Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)



ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

Отчет на тему:

«Умножение Матриц»

Марабян Корюн

ИУ7-55Б

Москва 2018

1. Аналитическая часть 3

1.1 Постановка задачи 3

1.2 Описание алгоритмов 3

2. Конструкторская часть 4

2.1 Разработка алгоритмов 4

3. Технологическая часть 10

3.1 Требования к программному обеспечению 10

3.2 Средства реализации 10

3.3 Листинг кода 11

4. Экспериментальная часть 16

4.1 Примеры работы 16

4.2 Постановка эксперимента 17

Заключение 21

1. Аналитическая часть

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка.

1.1 Постановка задачи.

В задачи лабораторной работы входит:

1. Изучение и реализация алгоритмов умножения матриц: стандартный и алгоритм Винограда.

2. Улучшение реализации алгоритма Винограда.

3. Теоретическая оценка алгоритмов умножения матриц.

4. Сравнение времени работы алгоритмов умножения матриц.

1.2 Описание алгоритмов

Стандартный (базовый) алгоритм умножения матриц

Для вычисления произведения двух матриц А и В каждая строка матрицы А скалярно умножается на каждый столбец матрицы В. Затем сумма данных

произведений записывается в соответствующую ячейку результирующей

матрицы.

Алгоритм Винограда

Алгоритм Винограда - более эффективный алгоритм умножения матриц, использующий предварительную обработку для уменьшения количества операций.

Так как умножение матриц представляет собой скалярное произведение строк и столбцов:

U = (u1, u2, u3, u4), W = (w1, w2, w3, w4)

U \* W = u1w1 + u2w2 + u3w3 + u4w4

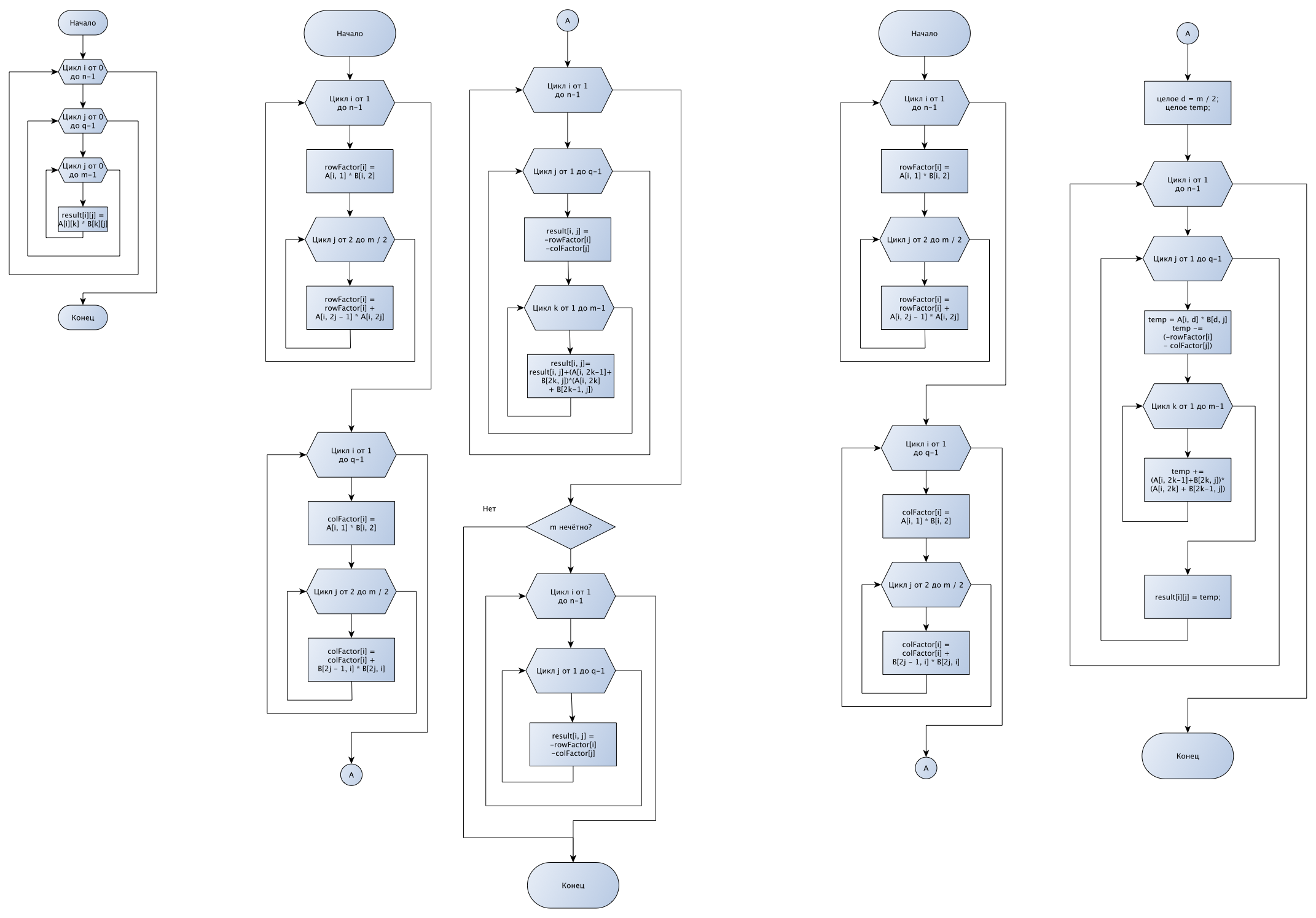
можно, представив правую часть последнего уравнения в виде:

U\*W = (u1 + w2)(u2 +w1) + (u3 +w4)(u4 +w3) – u1u2 – u3u4 – w1w2 -w3w4, предварительно рассчитать часть (u1u2 – u3u4 – w1w2 – w3w4) для каждой строки первой матрицы и для каждого столбца второй.

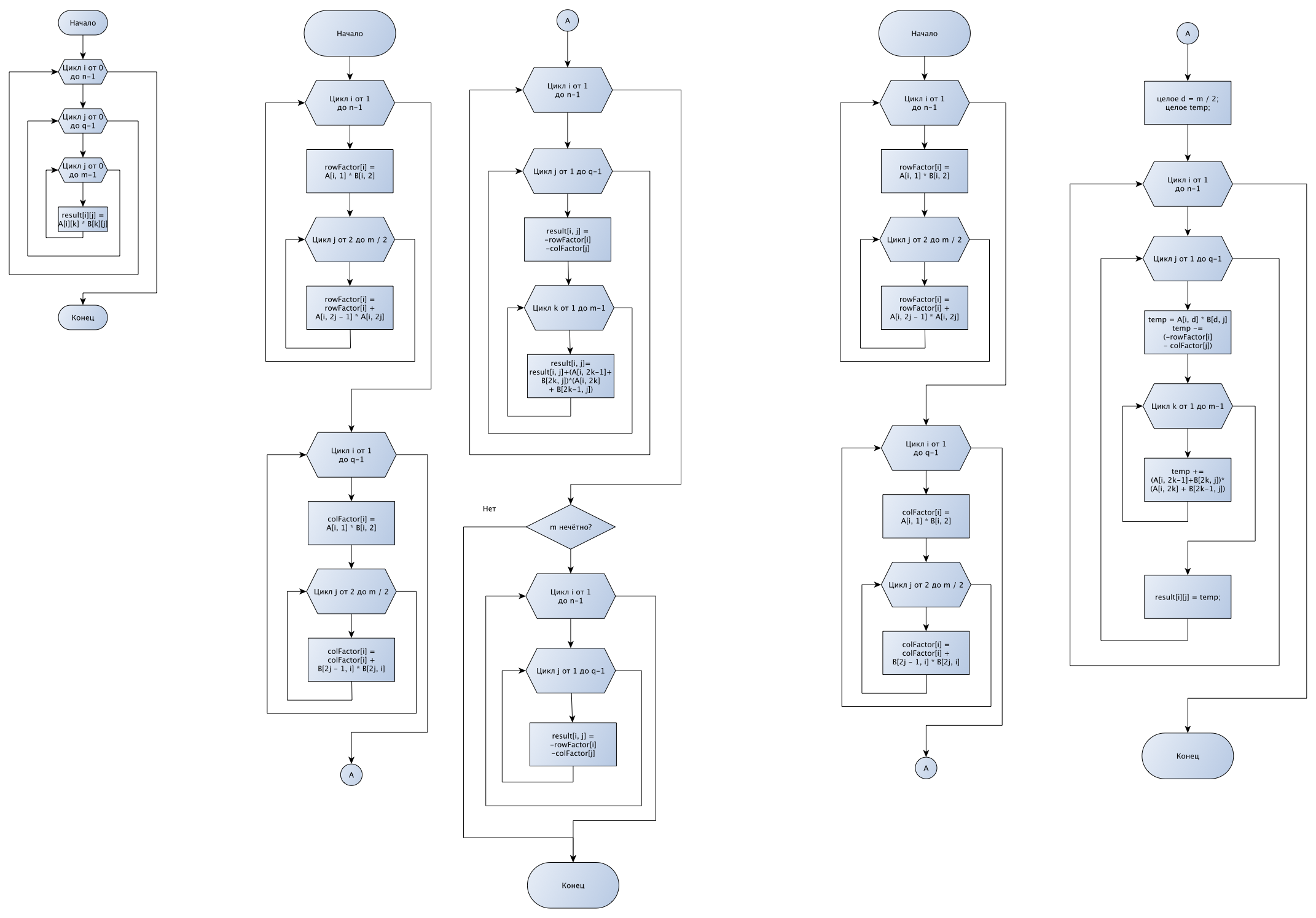
2. Конструкторская часть

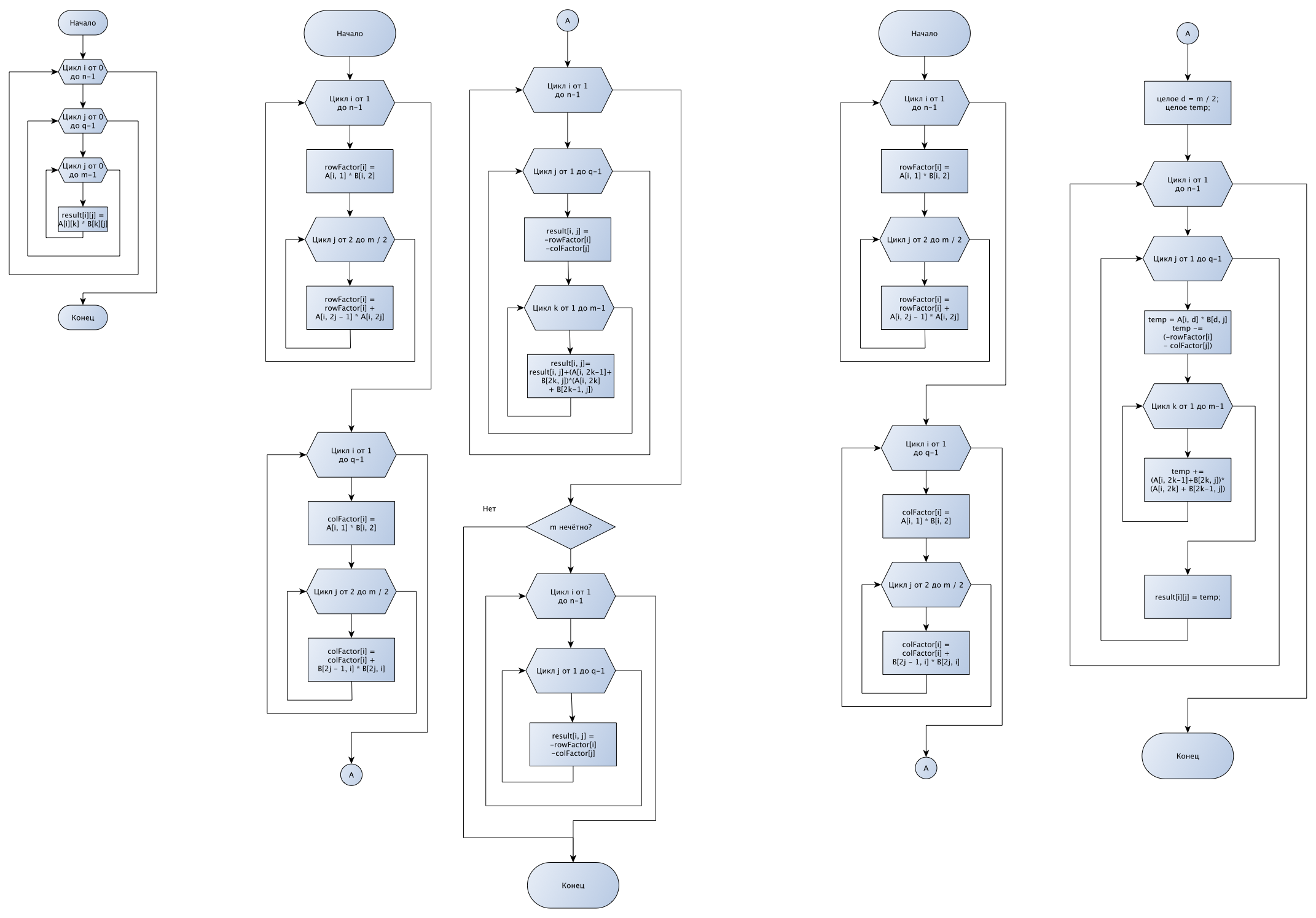
В конструкторской части описан ход разработки алгоритмов, а также сравнение рекурсивной и итеративной реализаций

2.1 Разработка алгоритмов



*Рис. 1.1 Схема базового алгоритма умножения матриц*

*Рис. 1.2 Схема алгоритма Винограда*

*Рис. 1.3 Схема улучшенного алгоритма Винограда*

2.1.1 Разработка базового алгоритма умножения матриц

Базовый алгоритм легко реализуется простым тройным циклом. В представленном ниже фрагменте кода матрица *result* была предварительно заполнена нулями.

Листинг 1. - Базовый алгоритм умножения матриц, псеводкод

1. Умнож(матр. А, целое n, целое m, матр. B, целое q, матр. result)
2. {
3. Цикл i от 0 до n-1
4. Цикл j от 0 до q-1
5. Цикл k от 0 до m-1
6. result[i][j] = result[i][j] + A[i][k] \* B[k][j];
7. }

2.1.2 Разработка алгоритма Винограда

При работе с матрицами нечётной общей размерности требуется дополнительный двойной цикл после вычисления матрицы *result*. Ожидается, что этот цикл будет ощутимо замедлять работу алгоритма

Псевдокод алгоритма представлен ниже

Листинг 2. - Алгоритм Винограда, псеводкод

1. Виноград(матр. А, целое n, целое m, матр. B, целое q, матр. result)
2. {
3. массив row\_factor[n];
5. // Вычисление множителей строк
6. Цикл i от 1 до n-1
7. rowFactor[i] = A[i, 1] \* A[i, 2]
8. Цикл j от 2 до m / 2
9. rowFactor[i] = rowFactor[i] + A[i, 2j - 1] \* A[i, 2j]
11. // Вычисление множителей столбцов
12. Цикл i от 1 до q-1
13. colFactor[i] = B[1, i] \* B[2, i]
14. Цикл j от 2 до m / 2
15. colFactor[i] = colFactor[i] + B[2j - 1, i] \* B[2j, i]
17. Цикл i от 1 до n-1
18. Цикл j от 1 до q-1
19. result[i, j] = -rowFactor[i] - colFactor[j]
20. Цикл k от 1 до m/2
21. result[i, j]=result[i, j]+(A[i, 2k-1]+B[2k, j])\*(A[i, 2k] + B[2k-1, j])

24. Если m % 2 != 0
25. Цикл i от 1 до n-1
26. Цикл j от 0 до q-1
27. result[i, j] = result[i, j] + A[i, b] \* B[b, j]
28. }

2.1.3 Разработка улучшенного алгоритма Винограда

Алгоритм винограда был улучшен следующими приёмами, перечисленными ниже.

1. Замена всех арифметических операций (+, - , и т.д.) после оператора присваивания на +=, -= и т.д. где это возможно (например, строки 15, 22, 28)
2. Занесение в циклах вычисления множителей вычисления первых двух элементов во внутренний цикл j (правая часть строк 7 и 13 теперь 0)
3. Введение переменной temp для сокращения числа обращений к матрице result через операторы индексирования
4. Замена цикла, корректирующего результат при нечётной общей размерности матриц, на аналогичные проверки в цикле умножения
5. Вычисление переменных m / 2 и чётности общей размерности заранее

Листинг 3. - Улучшенный алгоритм Винограда, псеводкод

1. Виноград(матр. А, целое n, целое m, матр. B, целое q, матр. result)
2. {
3. массив row\_factor[n];
5. // Вычисление множителей строк
6. Цикл i от 1 до n-1
7. rowFactor[i] = 0
8. Цикл j от 1 до m / 2
9. rowFactor[i] += A[i, 2j - 1] \* A[i, 2j]
11. // Вычисление множителей столбцов
12. Цикл i от 1 до q-1
13. colFactor[i] = 0
14. Цикл j от 1 до m / 2
15. colFactor[i] += B[2j - 1, i] \* B[2j, i]
17. целое d = m / 2;
18. целое odd = m % 2 ? 1 : 0
19. целое temp;
20. Цикл i от 1 до n-1
21. Цикл j от 1 до q-1
22. temp = odd ? A[i][m-1] \* B[m-1][j] : 0
23. temp -= row\_factor[i] + col\_factor[j]
24. Цикл k от 0 до d
25. temp +=(A[i, 2k-1]+B[2k, j])\*(A[i, 2k] + B[2k-1, j])
26. result[i][j] = temp
28. }

3. Технологическая часть

В технологической части описаны аппаратные характеристики машины, на которой запускался реализованный алгоритм и представлена реализация на языке программирования

3.1 Требования к программному обеспечению

Данная программа разрабатывалась на языке С стандарта C99, поддерживаемом многими операционными системами. Оптимизации компилятора отключены (-O0).

Характеристики машины, использовавшейся для разработки и тестов:

Процессор: 2,8 GHz Intel Core i7

Память16 ГБ 1600 MHz DDR3

Операционная система: MacOS Mojave

3.2 Средства реализации

Программа реализована на языке C стандарта C99

Язык Си был выбран за возможность ручной работы с памятью, скорости и возможности быстро и точно измерять процессорное время, затраченное программой. Оптимизации компилятора отключены (-O0)

3.3 Листинг кода

В листинге исходного кода опущены вспомогательные функции:

Выделение НУЛЕВОЙ матрицы

int\*\* alloc\_int\_matrix(int n, int m),

Чтение матрицы из файла

int\*\* read\_int\_matrix(char const\* filename, int\* row, int\* column),

Вывода матрицы на экран

void print\_int\_matrix(int row, int coloumn, int\*\* matrix)

Листинг 5. - Код вспомогательных функций, С99

1. int\*\* alloc\_int\_matrix(int n, int m)
2. {
3. int \*\*data = calloc(n \* sizeof(int\*) + n \* m, sizeof(int));
4. if (!data)
5. return NULL;
6. for(int i = 0; i < n; i++)
7. data[i] = (int\*)((char\*) data + n \* sizeof(int\*) + i \* m \* sizeof(int));
8. return data;
9. }
10. int\*\* read\_int\_matrix(char const\* filename, int\* row, int\* column)
11. {
12. FILE\* f = fopen(filename, "r");
13. int n;
14. int m;
15. fscanf(f, "%d", &n);
16. fscanf(f, "%d", &m);
17. \*row = n;
18. \*column = m;
19. int\*\* matrix = alloc\_int\_matrix(n, m);
20. for (int i = 0; i < n; ++i)
21. for (int j = 0; j < m; ++j)
22. {
23. if (!fscanf(f, "%d", &matrix[i][j]))
24. return NULL;
25. }
26. fclose(f);
27. return matrix;
28. }
29. void print\_int\_matrix(int row, int coloumn, int\*\* matrix)
30. {
31. for (int i = 0; i < row; ++i)
32. {
33. printf("\n\x1b[30;1m");
34. for (int j = 0; j < coloumn; ++j)
35. {
36. printf("%2d ", matrix[i][j]);
37. }
38. }
39. printf("\n\x1b[0m");
40. }
41. void classic(int\*\* A, int n, int m, int\*\* B, int q, int\*\* result)
42. {
43. for (int i = 0; i < n; i++)
44. for (int j = 0; j < q; j++)
45. for (int k = 0; k < m; k++)
46. result[i][j] += A[i][k] \* B[k][j];
47. }

Листинг 4. - Код программы, С99

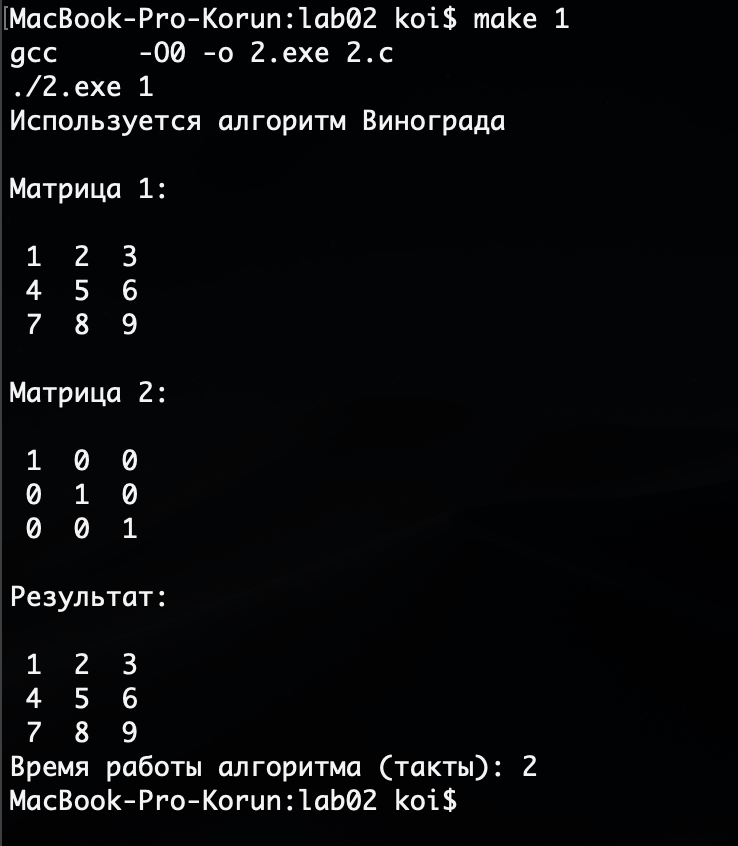
1. void winograd(int\*\* A, int n, int m, int\*\* B, int q, int\*\* result)
2. {
3. int d = m / 2;
4. // row factor A
5. int row\_factor[n];
6. for (int i = 0; i < n; ++i)
7. {
8. row\_factor[i] = A[i][0] \* A[i][1];
9. for (int j = 1; j < m/2; ++j)
10. row\_factor[i] = row\_factor[i] + A[i][2\*j] \* A[i][2\*j+1];
11. }
12. // column factor B
13. int col\_factor[q];
14. for (int i = 0; i < q; ++i)
15. {
16. col\_factor[i] = B[0][i] \* B[1][i];
17. for (int j = 1; j < m/2; ++j)
18. col\_factor[i] = col\_factor[i] + B[2\*j][i] \* B[2\*j+1][i];
19. }
21. // multiplying
22. for (int i = 0; i < n; ++i)
23. for (int j = 0; j < q; ++j)
24. {
25. result[i][j] = -(row\_factor[i] + col\_factor[j]);
26. for (int k = 0; k < m/2; ++k)
27. {
28. result[i][j] = result[i][j] + \
29. (A[i][2\*k] + B[2\*k+1][j]) \* (A[i][2\*k+1] + B[2\*k][j]);
30. }
31. // if odd m
32. if (m%2 != 0)
33. {
34. for (int i = 0; i < n; ++i)
35. for (int j = 0; j < q; ++j)
36. {
37. printf("result[%d][%d]: %d adding %d \* %d = %d\n", i,j, result[i][j], A[i][m-1], B[m-1][j],result[i][j]+ A[i][m-1] \* B[m-1][j]);
38. result[i][j] += A[i][m-1] \* B[m-1][j];
39. }
40. }
41. }
42. void winograd2(int\*\* A, int n, int m, int\*\* B, int q, int\*\* result)
43. {
44. int d = m / 2;
45. int odd = m % 2 ? 1 : 0;
46. // row factor A
47. int row\_factor[n];
48. for (int i = 0; i < n; ++i)
49. {
50. row\_factor[i] = 0;
51. for (int j = 0; j < d; ++j)
52. row\_factor[i] += A[i][2\*j] \* A[i][2\*j+1];
53. }
54. // column factor B
55. int col\_factor[q];
56. for (int i = 0; i < q; ++i)
57. {
58. col\_factor[i] = 0;
59. for (int j = 0; j < d; ++j)
60. col\_factor[i] += B[2\*j][i] \* B[2\*j+1][i];
61. }
62. int temp;
63. // multiplying
64. for (int i = 0; i < n; ++i)
65. for (int j = 0; j < q; ++j)
66. {
67. temp = odd ? A[i][m-1] \* B[m-1][j] : 0;
68. temp -= row\_factor[i] + col\_factor[j];
69. for (int k = 0; k < d; ++k)
70. temp += \
71. (A[i][2\*k] + B[2\*k+1][j]) \* (A[i][2\*k+1] + B[2\*k][j]);
72. result[i][j] = temp;
73. }
74. int main(int argc, char const \*argv[])
75. {
76. int alg = 0;
77. if (argc > 1)
78. switch (argv[1][0])
79. {
80. case '0': if (argc < 3) printf("Используется Базовый алгоритм \n");
81. alg = 0;
82. break;
83. case '1': if (argc < 3) printf("Используется алгоритм Винограда\n");
84. alg = 1;
85. break;
86. case '2': if (argc < 3) printf("Используется улучшенный алгоритм Винограда\n");
87. alg = 2;
88. break;
89. }
90. else
91. printf("Аргумент не распознан.\n");
92. int a\_row, a\_col;
93. int\*\* a = read\_int\_matrix("input1.txt", &a\_row, &a\_col);
94. int b\_row, b\_col;
95. int\*\* b = read\_int\_matrix("input2.txt", &b\_row, &b\_col);
96. int\*\* result = alloc\_int\_matrix(a\_row, b\_col);
97. void ((\*algs[3]))() = {&classic, &winograd, winograd2};
98. clock\_t alg\_time = clock();
99. algs[alg](a, a\_row, a\_col, b, b\_col, result);
100. alg\_time = clock() - alg\_time;
101. // print matrices
102. if (argc > 2 && argv[2][0] == 's')
103. {
104. printf("%lu\n", alg\_time);
105. }
106. else
107. {
108. printf("\nМатрица 1:\n");
109. print\_int\_matrix(a\_row, a\_col, a);
110. printf("\nМатрица 2:\n");
111. print\_int\_matrix(b\_row, b\_col, b);
112. printf("\nРезультат:\n");
113. print\_int\_matrix(a\_row, b\_col, result);
114. printf("Время работы алгоритма (такты): %lu\n", alg\_time);
115. printf("%c\n", argv[2][0]);
116. }
117. return 0;
118. }

4. Экспериментальная часть

В экспериментальной части производятся тесты ПО и анализ полученных данных

4.1 Примеры работы

Пример компиляции, запуска и демонстрация вывода программы (рис. 2):



*Рис. 2 Компиляция и запуск программы*

4.2 Постановка эксперимента

Алгоритмы будут протестированы по скорости работы и будет показана требуемая память.

Тестирование по памяти будет проводиться с помощью утилиты ps, рассчитываться будет объём резидентной памяти[[1]](#footnote-2) для процесса программы после отработки всех алгоритмов (для этого в конец программы была временно добавлена инструкция ожидания ввода).

Для замера времени выполнения (скорости работы) были выбраны квадратные матрицы чётных и нечётных размерностей от 10х10 до 4000х4000. Уже при размерности матриц 2000x2000 программа работает ощутимо долго, поэтому нецелесообразно тестировать её на размерностях в десятки тысяч единиц.

Для замера выделенной памяти нет необходимости ждать завершения работы алгоритма, поэтому размерности матриц выбраны в пределах от 10х10 до 30000х30000

Заключение

В данной работе были реализованы и протестированы алгоритмы умножения матриц. Также была улучшена реализация алгоритма Винограда.

Как видно из результатов экспериментов, алгоритм винограда с улучшениями предпочтительнее для матриц любой размерности, если нет ограничений по памяти.

1. [↑](#footnote-ref-2)