11/3/24

Mani S. Palimkar
231071036

S4 BTech CE
Batch B

Algorithm for LCS

NAME: Longest common Subsequence
// i/p: Grades (AA, AB . . .) of 20 students
// o/p:- longest common subsequence

# Define lcs fn
def lcs (str1, str2):
~~dp =~~
Initialize a 2D list 'dp' of size (len(str1)+1) x (len(str2)+1) with all elements

For i from 1 to len(str1) (inclusive):
For j from 1 to len(str2) (inclusive):
If str[i-1] == str[j-1]:
dp[i][j] = dp[i-1][j-1] + 1

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

Initialize an empty list 'lcs_sequence'
i = len(str1), j = len(str2)

While i>0 and j>0
If str[i-1] == str2[j-1]:
Append str[i-1] to 'lcs_sequence'
Decrease i and j

else if dp[i-1][j] > dp[i][j-1]:
    Decrease i

Else:
    Decrease j

Return the reversed 'lcs_sequence' joined as a string.

Algorithm to validate grades
// i/p :- grades of students
// o/p :- Error o/p if grades are not valid

F^ validate_grades(grades):
    If length of grades is not equal to 40:
        Raise ValueError with message about invalid length

    If any character in grades is a digit:
        Raise ValueError with message about digits in sequence.

For i from 0 to len(grades) with step 2:
    If the substring grades[i:i+2] does not match the pattern [A-F]{2}:
        Raise ValueError with message about invalid format or special characters

Time complexity

- creating a 2D dp array of size (len(str1)+1) x (len(str2)+1) requires O(len(str1) * len(str2)) time.
- The nested loops iterate over all pairs of indices (i, j) where 1 <= i <= len(str1) and 1 <= j <= len(str2). For each pair, we perform a constant time operation (comparison or taking a maximum). This step also has a time complexity of O(len(str1) * len(str2))
- loop for reconstructing LCS sequence traverses both strings and thus takes at most O(min(len str1), len(str2))) time.

°° If len(str) = n then $O(n^2)$
otherwise O(len(str1) * len(str2))

conclusion :- Algorithm for LCS takes O(len(str1) * len(str2)) time.

Algorithm for Matrix Chain multiplication.

NAME :- matrix_chain_multiplication(N, arr):
// i/p :- N matrices.
// o/p :- Min. no. of multiplications reqd.

Fn matrix_chain_multiplication(N, arr):
    If N < 2:
        Return "Error: There must be at least 2 matrices for multiplication"

    If length of arr is not N+1:
        ~~If~~ dimension.
        Return "Error: The dimensions array length must be N+1"

    For each dimension in arr:
        If dimension is less than or equal to 0:
            Return "Error: Matrix dimensions must be +ve values".

    Initialize a 2D table dp of size (N x N) with all elements set to 0

    For L from 2 to N-1:
        For i from 1 to N-L:
            Set j = i + L - 1

Set dp[i][j] = infinity
For k from i to j-1:
Calculate q = dp[i][k] + dp[k+1][j] + arr[i-1] * arr[k] * arr[j]

Update dp[i][j] to be the minimum of dp[i][j] and q.

Return dp[1][N-1] # the min no. of mul reqd.

Time complexity

1) Outer loop runs O(N) times for chain lengths L from 2 to N-1.

2) Middle loop runs O(N) times, as i goes from 1 to N-L.

3) The inner loop runs O(N) times in the worst case for each pair (i,j).

∴ Overall time complexity of algo is $O(N^3)$.

Conclusion :- Overall time complexity for algorithm is $O(n^3)$