Name: Shaan Agarwal  
UID: 2021300001  
Branch: Computer Engineering  
Batch: A  
Subject: Design And Analysis Of Algorithms (DAA)  
Experiment No.: 3

Aim: Experiment On Divide and Conquer (Strassen’s Multiplication Method)

Algorithm:

Algorithm Strass(n, x, y, z)

begin   
If n = threshold then compute  
C = x \* y is a conventional matrix.  
Else  
Partition a into four sub matrices a00, a01, a10, a11.  
Partition b into four sub matrices b00, b01, b10, b11.  
Strass ( n/2, a00 + a11, b00 + b11, d1)  
Strass ( n/2, a10 + a11, b00, d2)  
Strass ( n/2, a00, b01 – b11, d3)  
Strass ( n/2, a11, b10 – b00, d4)  
Strass ( n/2, a00 + a01, b11, d5)  
Strass (n/2, a10 – a00, b00 + b11, d6)  
Strass (n/2, a01 – a11, b10 + b11, d7)

C = d1+d4-d5+d7 d3+d5

d2+d4 d1+d3-d2-d6   
end if  
 return (C)  
end.

Program:

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> matrixMultiplication(vector<vector<int>> matrix1, vector<vector<int>> matrix2) {

    int rows1 = matrix1.size();

    int cols1 = matrix1[0].size();

    int rows2 = matrix2.size();

    int cols2 = matrix2[0].size();

    // Multiply matrices and store result in resultMatrix

    vector<vector<int>> resultMatrix(rows1, vector<int>(cols2, 0));

    for (int i = 0; i < rows1; ++i) {

        for (int j = 0; j < cols2; ++j) {

            for (int k = 0; k < cols1; ++k) {

                resultMatrix[i][j] += matrix1[i][k] \* matrix2[k][j];

            }

        }

    }

    return resultMatrix;

}

// Function to perform matrix addition

vector<vector<int>> add(vector<vector<int>> A, vector<vector<int>> B)

{

    int n = A.size();

    vector<vector<int>> C(n, vector<int>(n));

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            C[i][j] = A[i][j] + B[i][j];

        }

    }

    return C;

}

// Function to perform matrix subtraction

vector<vector<int>> subtract(vector<vector<int>> A, vector<vector<int>> B)

{

    int n = A.size();

    vector<vector<int>> C(n, vector<int>(n));

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            C[i][j] = A[i][j] - B[i][j];

        }

    }

    return C;

}

// Function to perform Strassen's matrix multiplication

vector<vector<int>> strassen(vector<vector<int>> A, vector<vector<int>> B)

{

    int n = A.size();

    vector<vector<int>> C(n, vector<int>(n));

    // Base case

    if (n == 1)

    {

        C[0][0] = A[0][0] \* B[0][0];

        return C;

    }

    // Divide the matrices into submatrices

    int m = n / 2;

    vector<vector<int>> A11(m, vector<int>(m));

    vector<vector<int>> A12(m, vector<int>(m));

    vector<vector<int>> A21(m, vector<int>(m));

    vector<vector<int>> A22(m, vector<int>(m));

    vector<vector<int>> B11(m, vector<int>(m));

    vector<vector<int>> B12(m, vector<int>(m));

    vector<vector<int>> B21(m, vector<int>(m));

    vector<vector<int>> B22(m, vector<int>(m));

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < m; j++)

        {

            A11[i][j] = A[i][j];

            A12[i][j] = A[i][j + m];

            A21[i][j] = A[i + m][j];

            A22[i][j] = A[i + m][j + m];

            B11[i][j] = B[i][j];

            B12[i][j] = B[i][j + m];

            B21[i][j] = B[i + m][j];

            B22[i][j] = B[i + m][j + m];

        }

    }

    // Compute the seven products of submatrices

    vector<vector<int>> P1 = strassen(A11, subtract(B12, B22));

    vector<vector<int>> P2 = strassen(add(A11, A12), B22);

    vector<vector<int>> P3 = strassen(add(A21, A22), B11);

    vector<vector<int>> P4 = strassen(A22, subtract(B21, B11));

    vector<vector<int>> P5 = strassen(add(A11, A22), add(B11, B22));

    vector<vector<int>> P6 = strassen(subtract(A12, A22), add(B21, B22));

    vector<vector<int>> P7 = strassen(subtract(A11, A21), add(B11, B12));

    // Compute the resulting submatrices of the product matrix C

    vector<vector<int>> C11 = add(subtract(add(P5, P4), P2), P6);

    vector<vector<int>> C12 = add(P1, P2);

    vector<vector<int>> C21 = add(P3, P4);

    vector<vector<int>> C22 = subtract(subtract(add(P5, P1), P3), P7);

    for (int i = 0; i < m; i++)

    {

        for (int j = 0; j < m; j++)

            {

                C[i][j] = C11[i][j];

                C[i][j + m] = C12[i][j];

                C[i + m][j] = C21[i][j];

                C[i + m][j + m] = C22[i][j];

            }

    }

    return C;

}

// Function to print a matrix

void printMatrix(vector<vector<int>> A)

{

    int n = A.size();

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++) {

            cout << left<<setw(4)<<A[i][j] << " ";

        }

        cout<<endl;

    }

    cout << endl;

}

// Main Program

int main()

{

    vector<vector<int>> A = {{5,7,9,10}, {2,3,3,8}, {8,10,2,3}, {3,3,4,8}};

    vector<vector<int>> B = {{3,10,12,18}, {12,1,4,9}, {9,10,12,2}, {3,12,4,10}};

    time\_t start, end;

    time(&start);

    ios\_base::sync\_with\_stdio(false);

    vector<vector<int>> C = strassen(A, B);

    time(&end);

    vector<vector<int>> D = matrixMultiplication(A,B);

    cout << "Matrix A:" << endl;

    printMatrix(A);

    cout << "Matrix B:" << endl;

    printMatrix(B);

    cout << "Matrix C:" << endl;

    printMatrix(C);

    cout << "After normal mutliplication:" << endl;

    printMatrix(D);

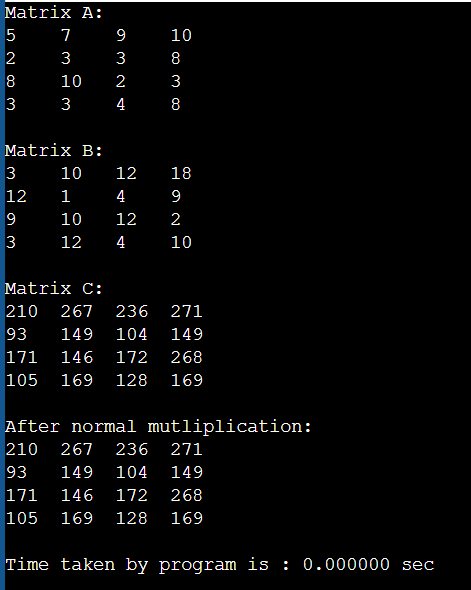
    double time\_taken = double(end - start);

    cout << "Time taken by program is : " << fixed << time\_taken << setprecision(5);

    cout << " sec " << endl;

    return 0;

}

Output  
  


Conclusion: After performing the above experiment, I have understood the concept of Strassen’s Matrix Multiplication and have applied to same to a C++ Program.