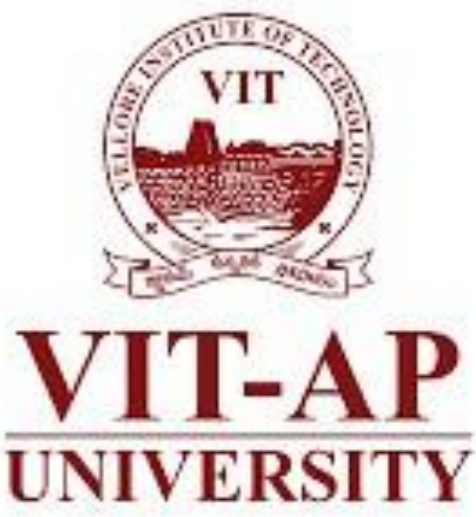

LAB REPORT

Assignment 7

NAME : P.SATHWIKA

REGISTRATION NUMBER : 21BCE8118

SUBJECT : Design and Analysis of Algorithms.

LAB SLOT : L53+L54

IMPLEMENTATION OF MATRIX MULTIPLICATION

USING DYNAMIC PROGRAMMING

Matrix Chain Multiplication is an algorithm that is applied to determine the lowest cost way for multiplying matrices. The actual multiplication is done using the standard way of multiplying the matrices, i.e., it follows the basic rule that the number of rows in one matrix must be equal to the number of columns in another matrix. Hence, multiple scalar multiplications must be done to achieve the product.

PSUEDOCODE:

```
MatrixChainOrder(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 l = 2 to n
        for i = 1 to n-l+1
            j = i + l - 1
            m[i,j] = INFINITY
            for k = i to j-1
                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 and s


PrintOptimalParentheses(s, i, j)
    if i == j
        print "A" + i
    else
        print "("
        PrintOptimalParentheses(s, i, s[i,j])
```

```
    PrintOptimalParentheses(s, s[i,j]+1, j)
    print ")"
```

ALGORITHM:

Define a function MatrixChainOrder(p) that takes a list of dimensions of matrices as input.

Initialize variables n as the length of p, m as a table for storing minimum number of scalar multiplications, and s as a table for storing split index.

Initialize the main diagonal of m with zeros.

Iterate over the chain length l from 2 to n.

Iterate over start index i from 1 to n-l+1.

Calculate the end index j.

Initialize m[i,j] with infinity.

Iterate over split index k from i to j-1.

Calculate the cost of multiplication at split index k.

Update minimum cost in m and store split index in s.

Return tables m and s.

CODE:

```
#sathwika 21BCE8118
print("P.Sathwika 21BCE8118\n")
def matrix_chain_order(p):
    n = len(p) - 1
    m = [[0 for _ in range(n)] for _ in range(n)]
    s = [[0 for _ in range(n)] for _ in range(n)]
    for l in range(2, n+1):
        for i in range(0, n-l+1):
            j = i + l - 1
            m[i][j] = float('inf')
```

```
            for k in range(i, j):
                q = m[i][k] + m[k+1][j] + p[i]*p[k+1]*p[j+1]
                if q < m[i][j]:
                    m[i][j] = q
                    s[i][j] = k
    return m, s
def print_optimal_parentheses(s, i, j):
    if i == j:
        print(f"A{i}", end="")
    else:
        print("(", end="")
        print_optimal_parentheses(s, i, s[i][j])
        print_optimal_parentheses(s, s[i][j]+1, j)
        print(")", end="")
# Example Usage:
dimensions = [15, 8, 7, 9, 12]
m_result, s_result = matrix_chain_order(dimensions)
print_optimal_parentheses(s_result, 0, len(dimensions)-2)
```

```python
#sathwika 21BCE8118
print("P.Sathwika 21BCE8118\n")
def matrix_chain_order(p):
    n = len(p) - 1
    m = [[0 for _ in range(n)] for _ in range(n)]
    s = [[0 for _ in range(n)] for _ in range(n)]
    for l in range(2, n+1):
        for i in range(0, n-l+1):
            j = i + l - 1
            m[i][j] = float('inf')
            for k in range(i, j):
                q = m[i][k] + m[k+1][j] + p[i]*p[k+1]*p[j+1]
                if q < m[i][j]:
                    m[i][j] = q
                    s[i][j] = k
    return m, s
def print_optimal_parentheses(s, i, j):
    if i == j:
        print(f"A{i}", end="")
    else:
        print("(", end="")
        print_optimal_parentheses(s, i, s[i][j])
        print_optimal_parentheses(s, s[i][j]+1, j)
        print(")", end="")
# Example Usage:
dimensions = [15, 8, 7, 9, 12]
m_result, s_result = matrix_chain_order(dimensions)
print_optimal_parentheses(s_result, 0, len(dimensions)-2)
```

OUTPUT:

```
P.Sathwika 21BCE8118

(A0((A1A2)A3))
```