

**Ahsania Mission University of Science & Technology**

**Lab Report**

**Lab No:** 06

**Course Code:** CSE2202

**Course Title:** Computer Algorithm Sessional.

**Submitted By:**

Md. Mushfikul Islam

ID: 1012320005101011

1st Batch, 2nd Year, 2nd Semester

Department of Computer science and Engineering,

Ahsania Mission University of Science & Technology

**Submitted To:**

Md. Fahim Faisal

Lecturer,

Department of Computer science and Engineering,

Ahsania Mission University of Science & Technology

**Task No.:** 01

**Problem Statement:** Matrix Multiplication Using Naive Approach.

**Source Code:**

#include<iostream>

using namespace std;

int main()

{

int n;

cout<<"Enter Square Matrix length n\*n: ";

cin>>n;

int A[n][n],B[n][n],C[n][n];

int i=0, j=0, k=0;

cout<<"Enter The Matrix A:"<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cin>>A[i][j];

}

}

cout<<"Enter The Matrix B:"<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cin>>B[i][j];

}

}

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

C[i][j]=0;

for(k=0;k<n;k++)

{

C[i][j]+=A[i][k]\*B[k][j];

}

}

}

cout<<"The result matrix C:"<<endl;

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

cout<<C[i][j]<<" ";

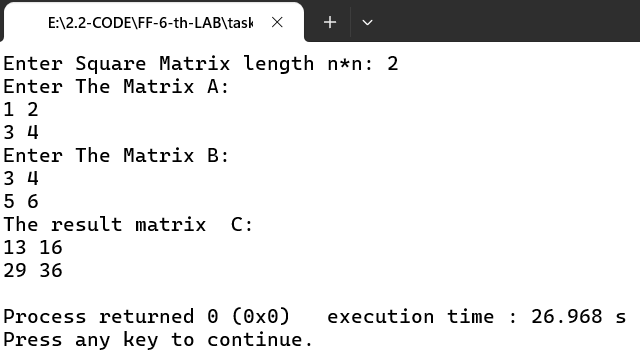
}

cout<<endl;

}

}

**Output:**

****

**Task No.:** 02

**Problem Statement:** Matrix Multiplication Using Divide and Conquer Approach.

**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

typedef vector<vector<int>> Matrix;

// Matrix addition

Matrix add(const Matrix &A, const Matrix &B) {

int n = A.size();

Matrix 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;

}

// Matrix subtraction

Matrix subtract(const Matrix &A, const Matrix &B) {

int n = A.size();

Matrix 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;

}

// Recursive matrix multiplication (divide and conquer)

Matrix multiply(const Matrix &A, const Matrix &B) {

int n = A.size();

Matrix C(n, vector<int>(n, 0));

if (n == 1) {

C[0][0] = A[0][0] \* B[0][0];

} else {

int k = n / 2;

Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k)), A21(k, vector<int>(k)), A22(k, vector<int>(k));

Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k)), B21(k, vector<int>(k)), B22(k, vector<int>(k));

for (int i = 0; i < k; i++)

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

A11[i][j] = A[i][j];

A12[i][j] = A[i][j + k];

A21[i][j] = A[i + k][j];

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

B11[i][j] = B[i][j];

B12[i][j] = B[i][j + k];

B21[i][j] = B[i + k][j];

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

}

Matrix C11 = add(multiply(A11, B11), multiply(A12, B21));

Matrix C12 = add(multiply(A11, B12), multiply(A12, B22));

Matrix C21 = add(multiply(A21, B11), multiply(A22, B21));

Matrix C22 = add(multiply(A21, B12), multiply(A22, B22));

for (int i = 0; i < k; i++)

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

C[i][j] = C11[i][j];

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

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

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

}

}

return C;

}

// Input a square matrix

Matrix inputMatrix(int size, char name) {

Matrix mat(size, vector<int>(size));

cout << "Enter elements of matrix " << name << " (" << size << "x" << size << "):\n";

for (int i = 0; i < size; ++i)

for (int j = 0; j < size; ++j)

cin >> mat[i][j];

return mat;

}

// Print a matrix

void printMatrix(const Matrix &mat) {

int size = mat.size();

for (int i = 0; i < size; ++i) {

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

cout << mat[i][j] << " ";

}

cout << endl;

}

}

// Main function

int main() {

int n;

cout << "Enter matrix size (must be power of 2): ";

cin >> n;

if ((n & (n - 1)) != 0 || n <= 0) {

cout << "Size must be a positive power of 2." << endl;

return 1;

}

Matrix A = inputMatrix(n, 'A');

Matrix B = inputMatrix(n, 'B');

Matrix C = multiply(A, B);

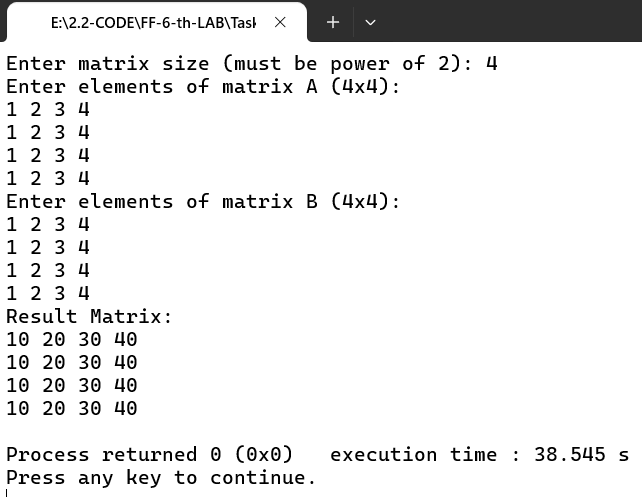
cout << "Result Matrix:" << endl;

printMatrix(C);

return 0;

}

**Output:**

****

**Task No.:** 03

**Problem Statement:** Matrix Multiplication Using Strassen's Algorithm

**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

typedef vector<vector<int>> Matrix;

Matrix add(const Matrix &A, const Matrix &B)

{

int n = A.size();

Matrix result(n, vector<int>(n, 0));

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

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

return result;

}

Matrix subtract(const Matrix &A, const Matrix &B)

{

int n = A.size();

Matrix result(n, vector<int>(n, 0));

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

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

return result;

}

void split(const Matrix &A, Matrix &A11, Matrix &A12, Matrix &A21, Matrix &A22)

{

int n = A.size() / 2;

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

A11[i][j] = A[i][j];

A12[i][j] = A[i][j + n];

A21[i][j] = A[i + n][j];

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

}

}

}

Matrix combine(const Matrix &C11, const Matrix &C12, const Matrix &C21, const Matrix &C22)

{

int n = C11.size();

Matrix C(2 \* n, vector<int>(2 \* n));

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

C[i][j] = C11[i][j];

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

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

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

}

}

return C;

}

Matrix strassen(const Matrix &A, const Matrix &B)

{

int n = A.size();

if (n == 1)

{

return Matrix{{A[0][0] \* B[0][0]}};

}

int newSize = n / 2;

Matrix A11(newSize, vector<int>(newSize));

Matrix A12(newSize, vector<int>(newSize));

Matrix A21(newSize, vector<int>(newSize));

Matrix A22(newSize, vector<int>(newSize));

Matrix B11(newSize, vector<int>(newSize));

Matrix B12(newSize, vector<int>(newSize));

Matrix B21(newSize, vector<int>(newSize));

Matrix B22(newSize, vector<int>(newSize));

split(A, A11, A12, A21, A22);

split(B, B11, B12, B21, B22);

Matrix P1 = strassen(A11, subtract(B12, B22));

Matrix P2 = strassen(add(A11, A12), B22);

Matrix P3 = strassen(add(A21, A22), B11);

Matrix P4 = strassen(A22, subtract(B21, B11));

Matrix P5 = strassen(add(A11, A22), add(B11, B22));

Matrix P6 = strassen(subtract(A12, A22), add(B21, B22));

Matrix P7 = strassen(subtract(A11, A21), add(B11, B12));

Matrix C11 = add(subtract(add(P5, P4), P2), P6);

Matrix C12 = add(P1, P2);

Matrix C21 = add(P3, P4);

Matrix C22 = subtract(subtract(add(P1, P5), P3), P7);

return combine(C11, C12, C21, C22);

}

void printMatrix(const Matrix &matrix)

{

for (const auto &row : matrix)

{

for (int val : row)

cout << val << " ";

cout << endl;

}

}

Matrix inputMatrix(int size, char name)

{

Matrix mat(size, vector<int>(size));

cout << "Enter elements of matrix " << name << " (" << size << "x" << size << "):\n";

for (int i = 0; i < size; ++i)

for (int j = 0; j < size; ++j)

{

//cout << name << "[" << i << "][" << j << "]: ";

cin >> mat[i][j];

}

return mat;

}

int main()

{

int n;

cout << "Enter matrix size (must be power of 2): ";

cin >> n;

if ((n & (n - 1)) != 0 || n <= 0)

{

cout << "Size must be a positive power of 2." << endl;

return 1;

}

Matrix A = inputMatrix(n, 'A');

Matrix B = inputMatrix(n, 'B');

Matrix C = strassen(A, B);

cout << "Result Matrix:" << endl;

printMatrix(C);

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated