KARIYA RAJ 21162101011

ALGORITHM ANALYSIS AND DESIGN

PRACTICAL -10

Given a sequence of matrices, we want to find the most efficient way to multiply these matrices together to obtain the minimum number of multiplications. The problem is not actually to perform the multiplication of the matrices but to obtain the minimum number of multiplications.
We have many options because matrix multiplication is an associative operation, meaning that the order in which we multiply do not matter. The optimal order depends only on the dimensions of the matrices.
The brute-force algorithm is to consider all possible orders and take the minimum. This is a very inefficient method.
Implement the minimum multiplication algorithm using dynamic programming and determine where to place parentheses to minimize the number of multiplications.

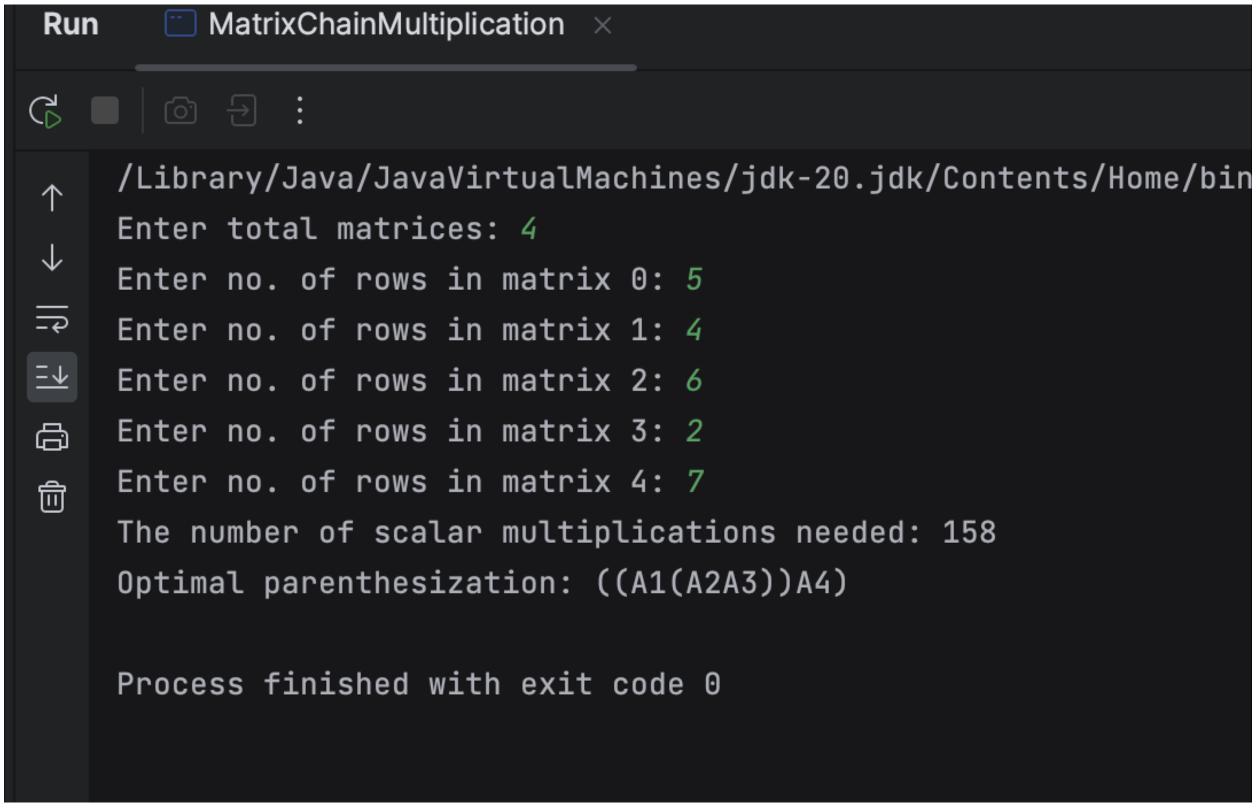
Find an optimal parenthesization of a matrix chain product whose sequence of dimensions are (5, 10, 3, 12, 5, 50, 6).
Input:
Enter total matrices: 4
Enter no. of rows in matrix 1: 5
Enter no. of rows in matrix 2: 4
Enter no. of rows in matrix 3: 6
Enter no. of rows in matrix 4: 2
Enter no. of columns in matrix 4: 7
Output:
The number of scalar multiplications needed: 158
Optimal parenthesization: ((A[1](A[2]A[3]))A[4])

CODE:

```
public class MatrixChainMultiplication {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter total matrices: ");
int n = scanner.nextInt();

int[] dimensions = new int[n + 1];
for (int i = 0; i <= n; i++) {
System.out.print("Enter no. of rows in matrix " + i + ": ");
dimensions[i] = scanner.nextInt();
}

int[][] dp = new int[n][n];
int[][] parenthesization = new int[n][n];

for (int chainLength = 2; chainLength <= n; chainLength++) {
for (int i = 0; i < n - chainLength + 1; i++) {
int j = i + chainLength - 1;
dp[i][j] = Integer.MAX_VALUE;
for (int k = i; k < j; k++) {
int cost = dp[i][k] + dp[k + 1][j] + dimensions[i] * dimensions[k + 1] *
dimensions[j + 1];
if (cost < dp[i][j]) {
dp[i][j] = cost;
parenthesization[i][j] = k;
}
}
}
}

System.out.println("The number of scalar multiplications needed: " + dp[0][n -
1]);
System.out.println("Optimal parenthesization: " +
printOptimalParenthesization(parenthesization, 0, n - 1));
}

private static String printOptimalParenthesization(int[][] parenthesization,
int i, int j) {
if (i == j) {
return "A" + (i + 1);
}
return "(" + printOptimalParenthesization(parenthesization, i,
parenthesization[i][j])
+ printOptimalParenthesization(parenthesization, parenthesization[i][j] + 1,
j) + ")";
}
}
```

OUTPUT:

KARIYA RAJ 21162101011



```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin
Enter total matrices: 4
Enter no. of rows in matrix 0: 5
Enter no. of rows in matrix 1: 4
Enter no. of rows in matrix 2: 6
Enter no. of rows in matrix 3: 2
Enter no. of rows in matrix 4: 7
The number of scalar multiplications needed: 158
Optimal parenthesization: ((A1(A2A3))A4)

Process finished with exit code 0
```