PAGE NO.:
DATE / /

DAA-LAB 6

TASK = SOLID PRINCIPLES :-

(1) S : Single Responsibility Principle
Each class should only have one job

(2) O : Open/Close Principle (OCP)
Software entities should be open for extension but closed for modification.
New functionality should be added without changing existing codes.

(3) L : Liskov Substitution principle
Objects of a superclass should be replacable with objects of a subclass without altering correctness of the program

(4) I : Interface Segregation principle (ISP)
Client should not be forced to implement interfaces they do not use. Make smaller more specific interfaces instead of one large interface

(5) D : Dependancy Inversion Principle
High level modules should not depend on low level modules. Both should depend on abstractions.

(A) LONGEST COMMON SUBSEQUENCE

(1.) Algorithm

```
def initialise (seq1[], seq2[])
        n = len(seq1)
        m = len(seq2)
        memo = []
        for i in range (n+1)
            r = [0] * (m+1)
            memo.append(r)

def LCS (n, m):
      if memo[n][m] != 0 :
            return memo[n][m]

      if n==0 or m==0 :
            ~~result = 1+LCS(n-1, m-1)~~
            result = 0

    elif seq1[n-1] == seq2[n-1]
            result = 1+LCS(n-1, m-1)

    else
          t1 = LCS(n-1, m)
          t2 = LCS(n, m-1)
          result = max (t1, t2)

    memo[n][m] = result
    return result
```

```
def extract( seq1[], seq2[], memo[][] )

    i, j = len(seq1), len(seq2)
    lcs_sequence = []

    while i!=0 and j!=0:
        if seq1[i-1] == seq2[j-1]:
            lcs_sequence.append(seq1[i-1])
            i -=1
            j -=1
        elif memo[i-1][j] >= memo[i][j-1]:
            i=0
        else:
            j-=0

    return lcs_sequence.reverse()

def find( sequences [] )
    for seq in range(1, len(sequences)):
        res = lcs ( res, seq)
        if !result:
            break

    return res.
```

PAGE NO.:
DATE / /

(2) ~~Testcases~~: Time Complexity

We pairwise calculate LCS, we need to iterate through both sequences

$\therefore TC = O(n \times n) = O(n^2)$

We have K such sequences

$\therefore TC = O(K(n^2))$

$\therefore$ Time Complexity $= O(K(n^2))$

... where $K = 20$ = number of sequences

(3) TestCases.

Positive TCS:

Positive TCS consist of 20 ~~subs~~ sequences with valid ~~Testcase~~ grades, and no missing values.

Output: LCS string.

Negative TCS:

(1) Sequence Dataset is empty
→ DISPLAY: sequence is empty

(2) If sequence has non-string characters
→ DISPLAY: All characters should be strings

(3) If sequence has special characters
→ Display: No special characters allowed

(4) If the characters have upper and lowercases
→ DISPLAY: Different cases.

PAGE NO.:
DATE / /

(B) MATRIX CHAIN MULTIPLICATION

(1) Algorithm

```
def matrix_chain (P) :
→ n = p.length - 1
→ let m [1... n, 1...n] and s [1... n-1, 2...n] be
  new tables
→   for i in range (1, n+1)
      m [i, i] = 0
→   for l = 2 to n
    { for i = 1 to n-l+1
        { j = i+l-1
          m [i, j] = ∞

→ for k in range (i, j) :
    { q = m [i][k] + m [k+1, j] + p[i-1] * p[k]
                                      * p[j]

→   if   q < m[i, j]
          m[i, j] = q
          s[i, j] = k
    }}}
→ return m, s
```

PAGE NO.:
DATE / /

(2) Time Complexity

$$T(n) = \sum_{l=2}^{n} \sum_{l=1}^{n-l+1} \sum_{k=i}^{n} = \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} (n-l+1) = \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} n$$

$$= \sum_{l=2}^{n} (n)(n-l+\cancel{1}-\cancel{1})$$

$$= \sum_{l=2}^{n} = (n)(n-l)(n-2+1)$$

$$= (n^2-nl)(n-1)$$

$$= n^3-n^2-n^2l+nl$$

$$\therefore T(n) = n^3-n^2-n^2l+nl$$

$$\therefore \boxed{TC = O(n^3)}$$

(3) Test Cases:

- Positive.

Valid input, i.e. non-negative dimensions and non-empty list.

Output ⇒ minimum operations = 2340

Optimal parenthesization = $(A_1 \times (A_2 \times A_3))$

- Negative

→ Negative dimensions: [10, 10, -15, 10]

DISPLAY ⇒ DIMENSIONS cannot be negative

→ Empty input: []

Display ⇒ Input is Empty : ERROR

→ Only one matrix: [5, 5]

Display → min. operations = 0

→ Decimal Values of Dimensions

OUPUT ⇒ Display: Dimensions should be int.