DAA LAB-6

Page No.
Date

1. Longest common subsequence:

```
// Implementing longest common subsequence to find
   longest subsequence between 2 strings.
// Input : String s1, s2, and 2 pointers m and n
// Output : length of the longest common subsequence

def lcs(s1, s2, m, n):

    if m == 0 or n == 0
        return 0
    if memo[m][n] != -1
        return memo[m][n]

    if s1[m-1] == s2[n-1]:
        memo[m][n] = 1 + lcs(s1, s2, m-1, n-1)
        return memo[m][n]

    else:
        memo[m][n] = max(lcs(s1, s2, m, n-1))
                         lcs(s1, s2, m-1, n))

    return memo[m][n]
```

Time Complexity:

a) Brute Force

- Generate all subsequences

→ for string s1 of length m ($m \leq n$) there are 2^m possible subsequences. This is because the characters of s1 can be either included or excluded in a subsequence leading to 2^m combinations

- Checking each subsequence

→ for each subsequence generated from s1 we need to check if it is a subsequence of s2 of length n. This is done in time $\sum_{i=1}^{n} 1 = O(n)$

$\therefore$ Time Complexity $= O(2^{\min(m,n)} * \max(m,n))$

$= O(2^m \times n)$

b) Subproblems

Without memoization the naive recursive approach ~~would~~ could have looked all the subsequence of each character s1 and s2. However with memoization we avoid recalculating subproblems. Instead we calculate them only once. There are 'm' possible lengths of substring of s1 and 'n' possible lengths of s2. $\therefore$ no. of unique subproblems is 'm x n'. Since there are 'm x n' unique subproblems and each problem is solved in constant time O(1) $\therefore$ $T(c) = O(m \times n)$

for 'n' strings:

```
Algorithm Find LCSMultiple (strings s)
// Computes the LCS among n strings
// Input: Array of n strings
// Output: LCS of n strings

max_seq = ""
for i = 0 to n-1:
    LCS = s[i]
    for j = 0 to n-1:
        if i ≠ j:
            LCS = FindLCS(LCS, s[j])
            if LCS == "":
                break
    max_seq = max(max_seq, LCS, key = length)

return max_seq
```

Time Complexity:

1) There are 2 loops, one from 0 to n-1, other from 0 to n-1
2) In the innermost operation, computing LCS is $O(L^2)$ in the worst case, assuming an average string length of L.
3) Hence total time complexity is

$$T = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} cL^2 = \cancel{O(n^2L^2)} \; n^2L^2 = O(n^2)$$

Test cases (lcs) : positive

①

ID	Grades
S1	CCDDAA...CC
L2	DDCCAB..FF
S3	FFBB...BBCC
⋮	⋮
S20	ABBC...CCDD

Output : DDD

②

ID	Grades
S1	AABB...AABB
S2	DDCC..AABB
S3	ABBC...BCBC
⋮	⋮
S20	BBAA..BBCC

Output : BBBBC

③

ID	Grades
S1	BCFF...AABB
S2	CCDD...ABBC
S3	BBCC...DDFF
⋮	⋮
S20	AAAB...BBBC

Output: BBCCC

④

ID	Grades
S1	DDFF...BCCC
S2	ABBB...BCCC
S3	AAAB..BCDD
⋮	⋮
S20	BCCC...DDFF

Output. BBCCC

⑤

ID	Grades
S1	ABBCCC..DV
S2	AAAB...CCDD
S3	AAAA..BBBB
⋮	⋮
S20	ABBC...CCDD

Output. BCCD

Page No
Date

Negative Test Cases

① ID — Grade
S1 BBCC...CCFF
S2 AAAB...BCC@
S3 ABBC...C@DD
⋮
S20 ABCC...AABB

Expected output: special character used

② ID — Grade
S1 AABBCC
S2 ABBCCCDD
⋮
S20 AB

Expected output: string length less than 40

③ ID — Grade
S1 A1B2...C3D4
S2 A2B8...CCDD
⋮
S20 A9BB...CCDF

Expected output: Numbers used in string

④ ID — Grade
S1 A1B@...CCDD
S2 ABBC...C@D4
⋮
S20 AABB...CCDD

Output: string length less than 40 and special characters

⑤ ID — Grades
S1 AFBD...EECK
S2 AABC...CBBD
⋮
S20 ACCA...ABBK

Output: Invalid grades

Matrix Chain Multiplication (p)

// We implement Matrix Chain Multiplication to find most effecient way to multiply 'n' matrices.
// Input : List containing size of matrix & 2 pointers i & j
// output : The minimum steps of matrix chain.

```
def mcm(p):

    n = p.length - 1
    let m[1...n, 1...n] and s[1,..n-1, 2...n] be
                                    new tables

    for i = 1 to n
        m[i,i] = 0
    for j = 2 to n
        for i = 1 to n-l+1
            j = i+l-1
            m[i,j] = ∞
            for k = i to j-1
                q = m[i,k] + m[k+1, j] + pi-1 pk pj

                if q < m[i,j]
                    m[i,j] = q
                    s[i,j] = k

return m and s
```

Time Complexity

$$\sum_{i=1}^{n} 1 + \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{j-1} 1$$

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

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

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

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

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

$\because (l-1)(n-l+1) = (l-1)(n+1) - l(l+1)$

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

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

$$\left[\because \sum_{l=2}^{n} l-1 = \sum_{l=1}^{n-1} l\right]$$

$$= n + \frac{(n+1)n(n-1)}{2} - \left(\frac{n(n+1)(2n+1)}{6} - 1\right) + \left(\frac{n(n+1)}{2} - 1\right)$$

$$= n + \frac{n^3}{2} - \frac{n^2}{2} - \frac{2n^3+3n^2+n}{6} + 1 + \frac{n^2+n}{2} - 1$$

$$\approx O(n^3)$$

Page No.
Date

Test cases : mcm

1] ([7, 5, 4, 6, 7, 8], 5) → o/p : 504

2] ([3, 7, 5, 10, 15], 4) → o/p : 255

3] ([2, 4, 5, 6, 8], 4) → o/p : 100

4] ([4, 8, 6, 7, 9], 4) → o/p : 360

5] ([7, 3, 6, 4, 8], 4) → o/p : 156

6] ([3, 7, 4, 7, 5]) → o/p : Error

7] ([2, 5, 6], 1) → o/p : Error

8] ([10, 20, 30], 2) → o/p : Error

9] ([10, -20], 1) → o/p : Error

10] ([0, 20, 10], 2) → o/p : Error

Conclusion :

Hence, we have studied the SOLID principles of programming and also studied Algorithms to find longest Common subsequence and matrix chain multiplication.