Day 12: Matrix Multiplication

Gati Goyal

"Mathematics is not about numbers, equations, or algorithms: it is about understanding."
— William Paul Thurston

---

1 Introduction

Matrix multiplication is a fundamental operation in linear algebra with wide applications in computer science, engineering, and data science. The task involves computing the dot product of rows from the first matrix and columns from the second matrix.

2 Problem Statement

Problem: Multiply two matrices of compatible sizes. **Hint:** Use nested loops for dot product computation. **Edge Case:** Ensure the number of columns in the first matrix equals the number of rows in the second matrix.

3 Algorithm

1. Input dimensions of the two matrices.
2. Check if the matrices are compatible for multiplication (`c1 == r2`).
3. Initialize a result matrix of size `r1 x c2` with zeros.
4. Compute each element of the result matrix:
   - Iterate through rows of the first matrix and columns of the second matrix.
   - Use a nested loop for element-wise multiplication and summation.

4 Code

```c
#include <stdio.h>

void inputMatrix(int rows, int cols, int matrix[rows][cols]) {
    printf("Enter␣elements␣of␣the␣%dx%d␣matrix:\n", rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
}

void printMatrix(int rows, int cols, int matrix[rows][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d␣", matrix[i][j]);
        }
        printf("\n");
    }
}

void multiplyMatrices(int r1, int c1, int r2, int c2, int mat1[r1][c1], int
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < c1; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

int main() {
    int r1, c1, r2, c2;

    printf("Enter␣rows␣and␣columns␣for␣the␣first␣matrix:␣");
    scanf("%d␣%d", &r1, &c1);

    printf("Enter␣rows␣and␣columns␣for␣the␣second␣matrix:␣");
    scanf("%d␣%d", &r2, &c2);

    if (c1 != r2) {
        printf("Matrix␣multiplication␣not␣possible.␣Columns␣of␣first␣matrix
        return 1;
    }

    int mat1[r1][c1], mat2[r2][c2], result[r1][c2];

    inputMatrix(r1, c1, mat1);
    inputMatrix(r2, c2, mat2);

    multiplyMatrices(r1, c1, r2, c2, mat1, mat2, result);

    printf("Resultant matrix after multiplication:\n");
    printMatrix(r1, c2, result);

    return 0;
}
```

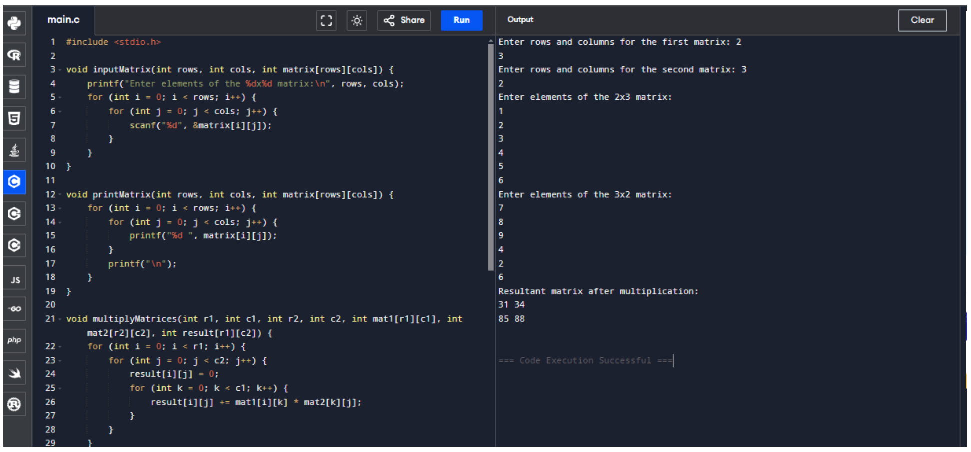
5 Complexity Analysis

- **Time Complexity:** $O(m \times n \times p)$, where m, n, p are the dimensions of the matrices.
- **Space Complexity:** $O(m \times p)$, for the result matrix.

6 Examples and Edge Cases

Matrix 1	Matrix 2	Result	Comments
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}$	$\begin{bmatrix}19 & 22\\43 & 50\end{bmatrix}$	Valid
$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$	$\begin{bmatrix}7\\8\\9\end{bmatrix}$	$\begin{bmatrix}50\\122\end{bmatrix}$	Valid
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}5\\6\end{bmatrix}$	-	Invalid dimensions

7 Output



```c
#include <stdio.h>

void inputMatrix(int rows, int cols, int matrix[rows][cols]) {
    printf("Enter elements of the %dx%d matrix:\n", rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
}

void printMatrix(int rows, int cols, int matrix[rows][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

void multiplyMatrices(int r1, int c1, int r2, int c2, int mat1[r1][c1], int
    mat2[r2][c2], int result[r1][c2]) {
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < c1; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
```

```
Enter rows and columns for the first matrix: 2
3
Enter rows and columns for the second matrix: 3
2
Enter elements of the 2x3 matrix:
1
2
3
4
5
6
Enter elements of the 3x2 matrix:
7
8
9
4
2
6
Resultant matrix after multiplication:
31 34
85 88


=== Code Execution Successful ===
```

Figure 1: Program Output Screenshot

8 Conclusion

Matrix multiplication forms the foundation of many computational algorithms. This implementation efficiently computes the product of two matrices, leveraging nested loops for element-wise operations. The approach ensures compatibility of dimensions before performing operations, highlighting the importance of input validation in matrix computations.