**Q. (1)**

For 4 matrix, total 3 multiplications

To calculate number of ways these matrices can multiply between each other we can use **Catalan Number.**

Formula- B(n) =

In this case n =3

Therefore, B(3) = ¼ (6C3)

= 5 ------- (1) number of combinations

**Matrix chain multiplication formulation-**

m[ i , j ] = optimal number of multiplications necessary to multiply matrices Mi through Mj

= m[i , k ] + m[ k+1 , j ] + pi-1 . pk . pj

**-------------- (2)**

**For i<=k<=j**

Given, P0= 13 p1 = 5 p2 = 89 p3= 3 p4 = 34 ------------ (a)

We have to calculate m [1, 4], so From (2) –

M[I,4] = min

Here we know, m[1,1] = m[4,4] = 0

M[1,2] = 13\*5\*89 = 5785

M [3, 4]=89\*3\*34 = 9078

M[2, 3] = 5\*89\*3 = 1335

For [1,3] =min

= min

= 1530 ------------------------(3)

For [2,4] =min

= min

= 1845 ------------------------(3)

M[1, 4] = min

= min

= min

**M[1, 4] = 2856**

**So the optimal Multiplication is 2856 at k = 3 and k = 1**

**So the order (A × (B × C)) × D.**

**Q. (2)**

**BCBA**

X = ABCBDAB

Y = BDCABA

**Longest Common Subsequence Length Formulation-**

X = <x1 …. xm>

Xi = <x1 …. xi>

Y = <y1 …. yn>

Yj = <y1 …. yj>

C[i, j] = length of the LCS of Xi and Yj

**C [i, j] =**

**Longest Common Subsequence Dynamic Programming Algorithm-**

X and Y be two given sequences

Initialize a table LCS of dimension X.length \* Y.length

X.label = X

Y.label = Y

LCS[0][] = 0

LCS[][0] = 0

Start from LCS[1][1]

Compare X[i] and Y[j]

If X[i] = Y[j]

LCS[i][j] = 1 + LCS[i-1, j-1]

Point an arrow to LCS[i][j]

Else

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

Using above formulation and above algorithm, we can find LCS length and store it in DP table.

Also suing arrow we can store the char where it matches-

The DP table-

X = A**BCB**D**A**B

Y = **B**D**C**A**BA**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i\j | 0 | 1 **B** | 2 D | 3 **C** | 4 A | 5 **B** | 6 **A** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 A | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 **B** | 0 | **1 \** | <- 1 | 1 | 1 | 2 | 2 |
| 3 **C** | 0 | 1 | 1 | **2\** | <- 2 | 2 | 2 |
| 4 **B** | 0 | 1 | 1 | 2 | 2 | **3\** | 3 |
| 5 D | 0 | 1 | 2 | 2 | 2 | 3| | 3 |
| 6 **A** | 0 | 1 | 2 | 2 | 3 | 3 | **4\** |
| J B | 0 | 1 | 2 | 2 | 3 | 4 | 4| |

So, the LCS is BCBA.

**Q. (3)**

**Interleaving String Formulation-**

Given String

S1 = <s1 …. sm>

S2 = <s21 …. s2n>

S1i = <s11 …. s1i>

S2j = <s21 …. s2j>

S3 = <s31 …. s3k>

S3i+j = <s31 …. s3i+j>

C [i, j] = **True** if S3i+j is an interleaving string of S1i and S2j, **False** otherwise.

Case 1: If i=j=0:

Base case- no string

C[i, j] = True

Case 2: If i=0:

C[0, j] = C[0, j-1]

Case 3: If j=0:

C[i, 0] = C[i-1, 0]

Case 4:

Look if S3i+j-1 is interleaving S1i-1 and S2j , i.e. look for value c [i-1, j]

c[i, j] = c [i-1, j]

Case 5:

Look if S3i+j-1 is interleaving S1i and S2j-1 , i.e. Look for value c [i, j-1]

c[i, j] = c [i, j-1]

Case 6: :

Not interleaving, Assign False

C[i, j] = False

**C [i, j] =**

• **Pseudo-code –**

IS(S1, S2, S3):

r 🡨 Length (S1)

c 🡨 Length (S2)

dp[0][0] = True

for i 🡨 1 to r do dp[i][0] 🡨 s3[i] == s1[i] and dp[i-1][0]

for j 🡨 1 to c do dp[0][j] 🡨 s3[j] == s1[j] and dp[0][j=1]

for i 🡨 1 to r do

for j 🡨 1 to c do

if s1[i] == s3[i+j] then

dp[i][j] 🡨 dp[i-1][j]

else if s2[j] == s3[i+j] then

dp[i][j] 🡨 dp[i][j-1]

return dp[r-1][c-1]

**Q. (4)**

**Longest Palindromic Subsequence Formulation-**

Given String S = < s0, s1, …, sm>

C [i, j] = length of the Longest Palindromic Subsequence of substring starting from i to j index.

i.e. < si, si+1, …, sj>

Case 1: If i=j:

single char is palindromic

return 1

Case 2:

Add these 2 to length to the inside window c[i+1, j-1]

2 + c [i+1, j-1]

Case 3: :

Take max of inner windows

Max(c[i+1, j], c[I, j-1])

**C [i, j] =**

• **Pseudo-code –**

LPS(S):

n 🡨 Length (S)

for i 🡨 0 to n-1 do c[i, i] 🡨 1

// cl is window length

// now loop window from 2 to n length

for cl 🡨 2 to cl<=n do

for i 🡨 0 to i<n-cl+1 do

j = i+cl-1

if xi = xj and cl==2 then

c[i, j] 🡨 2

else if xi = xj then

c[i, j] 🡨 c[i+1, j-1] + 2

Else:

c[i, j] 🡨 max(c[i+1, j], c[i, j-1])

return c[0,n-1]