

Design and Analysis of Algorithms

Topic: Matrix Multiplication

Submitted By: Chetna Dua (205124026)

Submitted To: Ms. Eshwari

**Abstract**

The presentation provides a comprehensive overview of matrix multiplication techniques, from traditional methods to advanced optimizations.

It begins with the classical matrix multiplication algorithm, explaining its O(N^3) complexity and its inefficiency for large datasets.

Strassen’s Algorithm is introduced as an optimized solution using a divide-and-conquer approach:

* Reduces recursive multiplications from eight to seven, achieving a complexity of approximately O(N^2.81)
* Includes detailed steps such as matrix partitioning, intermediate computations (S1 to S10), and final matrix assembly.

Explores the Matrix Chain Multiplication (MCM) problem using dynamic programming to:

* Minimize scalar multiplications by optimal parenthesization of matrix chains.
* Utilize recursive subproblem solving and efficient computation of split indices.

Introduces multithreading as a modern approach for matrix multiplication:

* Distributes computational tasks across threads to parallelize and speed up operations.
* Optimizes addition and multiplication processes for large matrices.

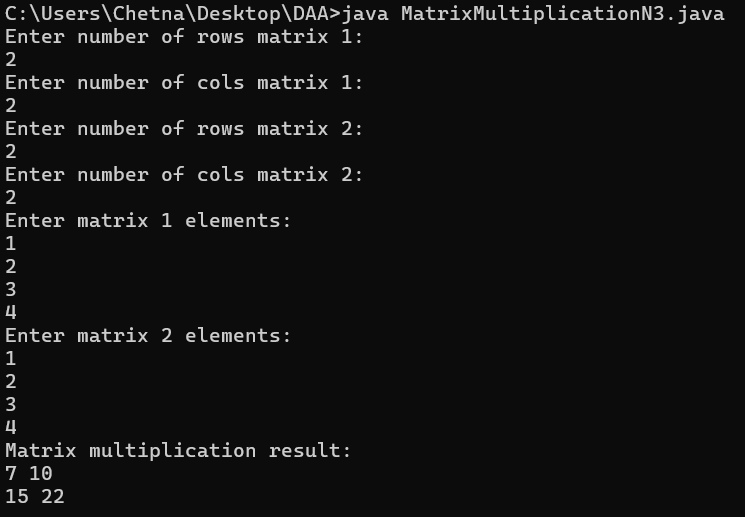
Highlighting the evolution of matrix multiplication algorithms, emphasizing optimization and parallelism for real-world applications.

**Core Concepts Discussed**

#### **1. Traditional Matrix Multiplication**

* **Algorithm Explanation:**Traditional matrix multiplication involves three nested loops, iterating through rows and columns to compute the resulting matrix C[i][j] as the summation of products from matrices A and B
  + Complexity: O(N^3) which is computationally intensive for large matrices.

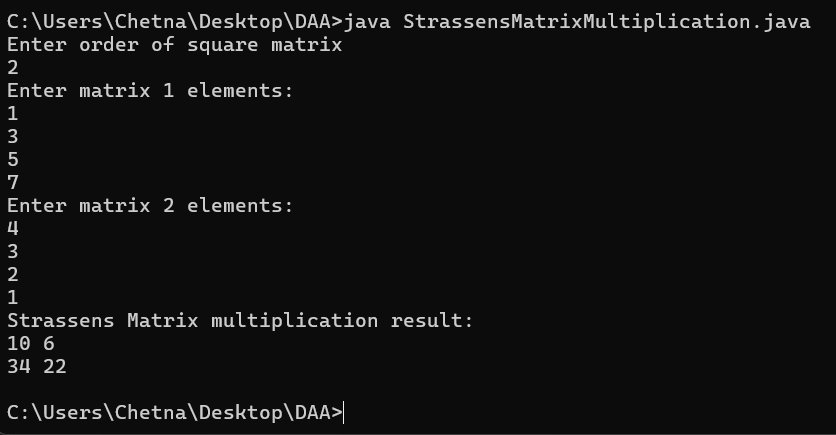
Code: **MatrixMultiplicationN3.java**



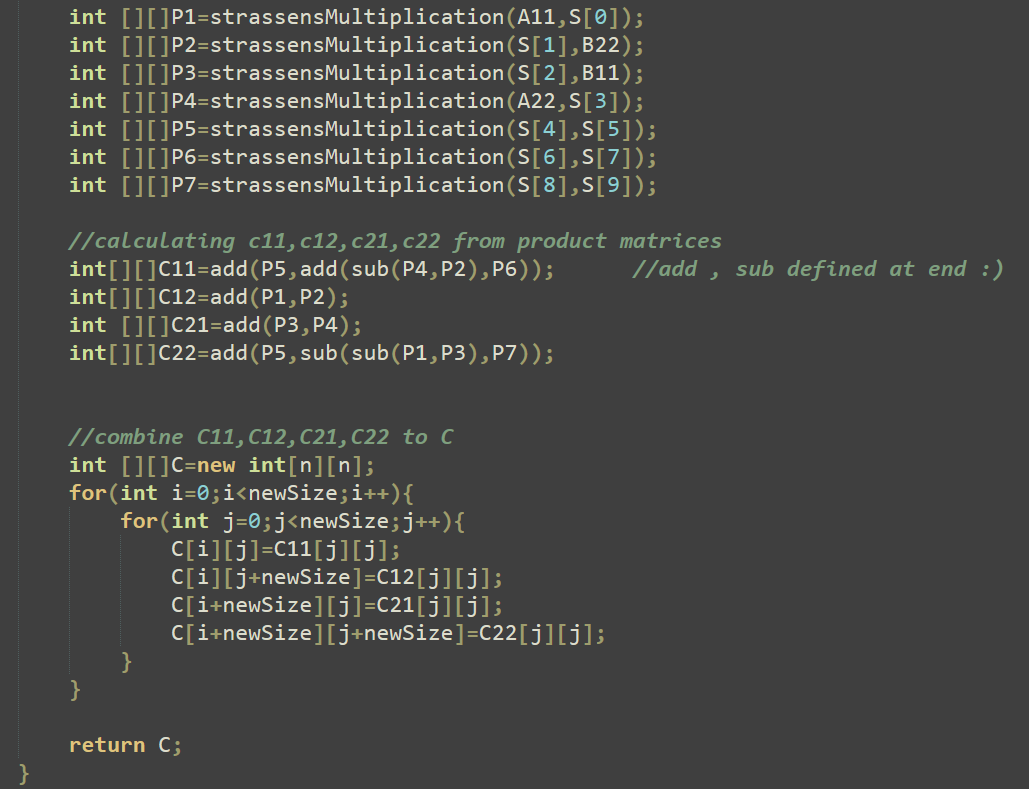
#### **2. Strassen’s Algorithm**

* **Motivation:** A divide-and-conquer approach to reduce the number of multiplications from 8 (traditional method) to 7, achieving a complexity of approximately O(N2.81)
* **Steps:**
  + **Matrix Partitioning:** Divide A, B, and C into submatrices of order N/2
  + **Matrix Operations:**
    - Create intermediate matrices S1 to S10 using constant additions of size N/2
    - Compute products P1 to P7 recursively.
    - Derive final submatrices C11, C12, C21, and C22 from the intermediate products.
* **Advantage:**By reducing one multiplication and replacing it with constant additions, it balances computational effort effectively for larger datasets.

Code: **StrassensMatrixMultiplication.java**



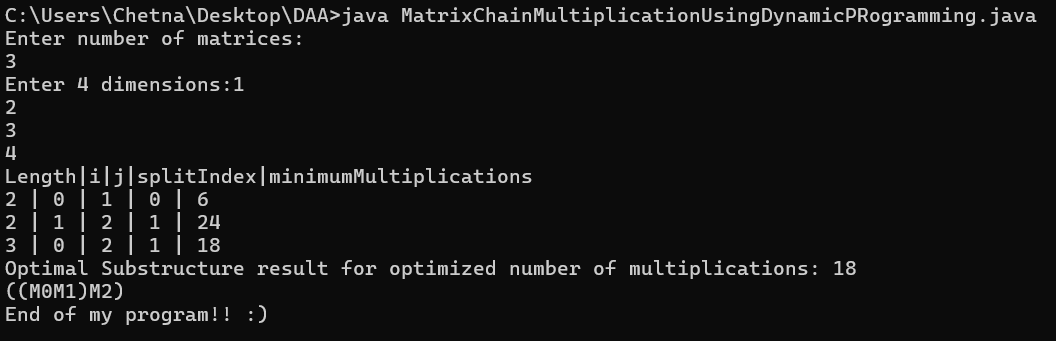
Core of Code



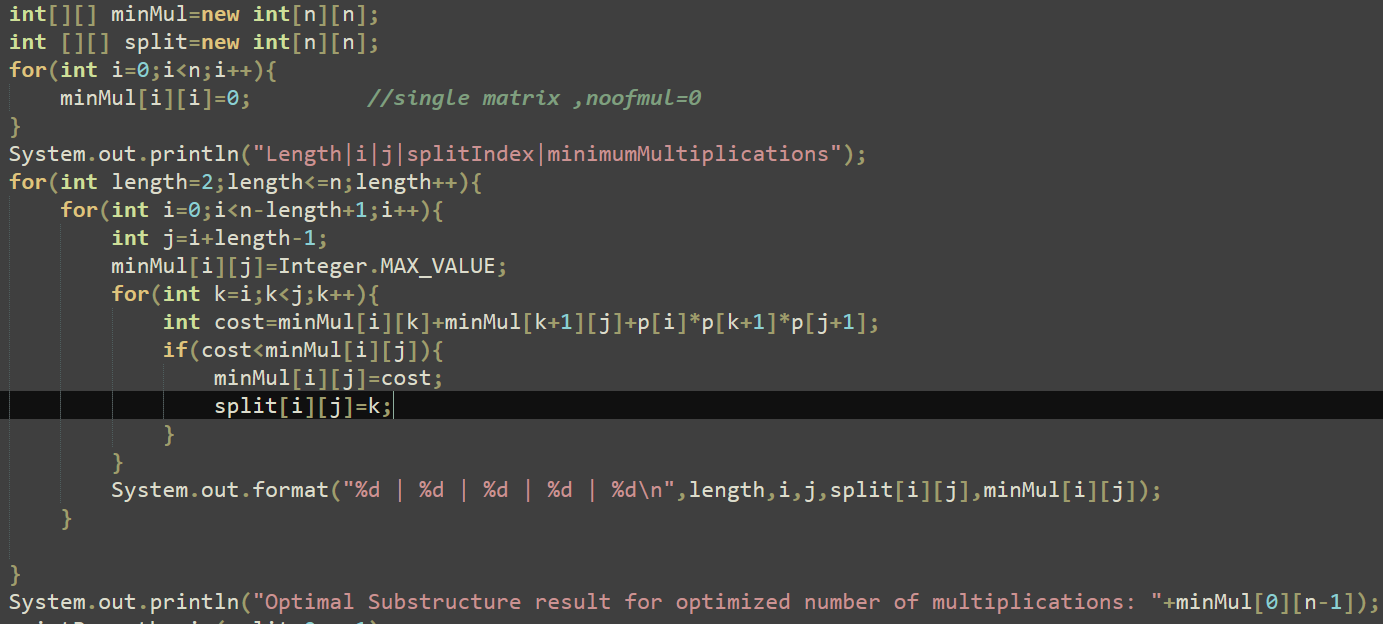
#### **3. Matrix Chain Multiplication (MCM)**

* **Dynamic Programming Approach:** Tackles the problem of parenthesizing a chain of matrices to minimize scalar multiplications.
* **Key Techniques:**
  + **Parenthesizing Variations:** For a chain of matrices A1,A2,A3,A4 different arrangements are considered to evaluate their computational cost.
  + **Optimal Substructure:** Uses dynamic programming to find the split index k that minimizes operations for every subproblem, leading to an overall minimal computation
  + **Complexity:** The recursive breakdown involves calculating and storing the results of smaller subproblems to optimize larger calculation

Code: **MatrixChainMultiplicationUsingDynamicProgramming.java**



I have printed minimum multiplications, split index at each code iteration

Code Crux:

#### **4. Multithreading for Matrix Multiplication**

* **Parallelism Insight:** Demonstrates how multithreading can distribute the computation load for matrix multiplication, particularly in splitting and adding operations.
* **Execution Strategy:**
  + Employs four threads to handle partial computations (e.g., calculating the upper and lower halves).
  + Combines the results of parallel threads efficiently.
  + Final addition step: O(N2)O(N^2)O(N2).
* **Potential:** Leverages modern hardware capabilities to enhance speed and scalability for large matrix operations.

Github Link for Java Code: