Министерство транспорта РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Российский университет транспорта» (МИИТ)

Лабораторная работа №6

«Умножение матриц»

по дисциплине

«Методы программирования»

Выполнил:

студент группы ТКИ-311,

Гаврилин-Кирьянов И.А.

Проверил:

к.т.н., доц. Логинова Л. Н,

к.т.н., доц. Сафронов А. И.

Москва - 2022 г.

**Цель работы:**

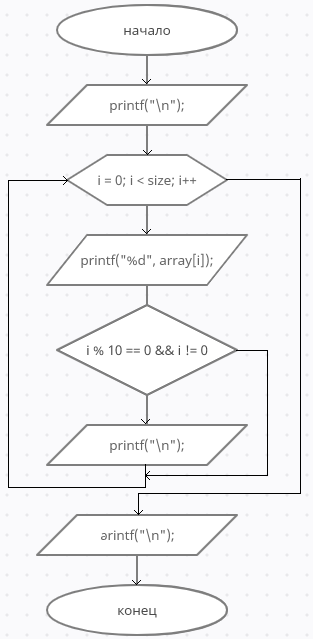
Изучение алгоритмов умножения матриц.

**Задача:**

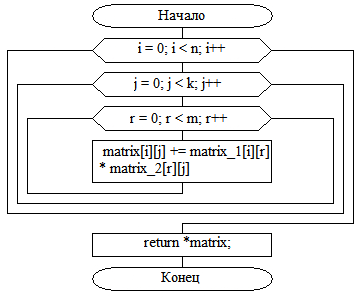
Реализовать алгоритмы, выполняющие умножение матриц на языке С

**Блок-схемы:**

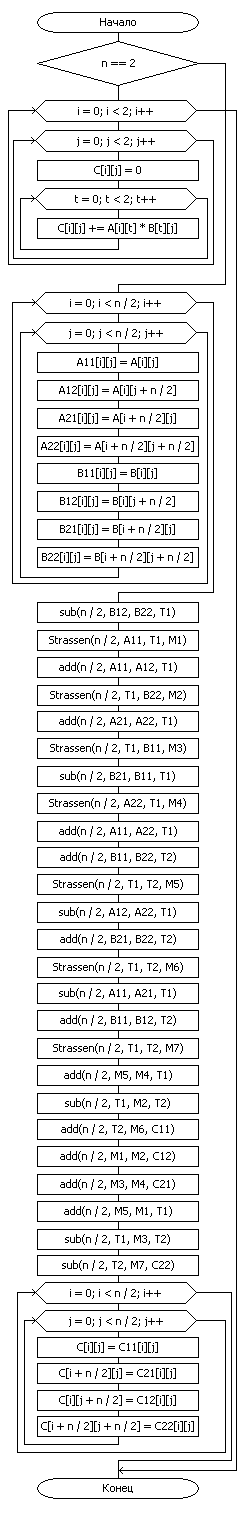
Функция вывода массива:

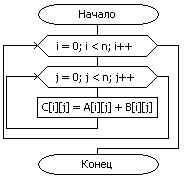


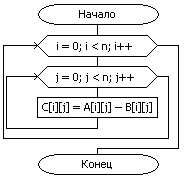
Функция перемножения грубой силой:



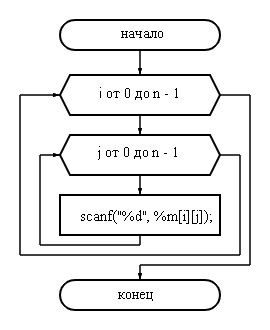
Функция перемножения матриц методом Штрассена:

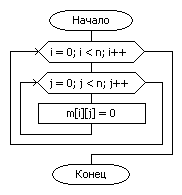


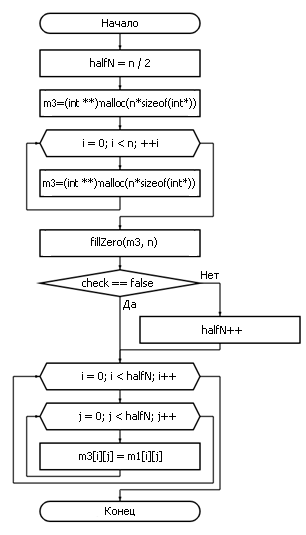


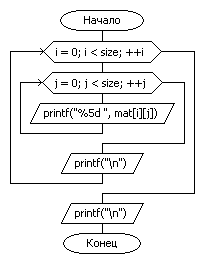


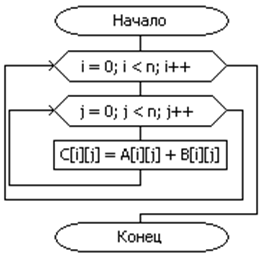
Функция перемножения матриц методом «Разделяй и властвуй»:

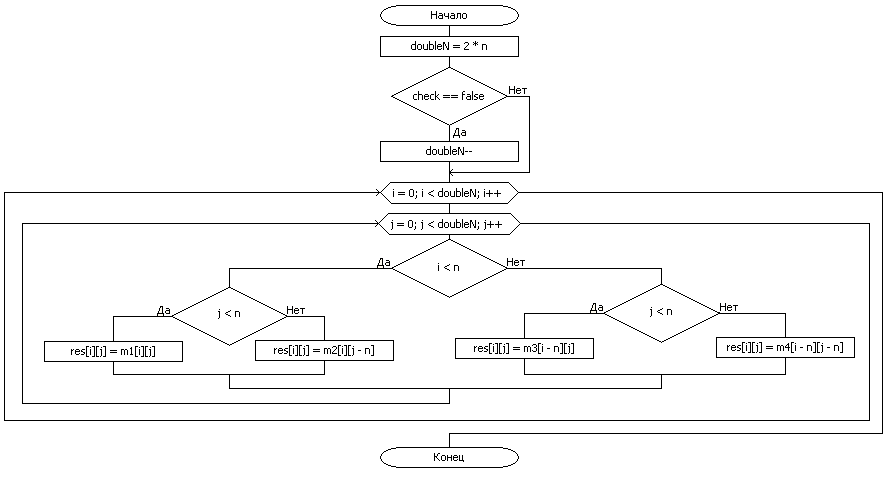


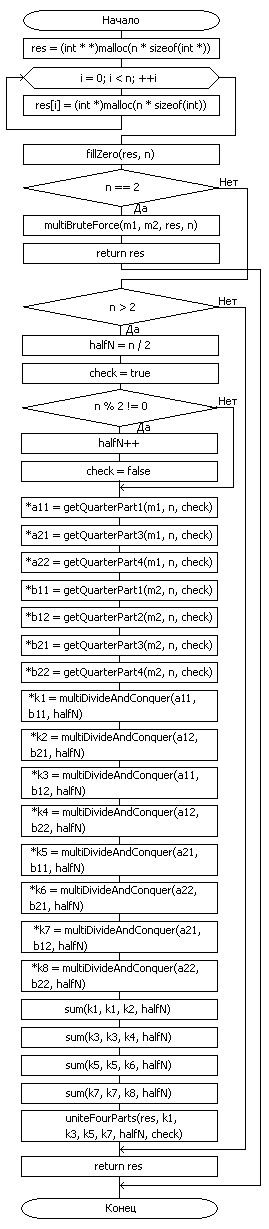












**Код программы:**

1. Brute Force

#include <stdio.h>

int n;

int \*mult(int matrix\_1[n][n], int matrix\_2[n][n], int matrix[n][n], int n);

int main()

{

n = 8;

int matrix\_1[n][n];

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

{

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

{

scanf("%d", &matrix\_1[i][j]);

}

}

int matrix\_2[n][n];

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

{

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

{

scanf("%d", &matrix\_2[i][j]);

}

}

printf("\nTheir multiplication is:\n");

int matrix[n][n];

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

{

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

{

matrix[i][j] = 0;

}

}

mult(matrix\_1, matrix\_2, matrix, n);

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

{

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

{

printf("%5d ", matrix[i][j]);

}

printf("\n");

}

}

int \*mult(int matrix\_1[n][n], int matrix\_2[n][n], int matrix[n][n], int n)

{

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

{

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

{

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

{

matrix[i][j] += matrix\_1[i][r] \* matrix\_2[r][j];

}

}

}

return \*matrix;

}

1. Метод Штрассена

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 8

void add(int n, int A[][N], int B[][N], int C[][N])

{

int i, j;

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

{

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

{

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

}

}

}

void sub(int n, int A[][N], int B[][N], int C[][N])

{

int i, j;

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

{

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

{

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

}

}

}

void Strassen(int n, int A[][N], int B[][N], int C[][N])

{

int A11[N][N], A12[N][N], A21[N][N], A22[N][N];

int B11[N][N], B12[N][N], B21[N][N], B22[N][N];

int C11[N][N], C12[N][N], C21[N][N], C22[N][N];

int M1[N][N], M2[N][N], M3[N][N], M4[N][N], M5[N][N], M6[N][N], M7[N][N];

int T1[N][N], T2[N][N];

int i, j;

if(n == 2)

{

for(i = 0; i < 2; i++)

{

for(j = 0; j < 2; j++)

{

C[i][j] = 0;

for(int t = 0; t < 2; t++)

{

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

}

}

}

return;

}

for(i = 0; i < n / 2; i++)

{

for(j = 0; j < n / 2 ; j++)

{

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

A12[i][j] = A[i][j + n / 2];

A21[i][j] = A[i + n / 2][j];

A22[i][j] = A[i + n / 2][j + n / 2];

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

B12[i][j] = B[i][j + n / 2];

B21[i][j] = B[i + n / 2][j];

B22[i][j] = B[i + n / 2][j + n / 2];

}

}

//M1 = A11(B12 - B22)

sub(n / 2, B12, B22, T1);

Strassen(n / 2, A11, T1, M1);

//M2 = (A11 + A12)B22

add(n / 2, A11, A12, T1);

Strassen(n / 2, T1, B22, M2);

//M3 = (A21 + A22)B11

add(n / 2, A21, A22, T1);

Strassen(n / 2, T1, B11, M3);

//M4 = A22(B21 - B11)

sub(n / 2, B21, B11, T1);

Strassen(n / 2, A22, T1, M4);

//M5 = (A11 + A22)(B11 + B22)

add(n / 2, A11, A22, T1);

add(n / 2, B11, B22, T2);

Strassen(n / 2, T1, T2, M5);

//M6 = (A12 - A22)(B21 + B22)

sub(n / 2, A12, A22, T1);

add(n / 2, B21, B22, T2);

Strassen(n / 2, T1, T2, M6);

//M7 = (A11 - A21)(B11 + B12)

sub(n / 2, A11, A21, T1);

add(n / 2, B11, B12, T2);

Strassen(n / 2, T1, T2, M7);

//C11 = M5 + M4 - M2 + M6

add(n / 2, M5, M4, T1);

sub(n / 2, T1, M2, T2);

add(n / 2, T2, M6, C11);

//C12 = M1 + M2

add(n / 2, M1, M2, C12);

//C21 = M3 + M4

add(n / 2, M3, M4, C21);

//C22 = M5 + M1 - M3 - M7

add(n / 2, M5, M1, T1);

sub(n / 2, T1, M3, T2);

sub(n / 2, T2, M7, C22);

for(i = 0; i < n / 2; i++)

{

for(j = 0; j < n / 2; j++)

{

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

C[i + n / 2][j] = C21[i][j];

C[i][j + n / 2] = C12[i][j];

C[i + n / 2][j + n / 2] = C22[i][j];

}

}

}

int main()

{

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

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

{

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

{

scanf("%d", &A[i][j]);

}

}

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

{

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

{

scanf("%d", &B[i][j]);

}

}

Strassen(N, A, B, C);

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

{

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

{

printf("%5d\t",C[i][j]);

}

printf("\n");

}

printf("\n");

printf("Multiplication is over\n");

}

1. Метод «Управляй и властвуй»

#include <iostream>

#include <iomanip>

using namespace std;

void fillMatrix(int\*\* m, int n)

{

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

{

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

{

scanf("%d", &m[i][j]);

}

}

}

void fillZero(int\*\* m, int n)

{

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

{

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

{

m[i][j] = 0;

}

}

}

int\*\* getQuarterPart1(int\*\* m1, int n, bool check)

{

int halfN = n / 2;

int\*\* m3;

m3 = (int\*\*)malloc(n \* sizeof(int\*));

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

m3[i] = (int\*)malloc(n \* sizeof(int));

fillZero(m3, n);

if (check == false)

{

halfN++;

}

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

{

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

{

m3[i][j] = m1[i][j];

}

}

return m3;

}

int\*\* getQuarterPart2(int\*\* m1, int n, bool check)

{

int halfN = n / 2;

int\*\* m3;

m3 = (int\*\*)malloc(n \* sizeof(int\*));

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

m3[i] = (int\*)malloc(n \* sizeof(int));

fillZero(m3, n);

if (check == false)

{

halfN++;

}

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

{

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

{

m3[i][j] = m1[i][j + halfN];

}

}

return m3;

}

int\*\* getQuarterPart3(int\*\* m1, int n, bool check)

{

int halfN = n / 2;

int\*\* m3;

m3 = (int\*\*)malloc(n \* sizeof(int\*));

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

m3[i] = (int\*)malloc(n \* sizeof(int));

fillZero(m3, n);

if (check == false)

{

halfN++;

}

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

{

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

{

m3[i][j] = m1[i + halfN][j];

}

}

return m3;

}

int\*\* getQuarterPart4(int\*\* m1, int n, bool check)

{

int halfN = n / 2;

int\*\* m3;

m3 = (int\*\*)malloc(n \* sizeof(int\*));

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

m3[i] = (int\*)malloc(n \* sizeof(int));

fillZero(m3, n);

if (check == false)

{

halfN++;

}

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

{

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

{

m3[i][j] = m1[i + halfN][j + halfN];

}

}

return m3;

}

void print(int\*\* mat, unsigned size)

{

unsigned i, j;

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

{

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

{

printf("%5d ", mat[i][j]);

}

printf("\n");

}

printf("\n");

}

void multiBruteForce(int\*\* m1, int\*\* m2, int\*\* m3, int n)

{

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

{

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

{

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

{

m3[i][j] += m1[i][str] \* m2[str][j];

}

}

}

}

void sum(int\*\* res, int\*\* m1, int\*\* m2, int n)

{

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

{

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

{

res[i][j] = m1[i][j] + m2[i][j];

}

}

}

void uniteFourParts(int\*\* res, int\*\* m1, int\*\* m2, int\*\* m3, int\*\* m4, int n, bool check)

{

int doubleN = 2 \* n;

if (check == false)

{

doubleN--;

}

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

{

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

{

if (i < n)

{

if (j < n)

{

res[i][j] = m1[i][j];

}

else

{

res[i][j] = m2[i][j - n];

}

}

else

{

if (j < n)

{

res[i][j] = m3[i - n][j];

}

else

{

res[i][j] = m4[i - n][j - n];

}

}

}

}

}

int\*\* multiDivideAndConquer(int\*\* m1, int\*\* m2, int n)

{

int\*\* res;

res = (int\*\*)malloc(n \* sizeof(int\*));

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

res[i] = (int\*)malloc(n \* sizeof(int));

fillZero(res, n);

if (n == 2)

{

multiBruteForce(m1, m2, res, n);

return res;

}

if (n > 2)

{

int halfN = n / 2;

bool check = true;

if (n % 2 != 0)

{

halfN++;

check = false;

}

int\*\* a11 = getQuarterPart1(m1, n, check);

int\*\* a12 = getQuarterPart2(m1, n, check);

int\*\* a21 = getQuarterPart3(m1, n, check);

int\*\* a22 = getQuarterPart4(m1, n, check);

int\*\* b11 = getQuarterPart1(m2, n, check);

int\*\* b12 = getQuarterPart2(m2, n, check);

int\*\* b21 = getQuarterPart3(m2, n, check);

int\*\* b22 = getQuarterPart4(m2, n, check);

int\*\* k1 = multiDivideAndConquer(a11, b11, halfN);

int\*\* k2 = multiDivideAndConquer(a12, b21, halfN);

int\*\* k3 = multiDivideAndConquer(a11, b12, halfN);

int\*\* k4 = multiDivideAndConquer(a12, b22, halfN);

int\*\* k5 = multiDivideAndConquer(a21, b11, halfN);

int\*\* k6 = multiDivideAndConquer(a22, b21, halfN);

int\*\* k7 = multiDivideAndConquer(a21, b12, halfN);

int\*\* k8 = multiDivideAndConquer(a22, b22, halfN);

sum(k1, k1, k2, halfN);

sum(k3, k3, k4, halfN);

sum(k5, k5, k6, halfN);

sum(k7, k7, k8, halfN);

uniteFourParts(res, k1, k3, k5, k7, halfN, check);

}

return res;

}

int main()

{

int sizeMatrix = 8;

int\*\* m1;

int\*\* m2;

int\*\* m3;

m1 = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

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

m1[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

m2 = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

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

m2[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

m3 = (int\*\*)malloc(sizeMatrix \* sizeof(int\*));

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

m3[i] = (int\*)malloc(sizeMatrix \* sizeof(int));

fillZero(m3, sizeMatrix);

fillMatrix(m1, sizeMatrix);

print(m1,sizeMatrix);

fillMatrix(m2, sizeMatrix);

print(m2,sizeMatrix);

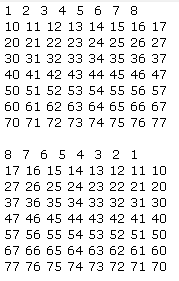
m3 = multiDivideAndConquer(m1, m2, sizeMatrix);

print(m3,sizeMatrix);

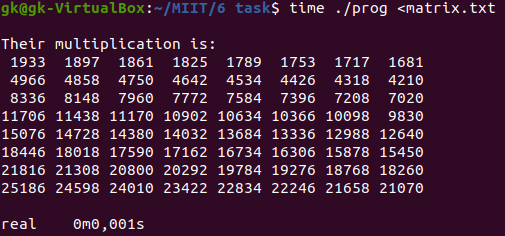
}

**Результаты программы:**

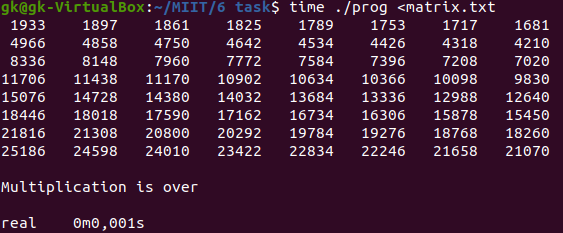
Исходные матрицы имеют вид:



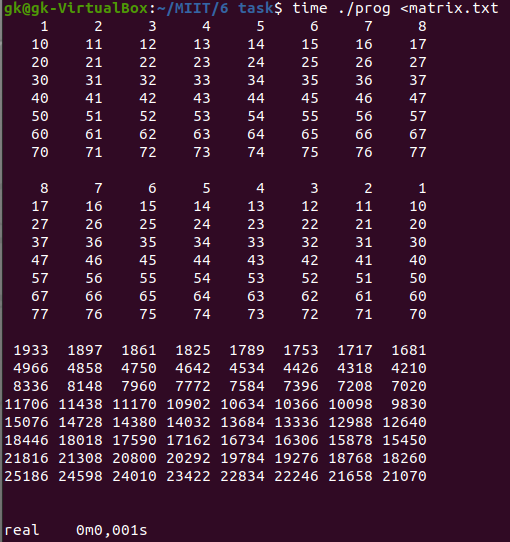
1. Brute Force



1. Метод Штрассена



1. Метод «Разделяй и властвуй»



**Вывод:**

В ходе работы были реализованы алгоритмы умножения матриц, а именно стандартный алгоритм умножения, метод «разделяй и властвуй» и алгоритм Штрассена. Все методы работают корректно. Алгоритм «разделяй и властвуй» и алгоритм Штрассена работают эффективнее перемножения матриц напрямую. Но используют гораздо больше памяти, чем полный перебор, поэтому при работе с матрицами очень больших размерностей могут возникнуть трудности.