Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 2

по курсу «Анализ Алгоритмов»

на тему: «Умножение матриц»

Студент ИУ7-55Б _____ Романов С. К.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.,Строганов Ю
(Подпись, дата) (И. О. Фамилия)

2022 г.

Оглавление

ВВЕДЕНИЕ **3**

1 Аналитическая часть **5**

1.1 Стандартный алгоритм 5

1.2 Алгоритм Копперсмита – Винограда 5

1.3 Вывод 6

2 Конструкторская часть **7**

2.1 Схемы алгоритмов 7

3 Технологическая часть **13**

3.1 Требования к программному обеспечению 13

3.2 Средства реализации 13

3.3 Листинги кода 13

3.3.1 Классический алгоритм перемножения матриц 14

3.3.2 Алгоритм перемножения матриц Копперсмита−Винограда 15

3.3.3 Оптимизированный алгоритм перемножения матриц Копперсмита−Винограда 16

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **19**

ВВЕДЕНИЕ

Алгоритм Копперсмита–Винограда[1] – алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. На текущий момент сложнотсь алгоритма составляет $O(n^{2,3728596})$.

Цель лабораторной работы:

– Изучение и исследование особенностей оптимизации сложных вычислений.

Задачи лабораторной работы:

1. Изучить и реализовать алгоритмы перемножения матриц.
   – Классический;
   – Копперсмита–Винограда;
   – Копперсмита–Винограда с оптимизациями согласно варианту;
2. Создать ПО, реализующее алгоритмы, указанные в варианте.
3. Провести анализ затрат работы программы по времени и по памяти, выяснить влияющие на них характеристики.
4. Создать отчёт, содержащий:
   – актуальность исследования;
   – характеристики предложенной реализации (по времени и памяти);
   – краткие рекомендации об особенностях применения оптимизаций (важно помнить об улучшениях, которые использует компилятор).
   – результаты тестирования;
   – Выводы.

Для достижения поставленных целей и задач необходимо:

1. Изучить теоретические основы алгоритма Копперсмита–Винограда;

2. Реализовать выше обозначенный алгоритм.

3. Изучить и реализовать возможные пути оптимизации.

4. Провести экспериментальное исследование.

В ходе работы будут затронуты следующие темы:

1. Оптимизация вычислений.

2. Алгоритмы перемножения матриц.

3. Оценка реализаций алгоритмов.

1 Аналитическая часть

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.3}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм Копперсмита – Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4$, что эквивалентно

(1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырёх умножений - шесть, а вместо трёх сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

1.3 Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которого от классического алгоритма — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

На рисунках 2.2 и 2.3 представлена схема алгоритма Копперсмита−Винограда.

На рисунках 2.4 и **??** представлена схема оптимизированного алгоритма Копперсмита−Винограда.

Для алгоритма Копперсмита−Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, из-за того что отпадает необходимость в последнем цикле.

Согласно варианту, алгоритм можно оптимизировать следующим образом:

- Предварительно получить строки столбцы соответствующих матриц;
- заменить операцию x = x + k; на x += k;
- заменить умножение на 2 на побитовый сдвиг;

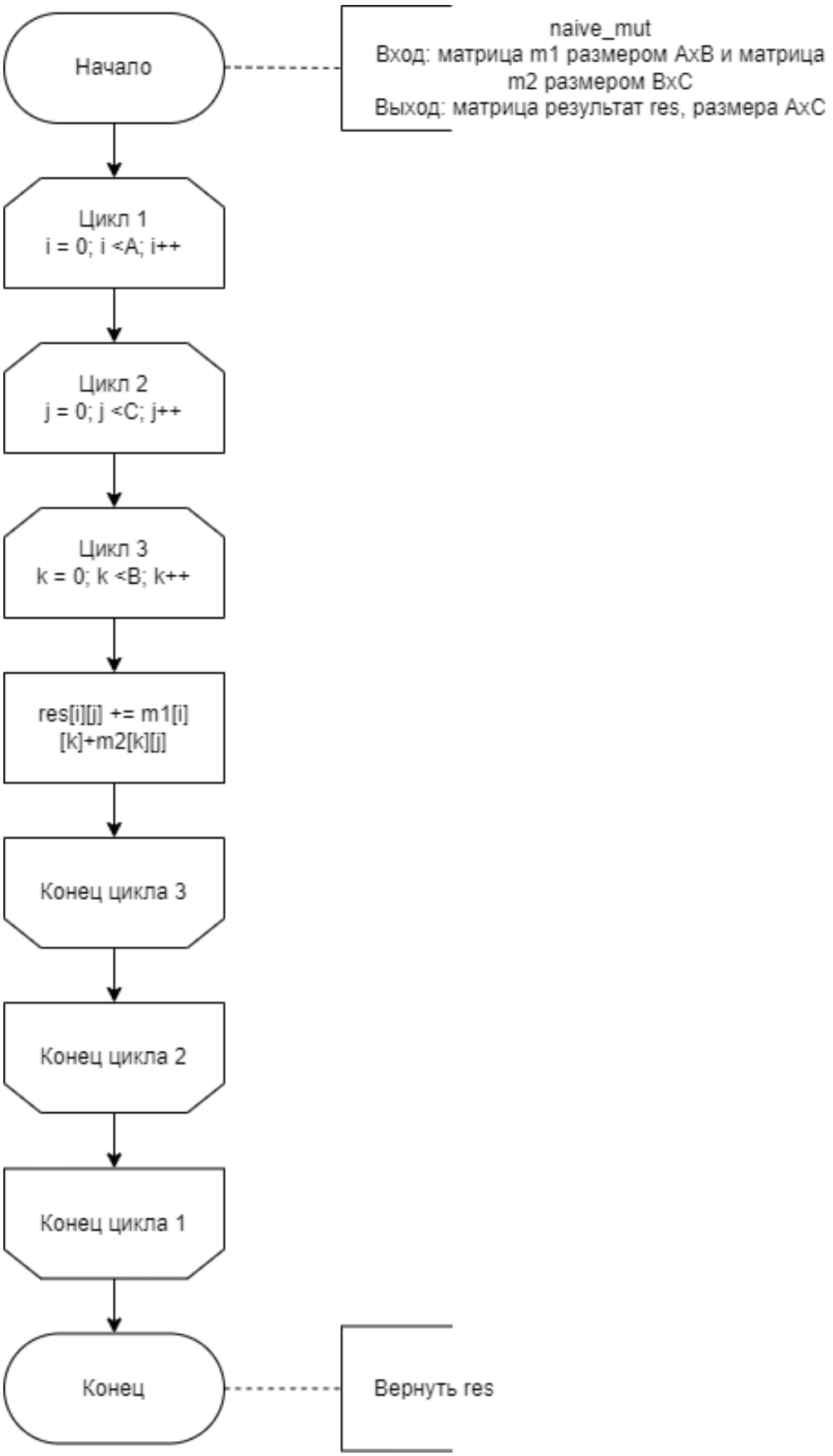
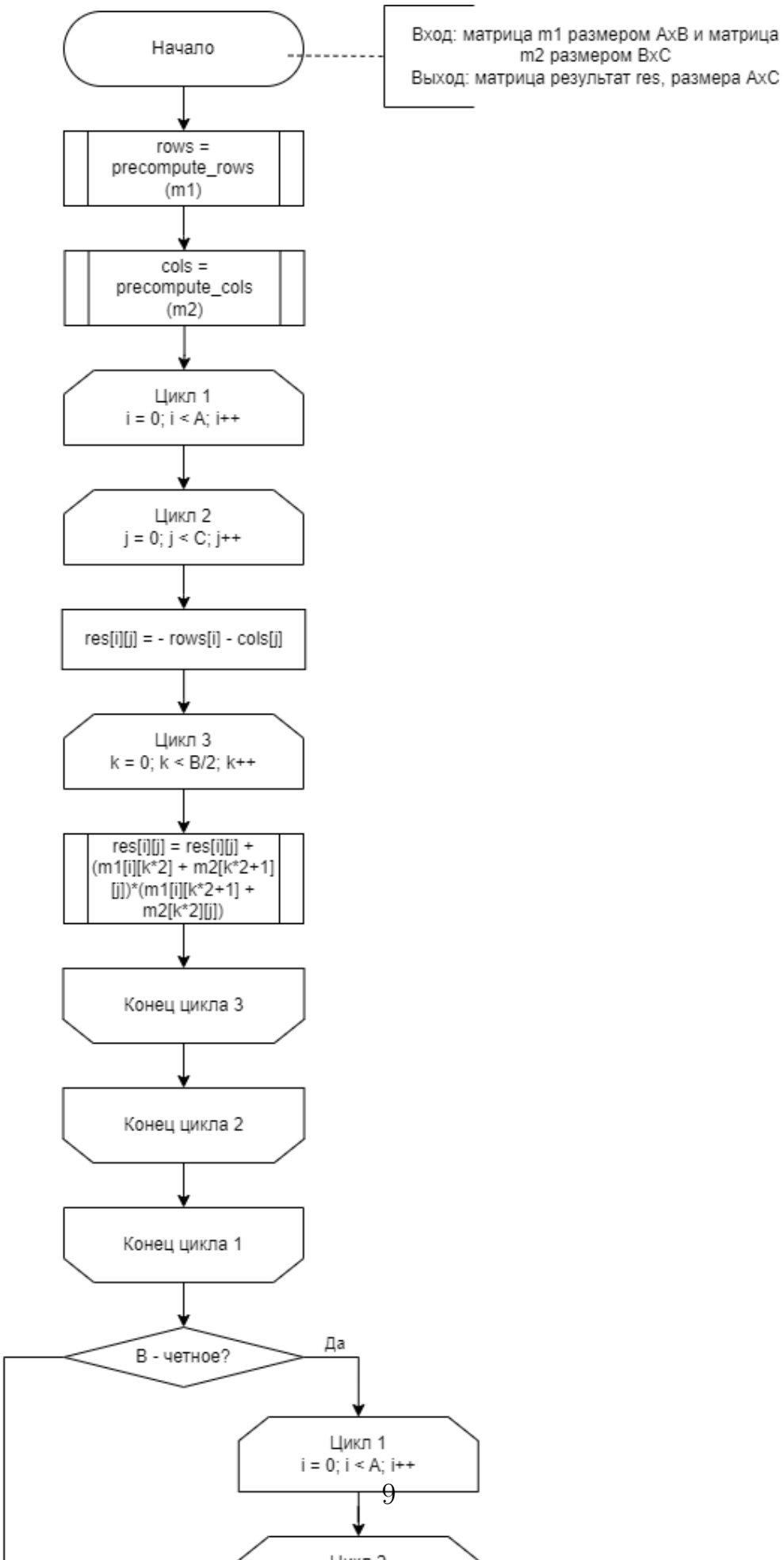
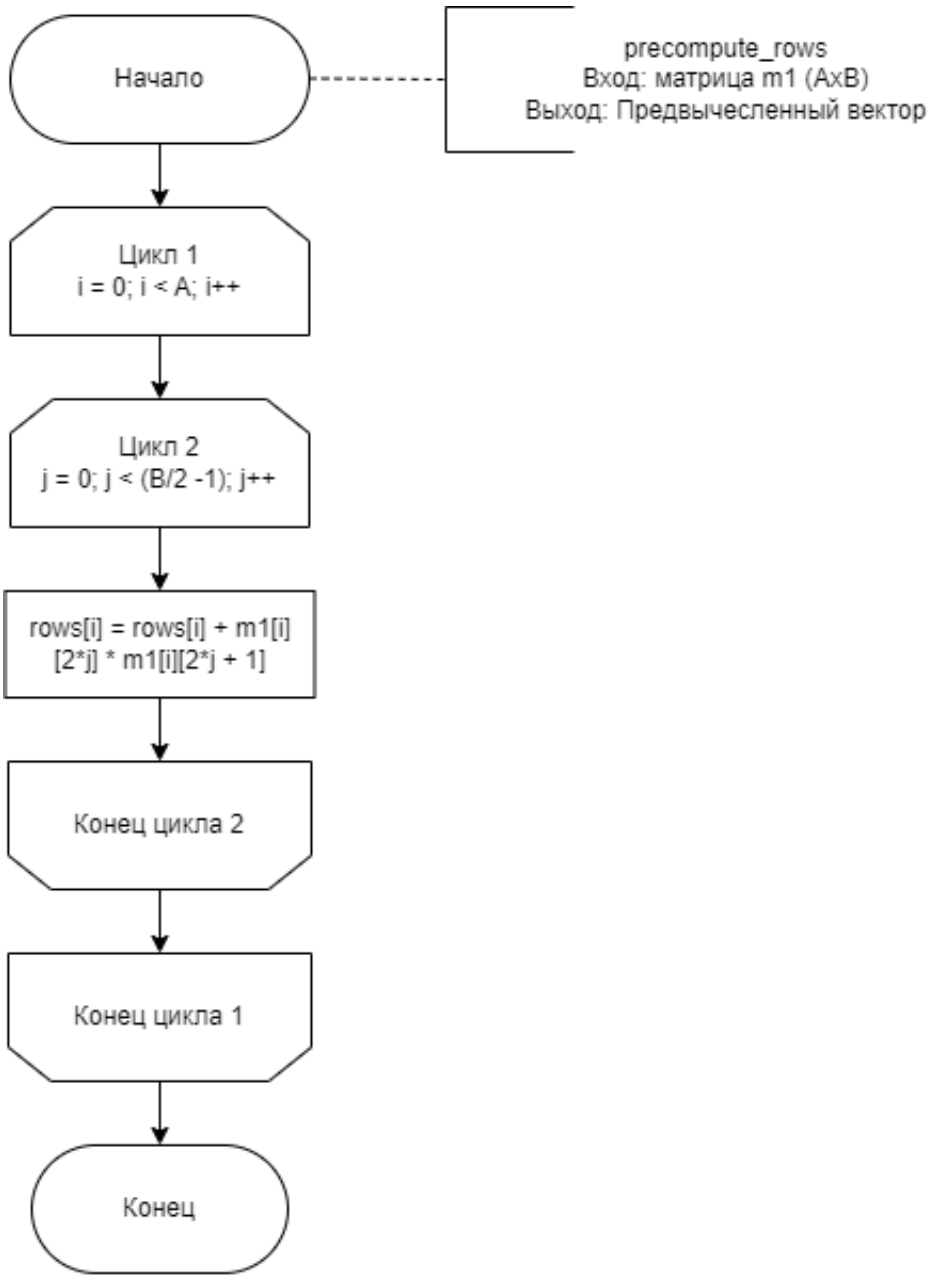
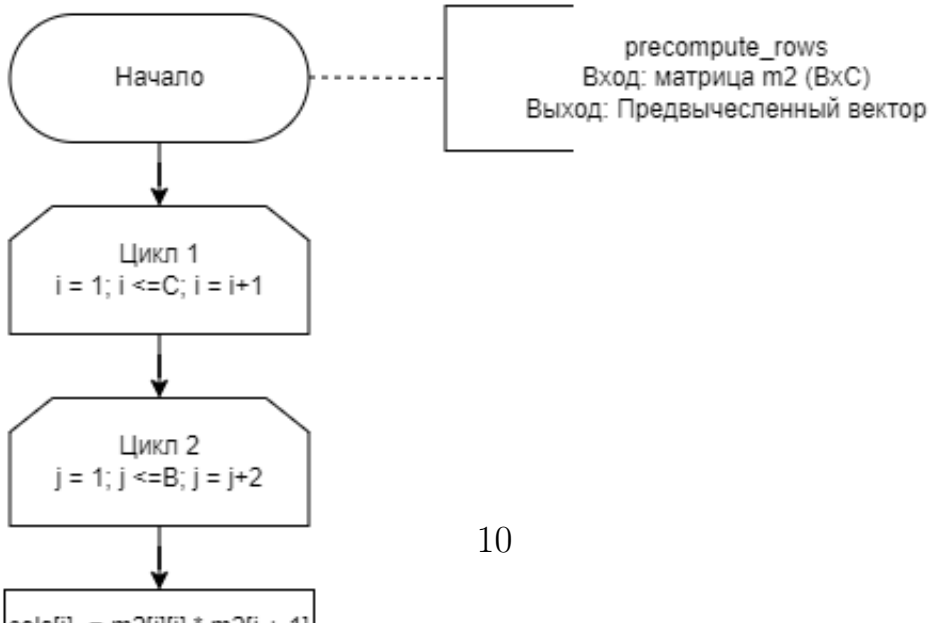
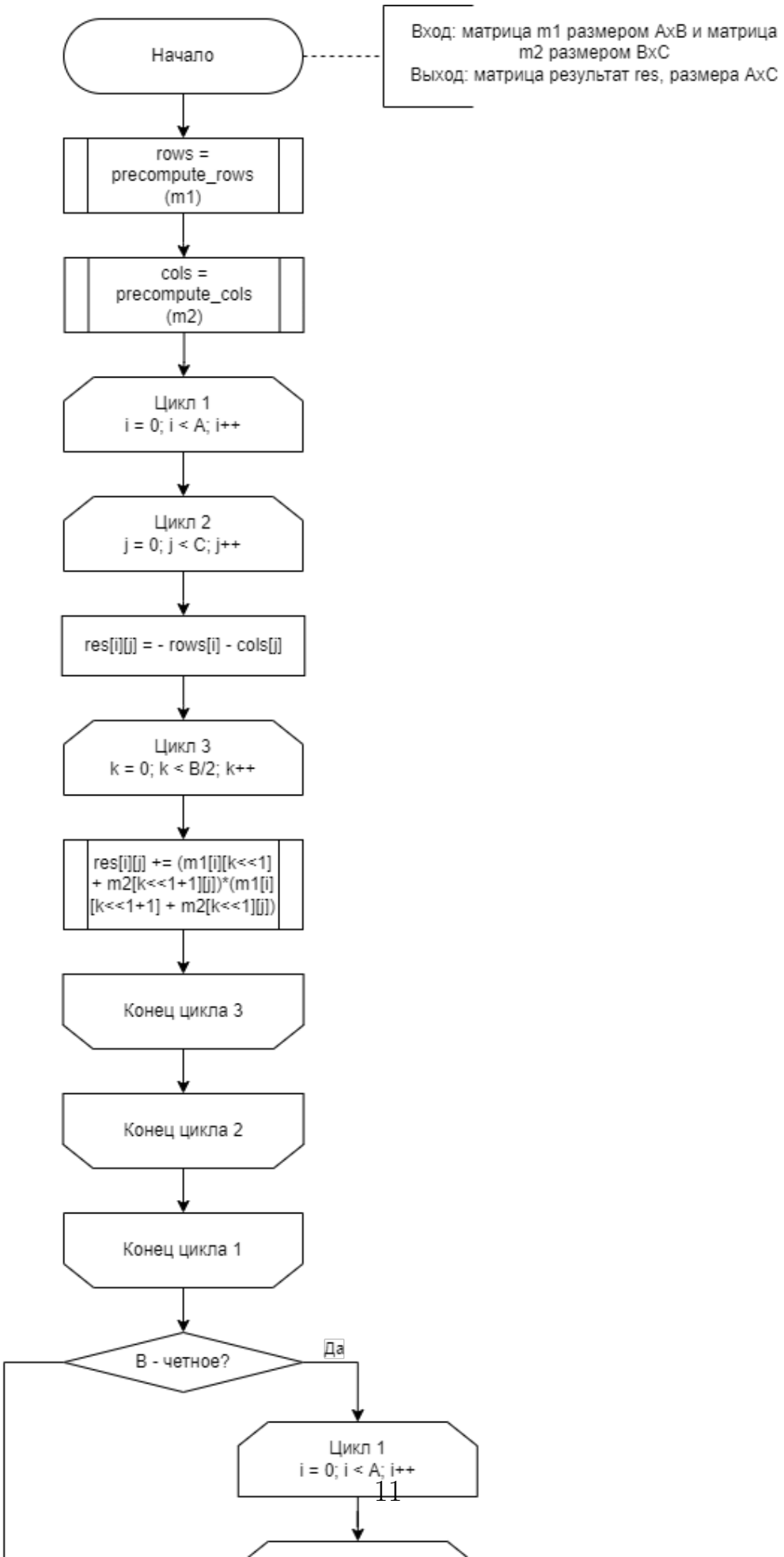


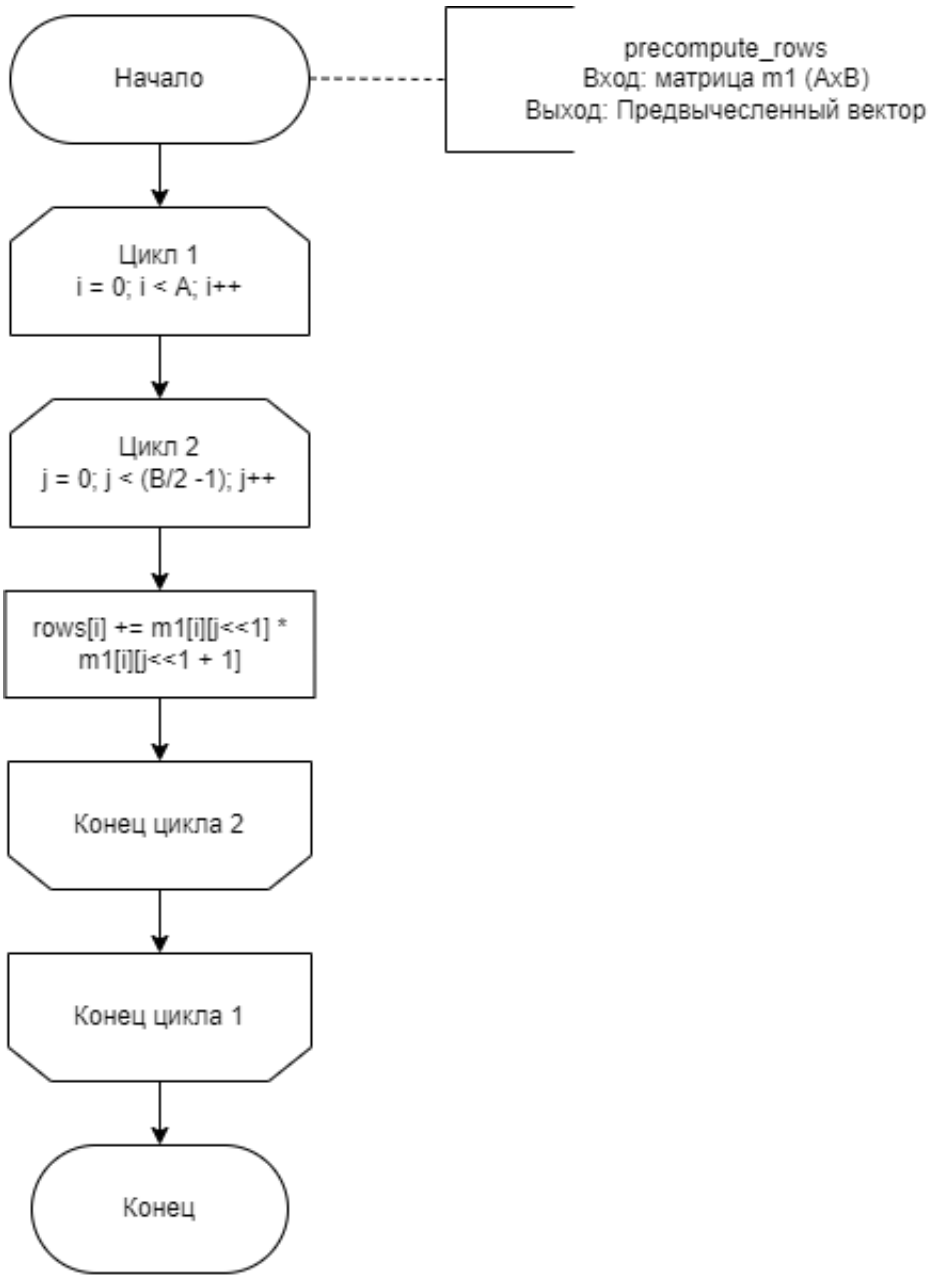
Рисунок 2.1 – Классическое перемножение матриц

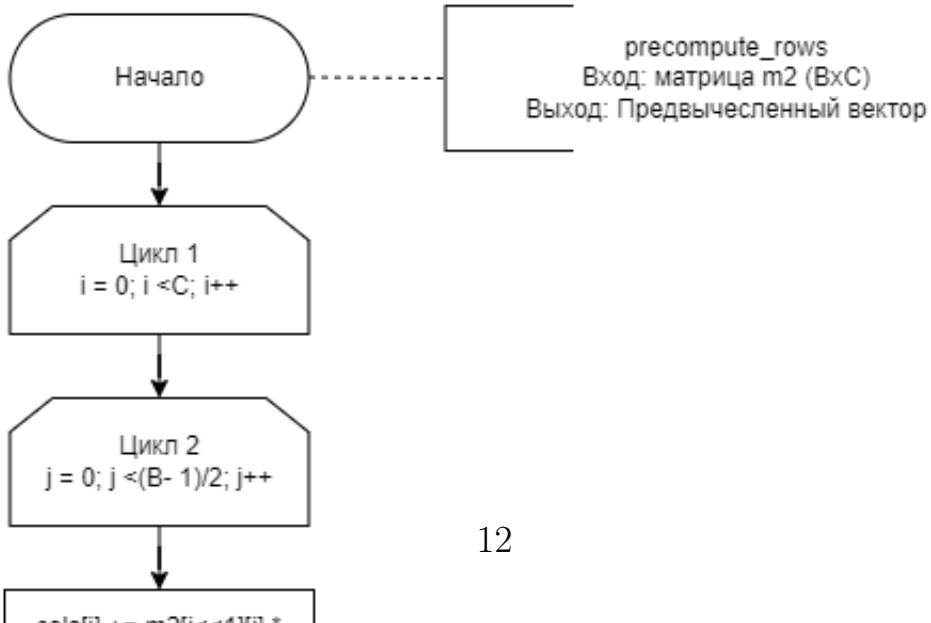

Начало
Вход: матрица m1 размером AxB и матрица m2 размером BxC
Выход: матрица результат res, размера AxC
rows = precompute_rows (m1)
cols = precompute_cols (m2)
Цикл 1
i = 0; i < A; i++
Цикл 2
j = 0; j < C; j++
res[i][j] = - rows[i] - cols[j]
Цикл 3
k = 0; k < B/2; k++
res[i][j] = res[i][j] + (m1[i][k*2] + m2[k*2+1][j])*(m1[i][k*2+1] + m2[k*2][j])
Конец цикла 3
Конец цикла 2
Конец цикла 1
B - четное?
Да
Цикл 1
i = 0; i < A; i++


Начало
precompute_rows
Вход: матрица m1 (AxB)
Выход: Предвычесленный вектор
Цикл 1
i = 0; i < A; i++
Цикл 2
j = 0; j < (B/2 -1); j++
rows[i] = rows[i] + m1[i]
[2*j] * m1[i][2*j + 1]
Конец цикла 2
Конец цикла 1
Конец


Начало
precompute_rows
Вход: матрица m2 (BxC)
Выход: Предвычесленный вектор
Цикл 1
i = 1; i <=C; i = i+1
Цикл 2
j = 1; j <=B; j = j+2


Начало
Вход: матрица m1 размером AxB и матрица m2 размером BxC
Выход: матрица результат res, размера AxC
rows = precompute_rows (m1)
cols = precompute_cols (m2)
Цикл 1
i = 0; i < A; i++
Цикл 2
j = 0; j < C; j++
res[i][j] = - rows[i] - cols[j]
Цикл 3
k = 0; k < B/2; k++
res[i][j] += (m1[i][k<<1] + m2[k<<1+1][j])*(m1[i][k<<1+1] + m2[k<<1][j])
Конец цикла 3
Конец цикла 2
Конец цикла 1
B - четное?
Да
Цикл 1
i = 0; i < A; i++


Начало
precompute_rows
Вход: матрица m1 (AxB)
Выход: Предвычесленный вектор
Цикл 1
i = 0; i < A; i++
Цикл 2
j = 0; j < (B/2 -1); j++
rows[i] += m1[i][j<<1] *
m1[i][j<<1 + 1]
Конец цикла 2
Конец цикла 1
Конец


Начало
precompute_rows
Вход: матрица m2 (BxC)
Выход: Предвычесленный вектор
Цикл 1
i = 0; i <C; i++
Цикл 2
j = 0; j <(B- 1)/2; j++

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программному обеспечению

К программе предъявляется ряд условий:

– На вход подается 3 числа (A, B, C), определяющие размеры матриц, а также сами матрицы размерами AxB и BxC;

– На выход ПО должно выводить результат перемножения 2 матриц;

3.2 Средства реализации

Для реализации данной лабораторной работы необходимо установить следующее программное обеспечение:

- Rust Programming Language v1.64.0 - язык программирования
- Criterion.rs v0.4.0 - Средство визуализации данных
- LaTeX - система документооборота

3.3 Листинги кода

В следующих листингах представлены следующие алгоритмы

1. В листинге 3.1 представлен классический алгоритм перемножения матриц
2. В листингах 3.2 и 3.3 представлены алгоритм перемножения матриц с использованием алгоритма Копперсмита−Винограда
3. В листингах 3.4 и 3.5 представлены алгоритм перемножения матриц с использованием оптимизированного алгоритма Копперсмита−Винограда

3.3.1 Классический алгоритм перемножения матриц

Листинг 3.1 – Классический алгоритм перемножения матриц

```
1    fn naive_mut(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
2        let mut result = vec![vec![0; m2[0].len()]; self.len()];
3        for i in 0..self.len() {
4            for j in 0..m2[0].len() {
5                for k in 0..m2.len() {
6                    result[i][j] += self[i][k] * m2[k][j];
7                }
8            }
9        }
```

Замечание: здесь и далее под ключевым словом self понимается ссылка на экземпляр типа матрицы ($Vec < Vec < i32 >>$).

3.3.2 Алгоритм перемножения матриц Копперсмита–Винограда

Листинг 3.2 – Алгоритм перемножения матриц Копперсмита–Винограда

```
1   fn winograd_mut(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
2       let mut result = vec![vec![0; m2[0].len()]; self.len()];
3
4       let row_factor = self.precompute_rows();
5       let col_factor = m2.precompute_cols();
6
7       for i in 0..result.len()
8       {
9           for j in 0..result[0].len()
10          {
11              result[i][j] = -row_factor[i] - col_factor[j];
12              for k in 0..(m2.len() / 2)
13              {
14                  result[i][j] = result[i][j] + (self[i][k*2] + m2[k*2+1][j]) *
                        (self[i][k*2+1] + m2[k*2][j]);
15              }
16          }
17      }
18
19      if m2.len()%2 == 1
20      {
21          for i in 0..result.len()
22          {
23              for j in 0..result[0].len()
24              {
25                  result[i][j] = result[i][j] + self[i][m2.len()-1] * m2[m2.len()-1][j]
26              }
27          }
28      }
29
30      result
31  }
```

Листинг 3.3 – Алгоритм перемножения матриц Копперсмита–Винограда, вычисление векторов

```
1   fn precompute_rows(&self) -> Vec<i32>
2   {
3       let mut row_factor = vec![0; self.len()];
4       for i in 0..self.len()
5       {
```

```
6           for j in 0..((self[0].len() - 1)/2)
7           {
8               row_factor[i] = row_factor[i] + self[i][j*2] * self[i][j*2 + 1];
9           }
10      }
11
12      row_factor
13  }
14
15  fn precompute_cols(&self) -> Vec<i32> {
16      let mut col_factor = vec![0; self[0].len()];
17      for i in 0..self[0].len()
18      {
19          for j in 0..((self.len() - 1)/2)
20          {
21              col_factor[i] = col_factor[i] + self[j*2][i] * self[j*2 + 1][i];
22          }
23      }
24
25      col_factor
26  }
```

3.3.3 Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда

Листинг 3.4 – Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда

```
1   fn winograd_mut_improved(&self, m2: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
2       let (a, b, c) = (self.len(), m2.len(), m2[0].len());
3       let mut result = vec![vec![0; c]; a];
4
5       let row_factor = self.precompute_rows_imp();
6       let col_factor = m2.precompute_cols_imp();
7
8       for i in 0..a
9       {
10          for j in 0..c
11          {
12              result[i][j] = -row_factor[i] - col_factor[j];
13              for k in 0..(m2.len() / 2)
14              {
15                  result[i][j] += (self[i][k<<1] + m2[(k<<1)+1][j]) * (self[i][(k<<1)+1] +
                        m2[k<<1][j]);
```

```
            }
        }
    }

    if m2.len()%2 == 1
    {
        for i in 0..a
        {
            for j in 0..c
            {
                result[i][j] += self[i][b-1] * m2[b-1][j]
            }
        }
    }

    result
}
```

Листинг 3.5 – Оптимизированный алгоритм перемножения матриц Копперсмита–Винограда, вычисление векторов

```
fn precompute_rows_imp(&self) -> Vec<i32> {
    let mut row_factor = vec![0; self.len()];
    for i in 0..self.len()
    {
        for j in 0..((self[0].len() - 1)/2)
        {
            row_factor[i] += self[i][j<<1] * self[i][(j<<1) + 1];
        }
    }

    row_factor
}

fn precompute_cols_imp(&self) -> Vec<i32> {
    let mut col_factor = vec![0; self[0].len()];
    for i in 0..self[0].len()
    {
        for j in 0..((self.len() - 1)/2)
        {
            col_factor[i] += self[j<<1][i] * self[(j<<1) + 1][i];
        }
    }

    col_factor
}
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Copeersmith D.*, *Winograd S.* Matrix Multiplication via Arithmetic Progressions // Academic Press. — 1990. — Дата обращения: 19.10.2022.