Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ВС

РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

по дисциплине «ТЕОРИЯ СЛОЖНОСТИ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ И СТРУКТУР»

Вариант 1

Выполнил: студент гр. ИП-712

Алексеев С.В.

Проверил: Разинкина. Т.Э.

Новосибирск 2019 г.

Оглавление

[Задание 2](#_Toc27758464)

[Описание алгоритма 2](#_Toc27758465)

[Листинг программы 3](#_Toc27758466)

[Вывод 7](#_Toc27758467)

# Задание

**Вариант1.** Написать 2 программы, вычисляющие произведение двух матриц A[i,j]=(-1)^(i+j), B[i,j]=i+j,i,j=1…100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

# Описание алгоритма

Мной был использован классический алгоритм умножения матриц с трудоёмкостью порядка сn3 (функция simplMul(int[][] A, int[][] B)) и алгоритм Штрассена, с трудоёмкостью cn2.81 (функция StrassMul(int[][] A, int[][] B)), здесь с – константа, n – порядок матрицы. Язык реализации: java.

Классический алгоритм перемножает по очереди i-й элемент строки матрицы A на i-й элемент стобца матрицы B, результаты складываются, получается очередной элемент результирующей матрицы res.

Алгоритм Штрассена предполагает разбиение каждой исходной матрицы на 4 равные части(4 новых двумерных массива). Т.о. порядок матрицы может быть только степенью двойки. Получаем 8 массивов: A1, A2, A3, A4, B1, B2, B3, B4. После этого с помощью рекурсивных вызовов StrassMul заполняются ещё 11 двумерных массивов:

*M1 = (A2 – A4)(B3 + B4)*

*M2 = (A1 + A4)(B1 + B4)*

*M3 = (A1 – A3)(B1 + B2)*

*M4 = (A1 + A2)B4*

*M5 = A1(B2 – B4)*

*M6 = A4 (B3 – B1)*

*M7 = (A3 + A4)B1*

*C1 = M1 +M2 – M4 + M6*

*C2 = M4 + M5*

*C3 = M6 +M7*

*C4 = M2 – M3 + M5 – M7*

После этого из С1, С2, С3, С4 собирается результирующая матрица.

Рекурсивные вызовы StrassMul доходят до той глубины, где размеры матриц становятся 1х1, там и происходит умножение и рекурсивный возврат в вызывавшую функцию.

# Листинг программы

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

static int cntSimple = 0, cntStrassen = 0;

public static int[][] simplMul(int[][] A, int[][] B) {

int[][] res = new int[A.length][A.length];

for (int i = 0; i < A.length; i++) {

for (int j = 0; j < A.length; j++) {

for (int k = 0; k < A.length; k++) {

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

cntSimple += 1;

}

}

}

return res;

}

public static int[][] StrassMul(int[][] A, int[][] B) {

int n = A.length;

int[][] res = new int[n][n];

if (n == 1) {

cntStrassen += 1;

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

} else {

int[][] A1 = new int[n / 2][n / 2];

int[][] A2 = new int[n / 2][n / 2];

int[][] A3 = new int[n / 2][n / 2];

int[][] A4 = new int[n / 2][n / 2];

int[][] B1 = new int[n / 2][n / 2];

int[][] B2 = new int[n / 2][n / 2];

int[][] B3 = new int[n / 2][n / 2];

int[][] B4 = new int[n / 2][n / 2];

matrDivision(A, A1, 0, 0);

matrDivision(A, A2, 0, n / 2);

matrDivision(A, A3, n / 2, 0);

matrDivision(A, A4, n / 2, n / 2);

matrDivision(B, B1, 0, 0);

matrDivision(B, B2, 0, n / 2);

matrDivision(B, B3, n / 2, 0);

matrDivision(B, B4, n / 2, n / 2);

int[][] M1 = StrassMul(subMatrices(A2, A4), addMatrices(B3, B4));

int[][] M2 = StrassMul(addMatrices(A1, A4), addMatrices(B1, B4));

int[][] M3 = StrassMul(subMatrices(A1, A3), addMatrices(B1, B2));

int[][] M4 = StrassMul(addMatrices(A1, A2), B4);

int[][] M5 = StrassMul(A1, subMatrices(B2, B4));

int[][] M6 = StrassMul(A4, subMatrices(B3, B1));

int[][] M7 = StrassMul(addMatrices(A3, A4), B1);

int[][] C11 = addMatrices(subMatrices(addMatrices(M2, M6), M4), M1);

int[][] C12 = addMatrices(M4, M5);

int[][] C21 = addMatrices(M6, M7);

int[][] C22 = subMatrices(subMatrices(addMatrices(M2, M5), M7), M3);

assembleMatrices(C11, res, 0, 0);

assembleMatrices(C12, res, 0, n / 2);

assembleMatrices(C21, res, n / 2, 0);

assembleMatrices(C22, res, n / 2, n / 2);

}

return res;

}

public static void main(String[] args) throws NumberFormatException, IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Введите размер матрицы:");

int order = Integer.parseInt(br.readLine());

int[][] A = new int[order][order];

int[][] B = new int[order][order];

int[][] res = new int[order][order];

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

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

A[i][j] = (int) Math.pow(-1, (i + j));

}

}

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

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

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

}

}

res = StrassMul(A, B);

printMatrix(res);

System.out.println("\nТрудоёмкость Штрассен:" + cntStrassen + "\n");

res = null;

res = simplMul(A, B);

printMatrix(res);

System.out.println("\nТрудоёмкость простого:" + cntSimple + "\n");

}

public static int[][] addMatrices(int[][] a, int[][] b) {

int n = a.length;

int[][] res = new int[n][n];

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

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

res[i][j] = a[i][j] + b[i][j];

}

}

return res;

}

public static int[][] subMatrices(int[][] a, int[][] b) {

int n = a.length;

int[][] res = new int[n][n];

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

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

res[i][j] = a[i][j] - b[i][j];

}

}

return res;

}

public static void printMatrix(int[][] a) {

int n = a.length;

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

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

System.out.print(a[i][j] + "\t");

}

System.out.println();

}

System.out.println();

}

public static void matrDivision(int[][] P, int[][] C, int iB, int jB) {

for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)

for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)

C[i1][j1] = P[i2][j2];

}

public static void assembleMatrices(int[][] C, int[][] P, int iB, int jB) {

for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)

for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)

P[i2][j2] = C[i1][j1];

}}

# Вывод

Убедился в эффективности алгоритма штрассена, показавшего расчётную трудоёмкость.

