Algorithm-6

—— Matrix-Multiply[Dynamic.Programming]

A. Problem Description

Matrix chain multiplication is an optimization problem which is that can be solved using dynamic programming. Given a sequence of matrices, we want to find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:

(ABC)D = (AB)(CD) = A(BCD) = A(BC)D = ...

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example,

suppose A is a 10 x 30 matrix, B is a 30 x 5 matrix, and C is a 5 x 60 matrix. Then,

(AB)C = (10x30x5) + (10x5x60) = 1500 + 3000 = 4500 operations

A(BC) = (30x5x60) + (10x30x60) = 9000 + 18000 = 27000 operations.

B. *Description of algorithm*

Let m[i][j] be the minimum number of scalar multiplications needed to compute the matrix A[i : j].

$$m[i][j] = \begin{cases} 0 & i = j \\ \min\{m[i][k] + m[k+1][j] + p_{i-1} p_k p_j\} & i \le k < j \end{cases}$$

```
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 = 1 to n
        m[i, j] = 0
    for l = 2 to n
        for i = 1 to n – l + 1
            j = i + l – 1
            m[i , j] = inf
```

for k = i to j – 1

$q = m[i, k] + p_{i-1} p_k p_j m[k+1, j] +$

if q < m[i, j]

m[i, j] = q

s[i, j] = k

return m and s

C. *Time Complexity*

T = $O(n^3)$

D. *Code[Python]*

```
#!/usr/bin/python
# Filename: MatrixChain.py

def MatrixChain(p, n, m, s):
  for i in range(1, n + 1):
    m[i][i] = 0;
  for r in range(2, n + 1):
    for i in range(1, n - r + 2):
      j = i + r - 1
      m[i][j] = m[i + 1][j] + p[i - 1] * p[i] * p[j]
      s[i][j] = i
      for k in range(i + 1, j):
        t = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j]
        if t < m[i][j]:
          m[i][j] = t
          s[i][j] = k

def Traceback(i, j, s):
  if i == j:
    return
  Traceback(i, s[i][j], s)
  Traceback(s[i][j] + 1, j, s)
```

```
print 'Multiply A', i, ',', s[i][j],
print 'and A', s[i][j] + 1, ',', j
```