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 6](#_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: a, b, c, d, e, f, g, h. После этого с помощью рекурсивных вызовов StrassMul заполняются ещё 11 двумерных массивов:

M1 = StrassMul((a + d), (e + h));

M2 = StrassMul( (c + d), e);

M3 = StrassMul(a, (f - h));

M4 = StrassMul(d, (g - e));

M5 = StrassMul((a + b), h);

M6 = StrassMul( (c - a), (e + f));

M7 = StrassMul( (b - d), (g + h));

C11 = ( ( (M1 + M4) - M5) + M7);

C12 = (M3+ M5);

C21 = (M2 + M4);

C22 = ( ( (M1 + M3) - M2) + M6);

После этого из С11, С12, С21, С22 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+=2;

}

}

}

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

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

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

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

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

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

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

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

matrDivision(A, a, 0, 0);

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

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

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

matrDivision(B, e, 0, 0);

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

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

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

int[][] M1 = StrassMul(addMatrices(a, d), addMatrices(e, h));

int[][] M2 = StrassMul(addMatrices(c, d), e);

int[][] M3 = StrassMul(a, subMatrices(f, h));

int[][] M4 = StrassMul(d, subMatrices(g, e));

int[][] M5 = StrassMul(addMatrices(a, b), h);

int[][] M6 = StrassMul(subMatrices(c, a), addMatrices(e, f));

int[][] M7 = StrassMul(subMatrices(b, d), addMatrices(g, h));

int[][] C11 = addMatrices(subMatrices(addMatrices(M1, M4), M5), M7);

int[][] C12 = addMatrices(M3, M5);

int[][] C21 = addMatrices(M2, M4);

int[][] C22 = addMatrices(subMatrices(addMatrices(M1, M3), M2), M6);

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];

System.out.println("введите первую матрицу:");

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

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

// A[i][j] = Integer.parseInt(br.readLine());

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

}

}

System.out.println("введите вторую матрицу:");

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

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

//B[i][j] = Integer.parseInt(br.readLine());

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

}

}

System.out.println("Умножение Штрассеном:\n");

res = StrassMul(A, B);

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

res = null;

System.out.println("Умножение простое:\n");

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;

System.out.println("Результат перемножения: ");

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.

