Arpita Singh
23107100 5

LAB-6 DAA

Task-1 (Algorithm)

1] ✱ Brute force Algorithm
Input // A list of students grades, each students grade as a sequence of string.
→ Generate All subsequences:
- For each students list of grades, generate possible sequence.
- Sequence can be by deleting elements without changing order.

→ Find Common Subsequences:
- Compare subsequences of first student with that of second student and so on for all students.
- keep track of longest common subsequence

→ Return the longest common subsequence.

✱ Optimal Solution using dynamic programming
Input // A list of students grad as a sequence of string
→ Initialize a Multi-dimensional Dp table:
- Create Dp table with entry dp[i1][i2]...[in] for all.
- DP has dimension m+1 (m - no of grades) extra for zero-length.

→ Initialize dp[0][0]...[0] if no grades from any students.
→ For each indices i2, i2, i3.. in iterate
- if grades of students match at current indices -
  dp[i1][i2]..[in] = dp[i1-1][i2-1]..[in-1] + 1
- if grades of students do not match we exclude current grade
  dp[i1][i2]...[in] = max(dp[i1-1][i2]...[in], d[i1][i2]..[in])

→ After filling DP table, backtrack from last cell
- If grades match at all indices, move diagonally (reduce index by 1)
- If not match we move to neighbouring cell that gave max value

→ Return the longest common subsequence - The LCS found during backtracking step is longest common subsequence.

Test Cases

2) Positive cases

① Input : { AA, AB, BB, AA, AB } { AB, BB, BC, AA, AB }
Expected output: AB BB AA AB

② Input: { AA, AB, BB, CC, AB } { AB, BB, BC, AA, AB }
Expected output : AB BB AB

③ Input: { BB, BC, CD, AA, AB } { AA, BC, AA, CD, AB }
Expected output: BC A A AB

④ Input : { AB, BB, CC, BC, FF } { BB, AB, BC, FF, CC }
Expected output : BB AB BC FF

⑤ Input : { AA, AB, BB, CC, AB } { AB, BB, BC, AA, AB }
Expected output : AB BB AB

Negative Test Cases.

① Input : { AA AB, XY, BB, BC }
Expected output : Invalid grade XY found.

② Input : { AA, AB, BB } { AA, AB, BC, CD, DD }
Expected output : Each student must have 5 grades.

③ Input : { }
Expected output : Student has no grades.

④ Input : { AA, BB, 12, CC }, CD } { AA, BB, FF, CD, CC }
Expected output : Invalid grade 12 found.

⑤ Input : { AA, BC, CD, CC, AA, FF }
Expected output : More than 5 grades are invalid.

231071005

3]

a) Time Complexity

For each students m grades (Brute force)
no of subsequence of list of length m is 2^m
as two options - include or not include
To compare 2^m subsequence from student 1 and 2^m student 2, Extending this to n students we get $O(2^m \times 2^m \ldots \times 2^m)$ for each students

$$O(n \times 2^m \times 2^{nm}) = O(2^{nm})$$

This is exponentially expensive for n (20 students) and m (no of grades per student.

b] Optimal Solution using dynamic programming.

DP table setup —
table has dimension (m+1) for each of n students)
so entries $-(m+1) \times (m+1) \times \ldots (m+1)$ (n times)
no, of entries in DP table is $(m+1)^n \approx O(m^n)$

Backtracking — from last entry to reconstruct we need to backtrack only student index so O(n)

However,
Ik, for the typical two-dimensional LCS problem, the time complexity is $O(m \times n)$ for two sequences only

Here for multiple students (n being no. of students) time complexity to evaluate each combination is $O(mn)$.

Arpita Singh

2310710005

Lab - 6 DAA

Experiment Task - 2 (Algorithm)

1) * Brute force Algorithm -

→ Generate All Parenthesis - generate all possible ways split the chain eg for $A_1A_2A_3$ = ((A1 * A2) * A3) and (A1 * (A2 * A3))

→ For each parenthesization, calculate total no of scalar multiplication by multiplying the matrices in subchains eg for A(i*j) and B(j*k) cost is i*j*k.

→ After evaluating all possible parenthesization Return the minimum cost

** Optimal Solution using dynamic programming

→ Define DP table : Let m[i][j] be minimum of scalar product. fill DP table m to store optimal solution.

→ Base Case - If only one matrix (i==j) then m[i][i] = 0 for all i

→ Recursive relation : For each pair (i,j) compute m[i][j]

$$m[i][j] = \min_{k=i}^{j-1} (m[i][k] + m[k+1][j] + p[i-1] \times p[k] \times p[j])$$

Here m[i][k] is min cost of multiplying Ai through AK, m[k+1][j] for m[k+1] through j and, p[i-1]*p[k]*p[j] is cost of multiplying two resulting matrix from subchain.

→ Fill DP table - fill table starting with small chain length and move towards larger chains.

For each chain length 1, we compute m[i][j] for all possible i and j such that i < j and chain length is 1.

→ Final Result - After filling in table, the min number of scalar multiplications for multiplying matrices from A_1 to A_n will be stored in m[1][n]

Test Cases

2] Positive test Cases –

① Input : { 10, 20, 30, 40, 30, 20, 10 }
Expected output : $((A_1(A_2(A_3(A_4(A_5A_6))))))$

② Input : { 5, 10, 15, 20, 25, 30, 35 }
Expected output : $(((A_1A_2)((A_3A_4)))(A_5A_6))$

③ Input : { 3, 6, 9, 12, 15, 18, 21 }
Expected output : $((A_1(A_2(A_3(A_4(A_5A_6))))))$

④ Input : { 10, 5, 30, 15, 20, 25, 5 }
Expected output : $(((A_1(A_2A_3)A_4)((A_5A_6)))$

⑤ Input : { 2, 4, 6, 8, 10, 12, 14 }
Expected output : $(((A_1A_2)((A_3A_4)))(A_5A_6))$

Negative test Cases –

① Input : { 10, 20, 10, 5, 30, 15 }
Expected output : Invalid matrix chain dimension.

② Input : { 10, 20, 10, AB, 60 }
Expected output : Invalid input.

③ Input : { 20, 40, 30, 10, 15, 30 }
Expected output : Invalid matrix chain dimension.

④ Input : { }
Expected output : No values available.

⑤ Input : { 20, 25, 10, 15, 50, 25 }
Expected output : Invalid matrix chain dimension.

23107100 5

Time complexity

3] A) Brute force

$n = 1$ if just one matrix

$n >= 2$ a fully parenthesized matrix product, and the split between two subproducts may occur between k^{th} and $(k+1)^{th}$ matrix for any $k = 1, 2, 3 \ldots n-1$

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)\,P(n-k) & \text{if } n \geq 1 \end{cases}$$

The recurrence seq is Catalan nombes which grows as $\Omega\left(\frac{4^n}{n^{3/2}}\right)$

B) Optimal using dynamic programming

- Filling the DP table
  - The DP table m[i][j] has $O(n^2)$ entries (for each pair A_i to A_j)
  - For each pair m[i][j] we compute minimum over all possible splits k between i and j.

no of possible splits for each pair is $O(n)$, because k can range from i to j-1

Total time complexity -

$O(n^2)$ pairs of matrix and for each pair, the computation involves checking $O(n)$ possible splits.

Therefore total time complexity is:

$$O(n^2) \times O(n) = O(n^3)$$