МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Умножение матриц

Работу выполнила: Подвашецкий Дмитрий, ИУ7-54Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение **2**

1 Аналитическая часть **3**

Вывод **4**

2 Конструкторская часть **5**
2.1 Схемы алгоритмов 5
2.2 Оценка трудоемкости алгоритмов 9
2.2.1 Стандартный алгоритм 10
2.2.2 Алгоритм Винограда 10

Вывод **12**

3 Технологическая часть **13**
3.1 Выбор ЯП 13
3.2 Замеры времени 13
3.3 Работа с матрицами 13
3.4 Требования к ПО 13
3.5 Сведения о модулях программы 14

Вывод **19**

4 Экспериментальная часть **20**
4.1 Сравнительный анализ алгоритмов 20

Вывод **22**

Заключение **23**

Список литературы **24**

Введение

Матрица - математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы.

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае, количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

Матрицы допускают следующие алгебраические операции:

1. сложение матриц, имеющих один и тот же размер;
2. умножение матриц подходящего размера;
3. умножение матрицы на элемент основного кольца или поля;

Умножение матриц - одна из основных операций над матрицами.

Целью данной лабораторной работы является реализация и изучение алгоритма умножения матриц методом Винограда и стандартного.

Задачами данной лабораторной являются:

1. изучение стандарного метода и метода Винограда для умножения матриц;
2. реализация данных двух методов, а так же оптимизация последнего;
3. теоретический анализ трудоемкости рассматриваемых алгоритмов;
4. экспериментальное подтверждение различий во временно́й эффективности рассматриваемых алгоритмов;
5. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

Операция умножения матриц повсеместно приминяется в математике, физике, программировании и т.д. Для того, чтобы произведение матрицы A, размерами n на m, на матрицу B, размерами u на v, было возможно, необходимо, чтобы n = u.

В данной лабораторной работе я рассмотрю два алгоритма умножения матриц.

Первый алгоритм - стандартный.

Пусть есть две матрицы:

$$A_{nm} = \begin{pmatrix} a_{00} & ... & a_{0m} \\ ... & ... & ... \\ a_{n0} & ... & a_{nm} \end{pmatrix}; B_{mk} = \begin{pmatrix} b_{00} & ... & b_{0k} \\ ... & ... & ... \\ b_{m0} & ... & b_{mk} \end{pmatrix}$$

Тогда, пусть:

$$C_{nk} = A_{nm}B_{mk}$$

Где ij эллемент матрицы C_{nk} вычисляется как скалярное произведение i строки матрицы A_{nm} на j столбец матрицы B_{mk}.

$$C_{ij} = a_{0i} * b_{j0} + ... + a_{mi} * b_{jm}, i = [1..n], j = [1..m] \tag{1.1}$$

Второй алгоритм - Винограда.

В данном алгоритме, также как и в предыдущем, каждый элемент производной матрицы считается как скалярное произведение. Рассмотрим два вектора:

$$V = \begin{pmatrix} v1 & v2 & v3 & v4 \end{pmatrix}$$
$$W = \begin{pmatrix} w1 & w2 & w3 & w4 \end{pmatrix}$$

Их скалярное произведение равно:

$$V * W = v1w1 + v2w2 + v3w3 + v4w4 \tag{1.2}$$

Выражение (1.2) можно переписать:

$$V*W = (v1+w2)(v2+w1)+(v3+v4)(v4+w3)-v1v2-v3v4-w1w2-w3w4 \tag{1.3}$$

Можно заметить, что правую часть данного выражения можно высчитать заранее для каждого вектора. Это означает, что, при предварительной обработке векторов мы можем, в дальнейшем, сэкономить 2 операции умножения, за счет 2х лишних операций сложения.

Если при умножении двух матриц произвести обработку строк первой и столбцов второй, то можно добиться большей эффективности по времени. [1]

Вывод

Можно сделать вывод, что алгоритм Винограда, на системах в которых умножение требудет больших вычислительных задач, должен работать значительно быстрее.

2 Конструкторская часть

2.1 Схемы алгоритмов

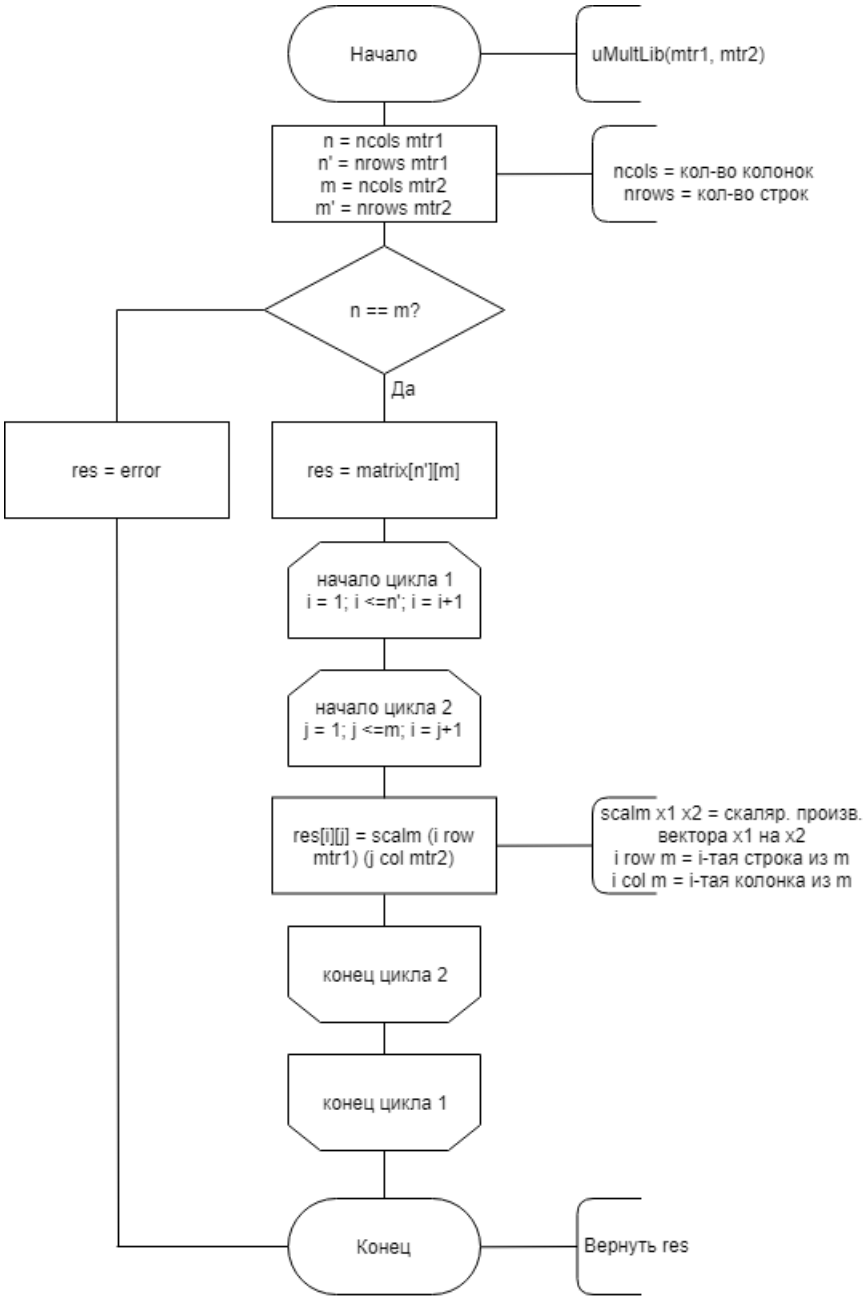

Схема 1. Стандартный алгоритм умножения

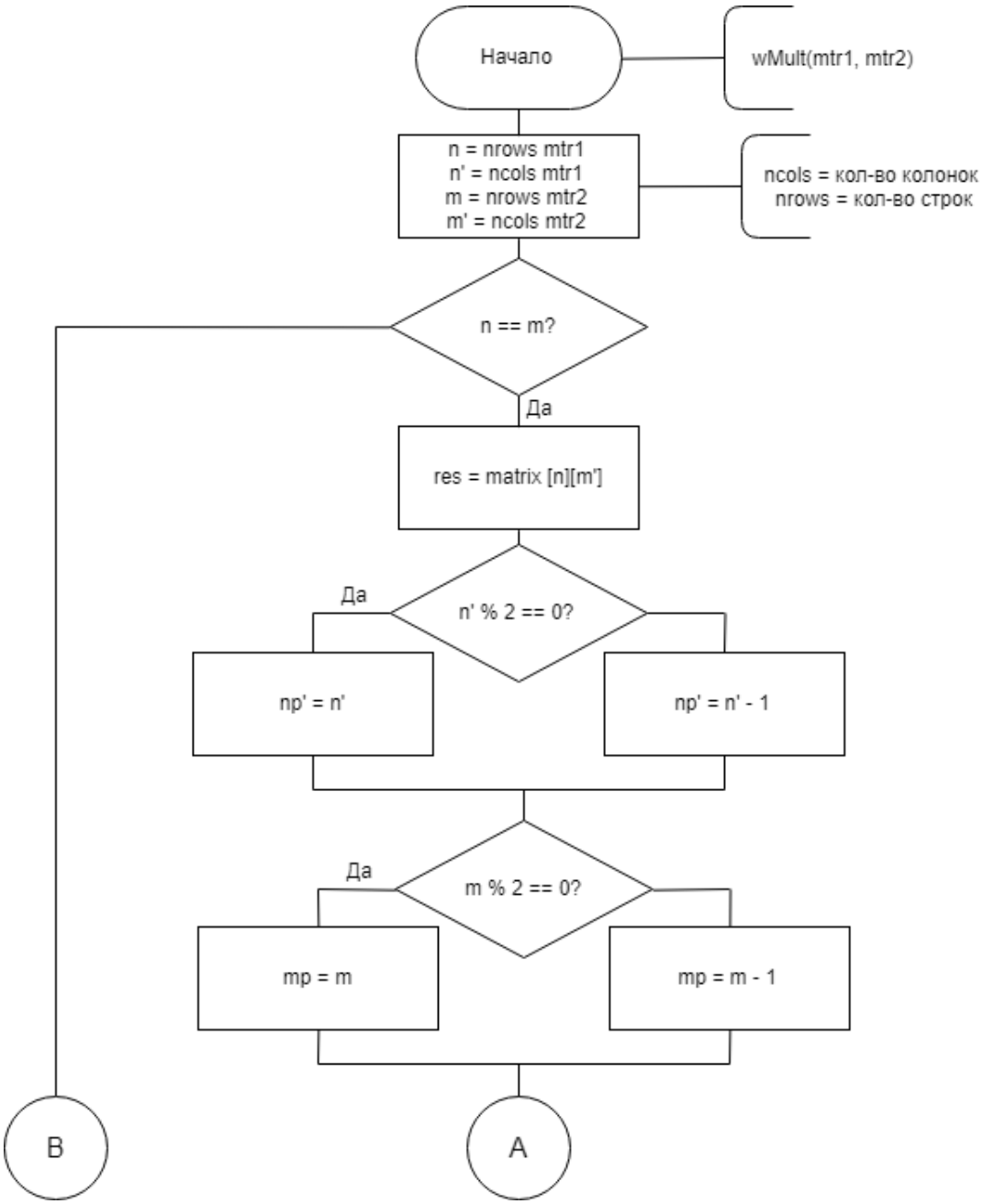

Схема 2.1. Алгоритм умножения методом Винограда

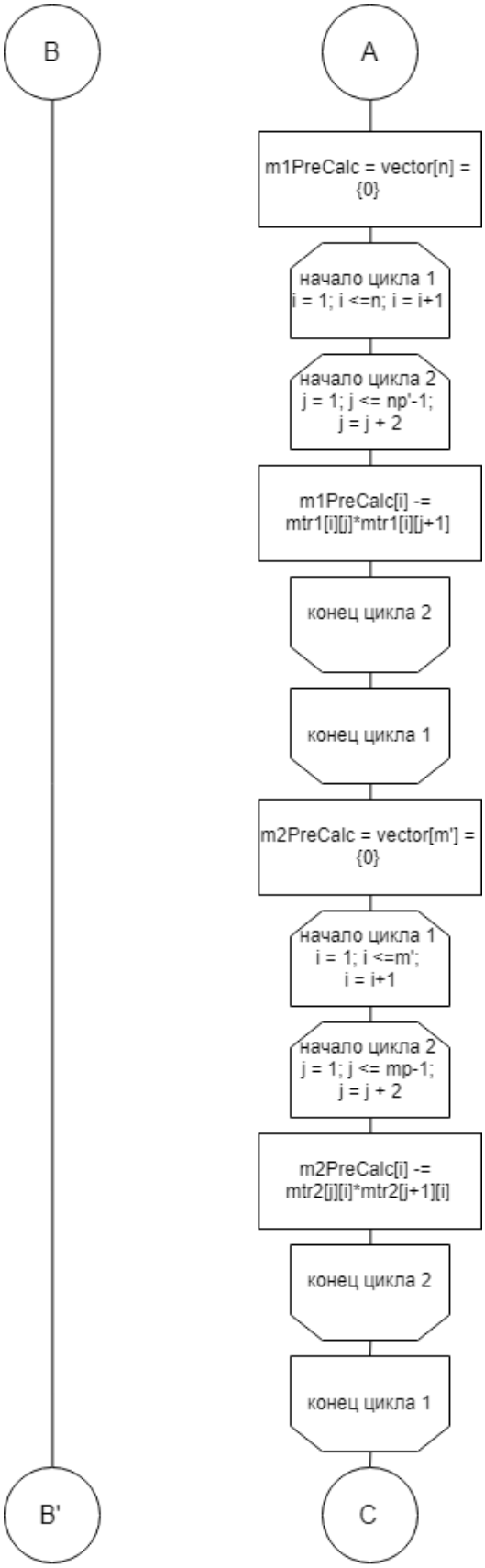

Схема 2.2. Алгоритм умножения методом Винограда

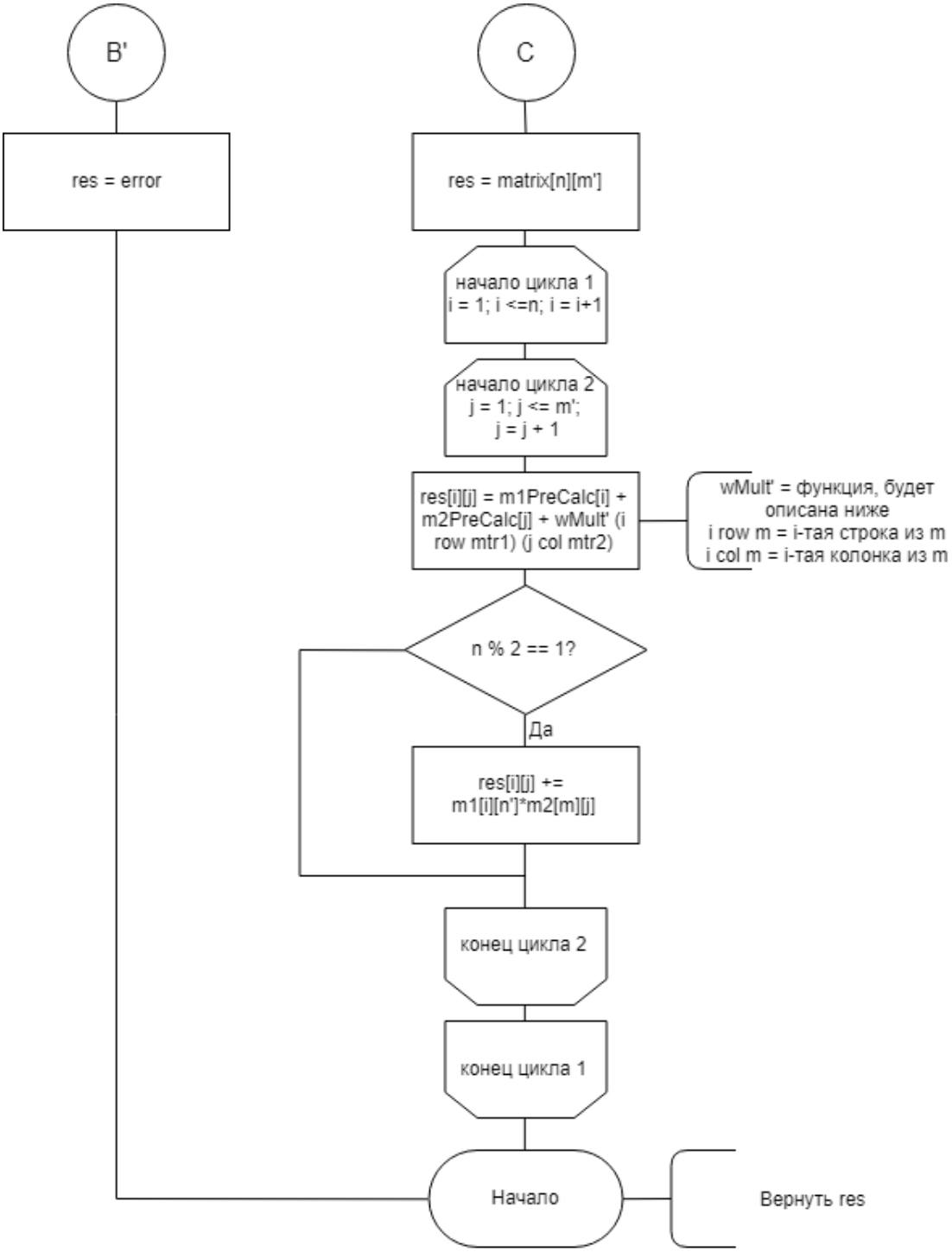

Схема 2.3. Алгоритм умножения методом Винограда

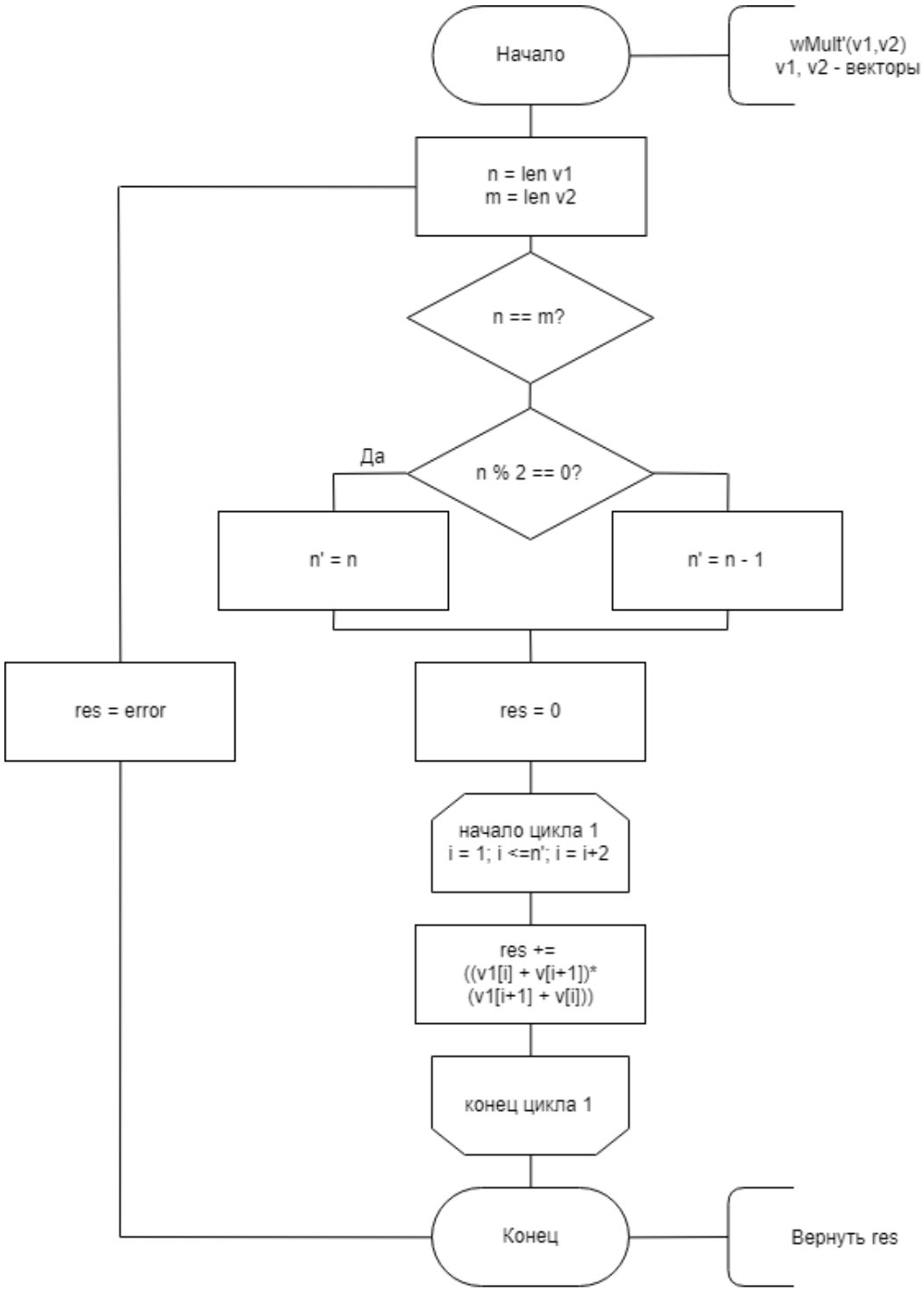

Схема 3. Функция wMult' (для Схема. 2.3.)

2.2 Оценка трудоемкости алгоритмов

Предположим, что у нас есть операции:

1. присваивание (=);
2. одномерная индексация a[i] (a !! i) (unsafeIntex a i), все они сводятся к (a'

+ i), a' - адресс a;

3. двумерная индексация a[i][j] ((a !! i) !! j) (getElem i j a), все они сводятся к ((a' + i*n) + j), где n - длина строки матрицы, a' - адресс a;

4. операция сравнения <, >, ==;

5. арифметические операции +, -, *, /

Так как мне не известно точное значение трудоемкости для каждой из этих операций, примем её за 1 для каждой. [4]

Пусть у нас на вход подается две матрицы: A, размерности NxM, и B, размерности MxK. N,M,K - четные.

2.2.1 Стандартный алгоритм

Обратимся к Листингу 3.1.

Для вычисления трудоемкости данного алгоритма, я предлагаю сначала вычислить трудоемкость алгоритма scalm', и после уже uMultLib.

Так как при данных входных значениях вероятность правдивости условия в ветвлении равно p = 1, то трудоемкость ветвления будет равняться [4]:

$$F_{if} = f_{then} * p + f_{else} * (1 - p) = f_{then} \tag{2.1}$$

Это же справедливо и для функции uMultLib.

Допустим, что трудоемкость функции V.length = 1, а трудоемкость функции Prelude.sum = 1 + 1 + M(3 + 3) = 2 + 6M, тогда трудоемкость scalm':

$$F_{scalm} = 1 + 1 + 1 + 2 + 6M + 1 + M(3 + 3) = 6 + 12M \tag{2.2}$$

Далее вычислим трудоемкость uMultLib. Допустим, что трудоемкость функций ncols, nrows = 1.

$$F_{uMultLib} = 1+4+4+1+N(3+1+K(3+F_{scalm})) = 10+4N+9NK+12NKM \tag{2.3}$$

2.2.2 Алгоритм Винограда

Обратимся к Листингу 3.2.

В данном алгоритме также не будем рассчитывать трудоемкость ветвелния (см. 4.1.1.).

Так как трудоемкость следования есть сумма трудоемкости каждого блока, то для начала можно расчитать трудоемкость строк 3-13. Допустим, что трудоемкость функции even (n % 2 == 0) = 2, тогда:

$$F_{3-13} = 1 + 4 + 4 + 3 + 3 = 15 \tag{2.4}$$

Трудоемкость 15 строки:

$$F_{15} = 1+N(3+1+M/2*(3+1+3+1+4))+2+6M = 5+4N+6M+6MN \tag{2.5}$$

Трудоемкость 16 строки:

$$F_{16} = 1+K(3+1+M/2*(3+1+3+1+4))+2+6M = 5+4K+6M+6MK \quad (2.6)$$

Трудоемкость алгоритма wMult':

$$F_{wMult'} = 1+4+3+1+M/2(3+9)+2+6M = 13+12M \quad (2.7)$$

Трудоемкость 18 строки:

$$F_{18} = 1+N(3+1+K(3+5+F_{wMult'}+2+2)) = 1+4N+25NK+12MKN \quad (2.8)$$

Следовательно, трудоемкость wMult равна:

$$F_{wMult} = F_{3-13}+F_{15}+F16+F18 = 49+8N+25NK+24M+6MN+4K+6MK+12MKN \quad (2.9)$$

Трудоемкость при нечетной матрице

При вычислении меняется только трудоемкость 18ой строки:

$$F_{18n} = 1+N(3+1+K(3+5+F_{wMult'}+2+7+2)) \quad (2.10)$$

Следовательно, трудоемкость wMult равна:

$$F_{wMult} = F_{3-13}+F_{15}+F16+F18n \quad (2.11)$$

Трудоемкость второй оптимизации алгоритма Винограда

Обратимся к Листингу 3.4.
из него видно, что строки 3-18 такие же как и в предыдущем случае, следовательно можно взять уже расчитанные значения.
Трудоемкость алгоритма wMultU2':

$$F_{wMultU2'} = 1+3+1+M/2(3+9)+2+6M = 9+12M \quad (2.12)$$

Трудоемкость 18 строки:

$$F_{18u2} = 1+2+N(3+1+K(3+5+F_{wMultU2'}+1)) = 3+4N+18NK+12MNK \quad (2.13)$$

Следовательно, трудоемкость wMultU2 равна:

$$F_{wMultU2} = F_{3-13}+F_{15}+F16+F18u2 = 48+8N+18NK+24M+6MN+4K+6MK+12MKN \quad (2.14)$$

Трудоемкость при нечетной матрице

При вычислении меняется только трудоемкость 18ой строки:

$$F_{18u2n} = 1+2+N(3+1+K(3+5+F_{wMultU2'}+2+7)) \quad (2.15)$$

Следовательно, трудоемкость wMult равна:

$$F_{wMult} = F_{3-13}+F_{15}+F16+F18u2n \quad (2.16)$$

Вывод

Из данного раздела можно сделать вывод, что алгоритм Винограда, по стравнению со стандартным алгоритмом умножения, более труден в реализации, но, согласно пунктам 2.2.1 и 2.2.2 для выполнения алгоритма Винограда требуется меньше операций, следовательно скорость работы у него выше.

3 Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран Haskell, для ознакомления с ним.

3.2 Замеры времени

Замер времени работы алгоритмов производился при помощи функций getCurrentTime, diffUTCTime из библиотеки Data.Time.Clock. [2]

Также производится усреднение времени работы улгоритмов. Для этого время считается для 10 вызовов, и после делится на 10.

3.3 Работа с матрицами

Для обработки матриц использовалась библиотека Data.Matrix, наряду с массивом массивов. [3]

3.4 Требования к ПО

Требования к вводу:

1. Имя файла, содержащего значения первой матрицы
2. Имя файла, содержащего значения второй матрицы

Требования к программе:

1. Корректный ввод, корректный вывод, программа не должна аварийно завершаться

Требования структуре входного файла: Значения в строках должны быть записаны на одной строке. Разделитель - пробел. Конец строки - переход на новую строку.

Пример:

1 2
3 4

3.5 Сведения о модулях программы

Программа состоит из:

- main.hs - главный файл программы
- UsualMultLib.hs - файл с функцией вычисления произведения матриц стандартным алгоритмом (Листинг 3.1.)
- WinogradMult.hs - файл с функцией вычисления произведения матриц методом Винограда (Листинг 3.2.)
- WinogradMultU1.hs - файл с первой модификацией метода Винограда (Листинг 3.3.)
- WinogradMultU2.hs - файл со второй модификацией метода Винограда (Листинг 3.4.)
- WinogradMultU3.hs - файл с третьей модификацией метода Винограда (Листинг 3.5.)

Листинг 3.1: Стандартный алгоритм

```
scalm' :: (Num a) => Vector a -> Vector a -> a
scalm' v1 v2 =
        if V.length v1 == V.length v2
            then Prelude.sum( [(unsafeIndex v1 i)*(unsafeIndex v2
                i) | i <- [0..(V.length v1)-1]] )
            else error "Mtr.size.Error"

uMultLib :: (Num a) => Matrix a -> Matrix a -> Matrix a
uMultLib m1 m2 = do
    if n' == m
        then fromLists [[scalm' (getRow i m1) (getCol j m2) | j
           <- [1..m]] | i <- [1..n']]
        else error "Mtr.size.Error"
            where
                n = ncols m1
                m = nrows m2

                n' = nrows m1
                m' = ncols m2
```

Листинг 3.2: Алгоритм Винограда

```
wMult :: (Num a) => Matrix a -> Matrix a -> Matrix a
wMult m1 m2 =
    if m == n'
        then res
        else error "Mtr.Size.Error"
    where
        n = nrows m1
```

```
        n' = ncols m1
        m = nrows m2
        m' = ncols m2

        np' = if even n' then n' else n' - 1
        mp = if even m then m else m - 1

        m1PreCalc = [Prelude.sum ([-1*(getElem i j m1)*(getElem i
            (j+1) m1) | j <- [1, 3..np'-1]]) | i <- [1..n]]
        m2PreCalc = [Prelude.sum ([-1*(getElem j i m2)*(getElem (
           j+1) i m2) | j <- [1, 3..mp-1]]) | i <- [1..m']]

        res = matrix n m' $ \(i,j) ->
                (m1PreCalc !! (i-1)) + (m2PreCalc !! (j-1)) +
                wMult' (getRow i m1) (getCol j m2) +
                if odd n' then getElem i n' m1 * getElem m j m2
                   else 0

wMult' :: (Num a) => Vector a -> Vector a -> a
wMult' v1 v2 =
    if n == m
        then res
        else error "Oh no, smth goes wrong"
    where
        n = Data.Vector.length v1
        m = Data.Vector.length v2

        n' = if even n then n else n-1

        res = Prelude.sum ( [ ((unsafeIndex v1 i) + (unsafeIndex
           v2 (i+1)))*
                              ((unsafeIndex v1 (i+1) + (
                                 unsafeIndex v2 i))) | i <- [0,
                                 2..n'-2] ] )
```

Листинг 3.3: Алгоритм Винограда модификация 1

```
wMultU1 :: (Num a) => Matrix a -> Matrix a -> Matrix a
wMultU1 m1 m2 =
    if m == n'
        then res
        else error "Mtr.Size.Error"
    where
        n = nrows m1
        n' = ncols m1
        m = nrows m2
        m' = ncols m2

        np' = if even n' then n' else n' - 1
        mp = if even m then m else m - 1

```

```
15          m1PreCalc = [Prelude.sum ([-1*(getElem i j m1)*(getElem i
                (j+1) m1) | j <- [1, 3..np'-1]])
16                              | i <- [1..n]]
17          m2PreCalc = [Prelude.sum ([-1*(getElem j i m2)*(getElem (
                j+1) i m2) | j <- [1, 3..mp-1]])
18                              | i <- [1..m']]
19
20          res = matrix n m' $ \(i,j) ->
21                  (m1PreCalc !! (i-1)) + (m2PreCalc !! (j-1)) +
22                  wMultU1' (getRow i m1) n (getCol j m2) m' +
23                  if odd n' then getElem i n' m1 * getElem m j m2
                        else 0
24
25  wMultU1' :: (Num a) => Vector a -> Int -> Vector a -> Int -> a
26  wMultU1' v1 n v2 m =
27      if n == m
28          then res
29          else error "Oh no, smth goes wrong"
30      where
31          n' = if even n then n else n-1
32
33          res = Prelude.sum ( [ ((unsafeIndex v1 i) + (unsafeIndex
                v2 (i+1)))*
34                                  ((unsafeIndex v1 (i+1) + (
                                        unsafeIndex v2 i))) | i <- [0,
                                        2..n'-2] ] )
```

Листинг 3.4: Алгоритм Винограда модификация 2

```
1   wMultU2 :: (Num a) => Matrix a -> Matrix a -> Matrix a
2   wMultU2 m1 m2 =
3       if m == n'
4           then res
5           else error "Mtr.Size.Error"
6       where
7           n = nrows m1
8           n' = ncols m1
9           m = nrows m2
10          m' = ncols m2
11
12          np' = if even n' then n' else n' - 1
13          mp = if even m then m else m - 1
14
15          m1PreCalc = [Prelude.sum ([-1*(getElem i j m1)*(getElem i
                (j+1) m1) | j <- [1, 3..np'-1]])
16                              | i <- [1..n]]
17          m2PreCalc = [Prelude.sum ([-1*(getElem j i m2)*(getElem (
                j+1) i m2) | j <- [1, 3..mp-1]])
18                              | i <- [1..m']]
19
20          res = if odd n'
```

```
            then matrix n m' $ \(i,j) ->
                    (m1PreCalc !! (i-1)) + (m2PreCalc !! (j-1)) +
                    wMultU2' (getRow i m1) n (getCol j m2) m'
                    + getElem i n' m1 * getElem m j m2
            else matrix n m' $ \(i,j) ->
                    (m1PreCalc !! (i-1)) + (m2PreCalc !! (j-1)) +
                    wMultU2' (getRow i m1) n (getCol j m2) m'

wMultU2' :: (Num a) => Vector a -> Int -> Vector a -> Int -> a
wMultU2' v1 n v2 m =
    if n == m
        then res
        else error "Oh no, smth goes wrong"
    where
        n' = if even n then n else n-1

        res = Prelude.sum ( [ ((unsafeIndex v1 i) + (unsafeIndex
           v2 (i+1))) *
                              ((unsafeIndex v1 (i+1) + (
                                 unsafeIndex v2 i))) | i <- [0,
                                 2..n'-2] ] )
```

Листинг 3.5: Алгоритм Винограда модификация 3

```
transpW :: Num a => [[a]] -> [[a]]
transpW mtr = [[(mtr !! i) !! j | i <- [0..m-1]] | j <- [0..n-1]]
                where
                    n = Prelude.length $ mtr !! 0
                    m = Prelude.length $ mtr

wMultU3 :: (Num a) => [[a]] -> [[a]] -> [[a]]
wMultU3 m1 m2 =
    if m == n'
        then res
        else error "Mtr.Size.Error"
    where
        n = length m1
        n' = length (m1 !! 0)

        tm2 = transpW m2

        m = length tm2
        m' = length (tm2 !! 0)

        np' = if even n' then n' else n' - 1
        mp' = if even m' then m' else m' - 1

        m1PreCalc = [Prelude.sum ([-1*((m1 !! i) !! j)*((m1 !! i)
            !! (j+1)) | j <- [0, 2..np'-2]]) | i <- [0..n-1]]
        m2PreCalc = [Prelude.sum ([-1*((tm2 !! i) !! j)*((tm2 !!
           (i)) !! (j+1)) | j <- [0, 2..mp'-2]]) | i <- [0..m-1]]
```

```
26
27          res = if even n'
28              then [[ (m1PreCalc !! (i)) + (m2PreCalc !! (j)) +
                    wMultU3' (m1 !! i) (tm2 !! j) | j <- [0..n-1] ] |
                    i <- [0..m'-1] ]
29              else [[ (m1PreCalc !! (i)) + (m2PreCalc !! (j)) +
                    wMultU3' (m1 !! i) (tm2 !! j) +
30                          ((m1 !! i) !! (n'-1)) * ((tm2 !! j) !! (m
                               -1)) | j <- [0..n-1] ] | i <- [0..m'-1]
                                ]
31
32  wMultU3' :: (Num a) => [a] -> [a] -> a
33  wMultU3' v1 v2 =
34      if n == m
35          then res
36          else error "Oh no, smth goes wrong"
37      where
38          n = length v1
39          m = length v2
40
41          n' = if even n then n else n-1
42
43          res = Prelude.sum ( [ ((v1 !! i) + (v2 !! (i+1)))*
44                                  ((v1 !! (i+1) + (v2 !! i))) | i <-
                                       [0, 2..n'-2] ] )
```

Вывод

В данном разделе мною были реализованны необходимые алгоритмы.

4 Экспериментальная часть

4.1 Сравнительный анализ алгоритмов

Анализируя Таблицу 1. и Рис 1., можно увидеть, что алгоритм Винограда работает в среднем на 86% быстрее стандартного.

Таблица. 1. Сравнение времени работы.

Размер	uMultLib (c)	wMult (c)	wMultU1 (c)	wMultU2 (c)	wMultU3 (c)
1000	0.28867	0.16954	0.12037	0.13563	0.01286
1100	0.33329	0.19832	0.14068	0.18681	0.01186
1200	0.40183	0.23013	0.18502	0.20674	0.01611
1300	0.41554	0.28036	0.24424	0.21943	0.02106
1400	0.51159	0.31670	0.26930	0.24890	0.02652
1500	0.55242	0.35118	0.30339	0.28773	0.01721
1600	0.81657	0.39693	0.33408	0.31989	0.02862
1700	0.90922	0.44291	0.37459	0.36126	0.03922
1800	0.97126	0.49055	0.41933	0.47617	0.04633
1900	1.06381	0.54654	0.44668	0.52928	0.04538
2000	1.15446	0.60316	0.49008	0.58330	0.03577

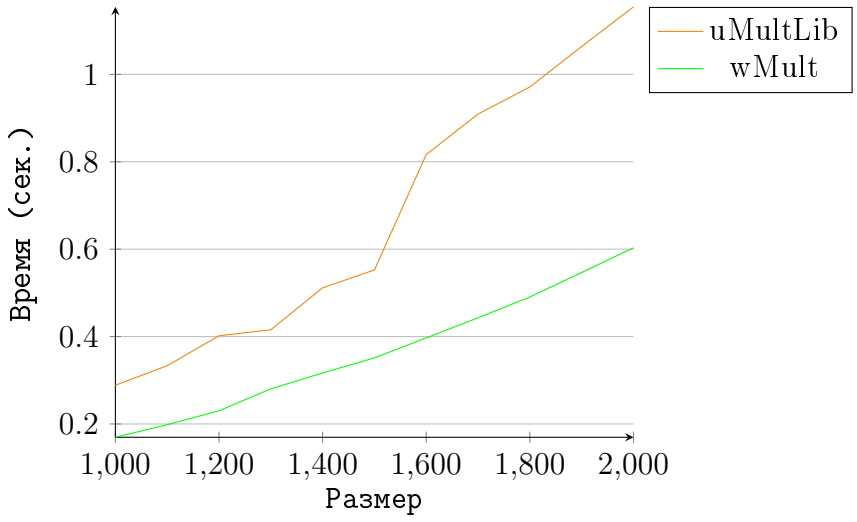

Рис. 1. График времени работы стандартного и немодифицированного Винограда. (Таблице 1.)

Сравнивая алгоритм Винограда и его модификации можно увидеть что, обе модификации показывают приблизительно одинаковое время, в тоже время,

немодифицированный алгоритм медленее в среднем на 23%. (Таблица 1. и Рис 2.)

Также мною была написанна вариация второй модификации алгоритма без использования библиотеки для работы с матрицами.

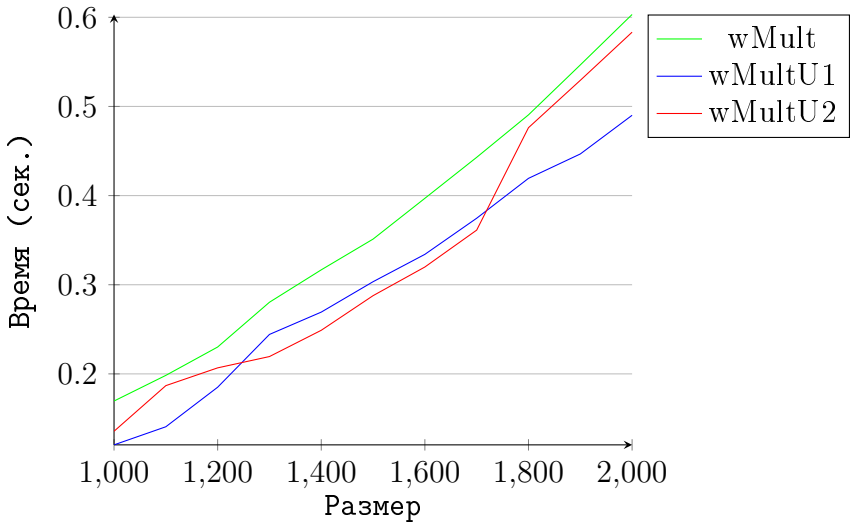

Рис. 2. График времени работы немодифицированного Винограда и двух его модификаций. (Таблице 1.)

На Рис 3. видно, что разница во времени работы колосальна. Алгоритм, написанный с использованием сторонней библиотеки, работает в среднем в 15 раз медленее.

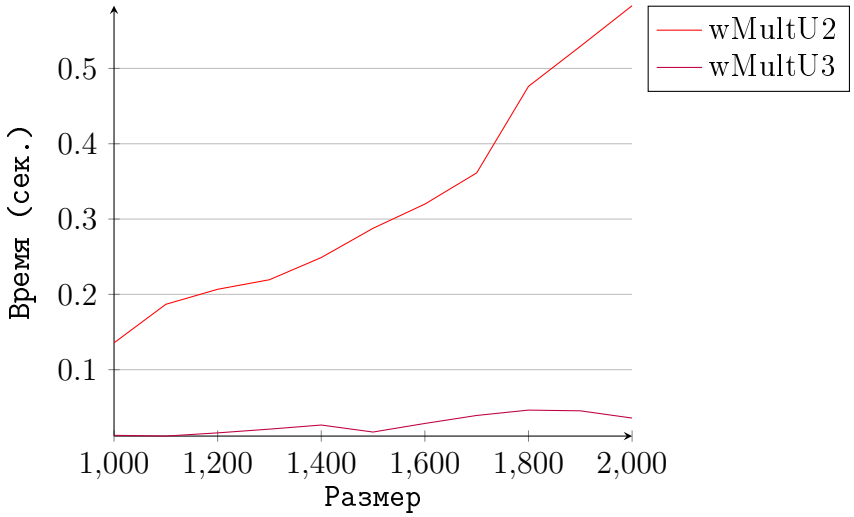

Рис. 3. График времени работы второй модификации, написанной с использованием библиотеки, и без.

Вывод

По результатам данной главы можно сделать вывод, что немодифицированный алгоритм Винограда работает в среднем на 86% быстрее стандартного.

Первая и вторая модификации алгоритма Винограда показывают примерно одинаковое время. в тоже время немодифицированный алгоритм медленне в среднем на 23%.

Также вторая модификация была реализованна без использования сторонней библиотеки, что позволило уменьшить время обработки в 15 раз. Так как Haskell использует ленивые вычисления, у меня нет уверенности в том, что я смог полностью избавиться от них.

Заключение

При написании данной лабораторной работы, мною были изучены стандартный метод умножения матриц и метод Винограда.

Были реализованны данные методы, наряду с тремя видами оптимизации последнего.

Эксперементально было подтверждено различие во временной эффективности данных методов. Стандартный метод в среднем работает медленее Винограда на 86%. Первая и вторая модификации - на 23% быстрее, чем немодифицированный. Третья модификация (которая является версией второй модификации, написанной без использования сторонних библиотек) работает в 15 раз быстрее чем вторая.

Список литературы

1. Умножение матриц. [Электронный ресурс] Режим доступа: http://www.algolib.narod.ru/Matl Последння дата обращения: 15.10.2019

2. Документация Haskell по модулю Data.Time.Clock. [Электронный ресурс] Режим доступа: http://hackage.haskell.org/package/time-1.9.3/docs/Data-Time-Clock.html Последння дата обращения: 15.10.2019

3. Документация Haskell по модулю Data.Matrix. [Электронный ресурс] Режим доступа: https://hackage.haskell.org/package/matrix-0.2.2/docs/Data-Matrix.html Последння дата обращения: 15.10.2019

4. Трудоемкость алгоритмов и временные оценки. [Электронный ресурс] Режим доступа: https://is.gd/3FilMM Последння дата обращения: 15.10.2019

5. Реализация алгоритма умножения матриц по Винограду на языке Haskell. Анисимов Н.С, Строганов Ю.В. [Электронный ресурс] Режим доступа: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell Последння дата обращения: 15.10.2019