**Assignment No. 2**

**Aim:** To implement Matrix Chain Multiplication using Dynamic Programming Approach

**Code:**

import java.io.\*;

import java.util.\*;

class Main {

static char name;

static void printParenthesis(int i, int j, int n, int[][] bracket)

{

// If only one matrix left in current segment

if (i == j) {

System.out.print(name++);

return;

}

System.out.print("(");

// Recursively put brackets around subexpression

// from i to bracket[i][j].

// Note that "\*((bracket+i\*n)+j)" is similar to bracket[i][j]

printParenthesis(i, bracket[i][j], n, bracket);

// Recursively put brackets around subexpression

// from bracket[i][j] + 1 to j.

printParenthesis(bracket[i][j] + 1, j, n, bracket);

System.out.print(")");

}

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n

static void matrixChainOrder(int p[], int n)

{

int[][] m = new int[n][n];

// bracket[i][j] stores optimal break point in

// subexpression from i to j.

int[][] bracket = new int[n][n];

for (int i = 1; i < n; i++)

m[i][i] = 0;

// L is chain length.

for (int L = 2; L < n; L++) {

for (int i = 1; i < n - L + 1; i++) {

int j = i + L - 1;

m[i][j] = Integer.MAX\_VALUE;

for (int k = i; k <= j - 1; k++) {

// q = cost/scalar multiplications

int q = m[i][k] + m[k + 1][j]

+ p[i - 1] \* p[k] \* p[j];

if (q < m[i][j]) {

m[i][j] = q;

// Each entry bracket[i,j]=k shows

// where to split the product arr

// i,i+1....j for the minimum cost.

bracket[i][j] = k;

}

}

}

}

for (int a = 0; a < n; a++){

for (int b = 0; b < n; b++){

System.out.print(m[a][b] + " ");

}

System.out.println();

}

// The first matrix is printed as 'A', next as 'B',

name = 'A';

System.out.print("\nOptimal Parenthesization is : ");

printParenthesis(1, n - 1, n, bracket);

System.out.print("\nOptimal Cost is : " + m[1][n - 1]);

}

public static void main(String[] args)

{

int arr[] = { 5,4,6,2,7 };

int n = arr.length;

matrixChainOrder(arr, n);

}

}

**Output:**