;
}
```

Status : Correct

Marks : 10/10

VIT - Vellore

Name: SHIVAM KUMAR JAISWAL .
Email: shivamkumar.jaiswal2023@vitstudent.ac.in
Roll no: 23BKT0079
Phone: 9999999999
Branch: VETRISELVI T_DSA
Department: admin
Batch: VL2024250503993
Degree: admin

Scan to verify results

BCSE204P_Design and Analysis of Algorithms Lab_VL2024250503993

VIT V_DAA_Week 3_Matrix chain_COD_Easy

Attempt : 1
Total Mark : 10
Marks Obtained : 10

1. Problem Statement

Alice is tasked with solving the Matrix Chain Multiplication problem using a dynamic programming approach. Given a sequence of matrices and their dimensions, her aim is to determine the minimum number of scalar multiplications required to compute the matrix product.

The problem is to fnd the optimal order of matrix multiplication, as the order in which the matrices are multiplied affects the number of operations.

Help Alice to complete the task

Answer

/ You are using GCC

```
#include <iostream>
#include <vector>
#include <climits>
```

```
using namespace std;
int matrixChainMultiplication(vector<int>& dimensions) {
 int n = dimensions.size() - 1; // Number of matrices

 // Create dp table for storing minimum operations
 vector<vector<int>> dp(n, vector<int>(n, 0));

 // Chain length starts from 2 to n
 for (int chainLen = 2; chainLen <= n; chainLen++) {
 for (int i = 0; i < n - chainLen + 1; i++) {
 int j = i + chainLen - 1;
 dp[i][j] = INT_MAX;

 // Try all possible splits and fnd minimum
 for (int k = i; k < j; k++) {
 int operations = dp[i][k] + dp[k+1][j] +
 dimensions[i] * dimensions[k+1] * dimensions[j+1];

 if (operations < dp[i][j]) {
 dp[i][j] = operations;
 }
 }
 }
 }

 // Return minimum operations needed for entire chain
 return dp[0][n-1];
}
```

```
int main() {
 // Read number of dimensions
 int n;
 cin >> n;

 // Read dimensions array
 vector<int> dimensions(n);
 for (int i = 0; i < n; i++) {
cin >> dimensions[i];
 }

 // Calculate and print result
 cout << matrixChainMultiplication(dimensions) << endl;

 return 0;
}
```

Status : Correct Marks : 10/10