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 и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт по лабораторной работе №2
по дисциплине "Анализ алгоритмов"

Тема Умножение матриц

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) ________________

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

Москва — 2020 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Стандартный алгоритм . . . 3
1.2 Алгоритм Копперсмита – Винограда . . . 3
1.3 Вывод . . . 4

2 Конструкторская часть **5**
2.1 Схемы алгоритмов . . . 5
2.2 Модель вычислений . . . 6
2.3 Трудоёмкость алгоритмов . . . 6
2.3.1 Стандартный алгоритм умножения матриц . . . 6
2.3.2 Алгоритм Копперсмита — Винограда . . . 6
2.3.3 Оптимизированный алгоритм Копперсмита — Винограда . . . 6
2.4 Вывод . . . 6

3 Технологическая часть **10**
3.1 Требование к ПО . . . 10
3.2 Средства реализации . . . 10
3.3 Реализация алгоритмов . . . 10
3.4 Тестовые данные . . . 12
3.5 Вывод . . . 13

4 Исследовательская часть **14**
4.1 Технические характеристики . . . 14
4.2 Время выполнения алгоритмов . . . 14
4.3 Вывод . . . 14

Заключение **16**

Литература **16**

Введение

Алгоритм Копперсмита — Винограда — 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.

Алгоритм Копперсмита — Винограда, с учётом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости.

Задачи лабораторной работы:

1. Изучение и реализация трёх алгоритмов умножения матриц: обычный, Копперсмита-Винограда, оптимизированный Копперсмита-Винограда;

2. Сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

3. Сравнительный анализ алгоритмов на основе экспериментальных данных;

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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \tag{1.4}$$

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 ?? приведена схема стандартного алгоритма умножения матриц.

На рисунках ?? и ?? представлена схема алгоритма Копперсмита — Винограда.

На рисунках ?? и ?? представлена схема оптимизированного алгоритма Копперсмита — Винограда.

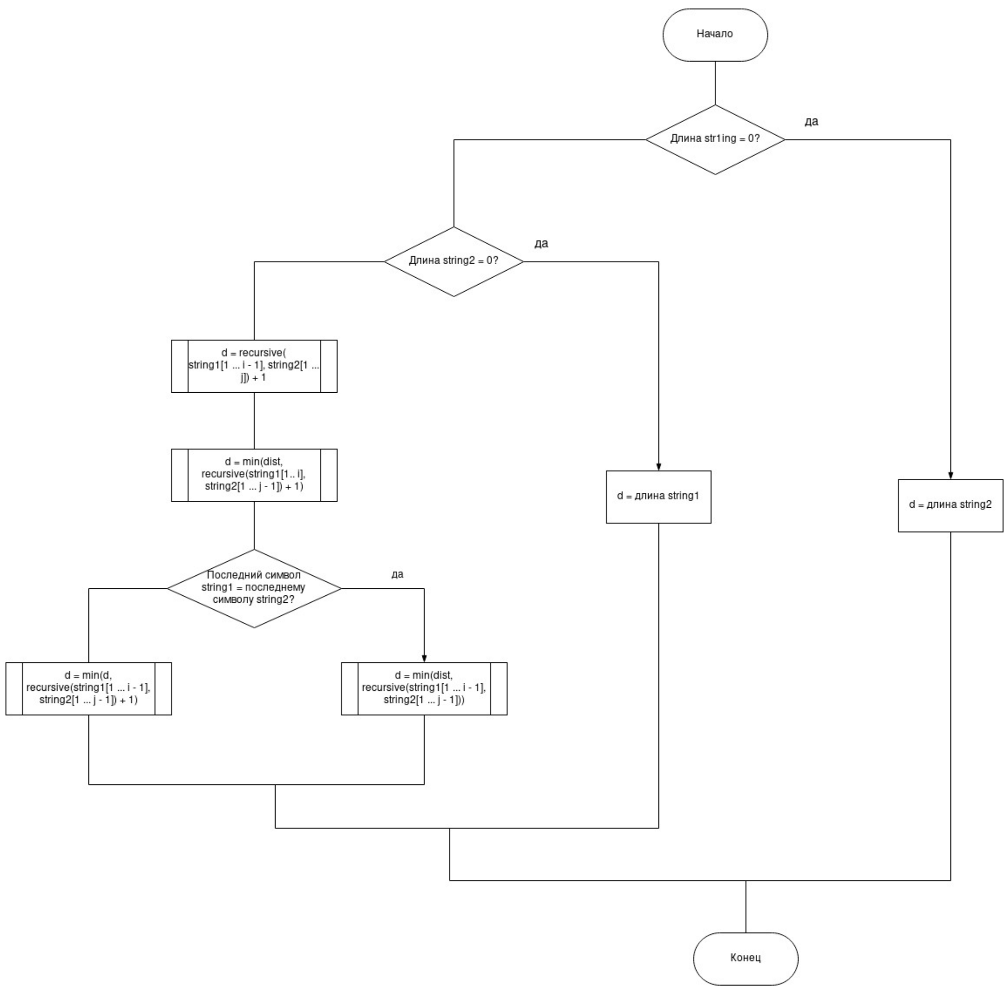

Рис. 2.1: Схема рекурсивного алгоритма нахождения расстояния Левенштейна

2.2 Модель вычислений

2.3 Трудоёмкость алгоритмов

2.3.1 Стандартный алгоритм умножения матриц

2.3.2 Алгоритм Копперсмита — Винограда

2.3.3 Оптимизированный алгоритм Копперсмита — Винограда

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы обоих алгоритмов умножения матриц. Оценены их трудоёмкости в лучшем и худшем случаях.

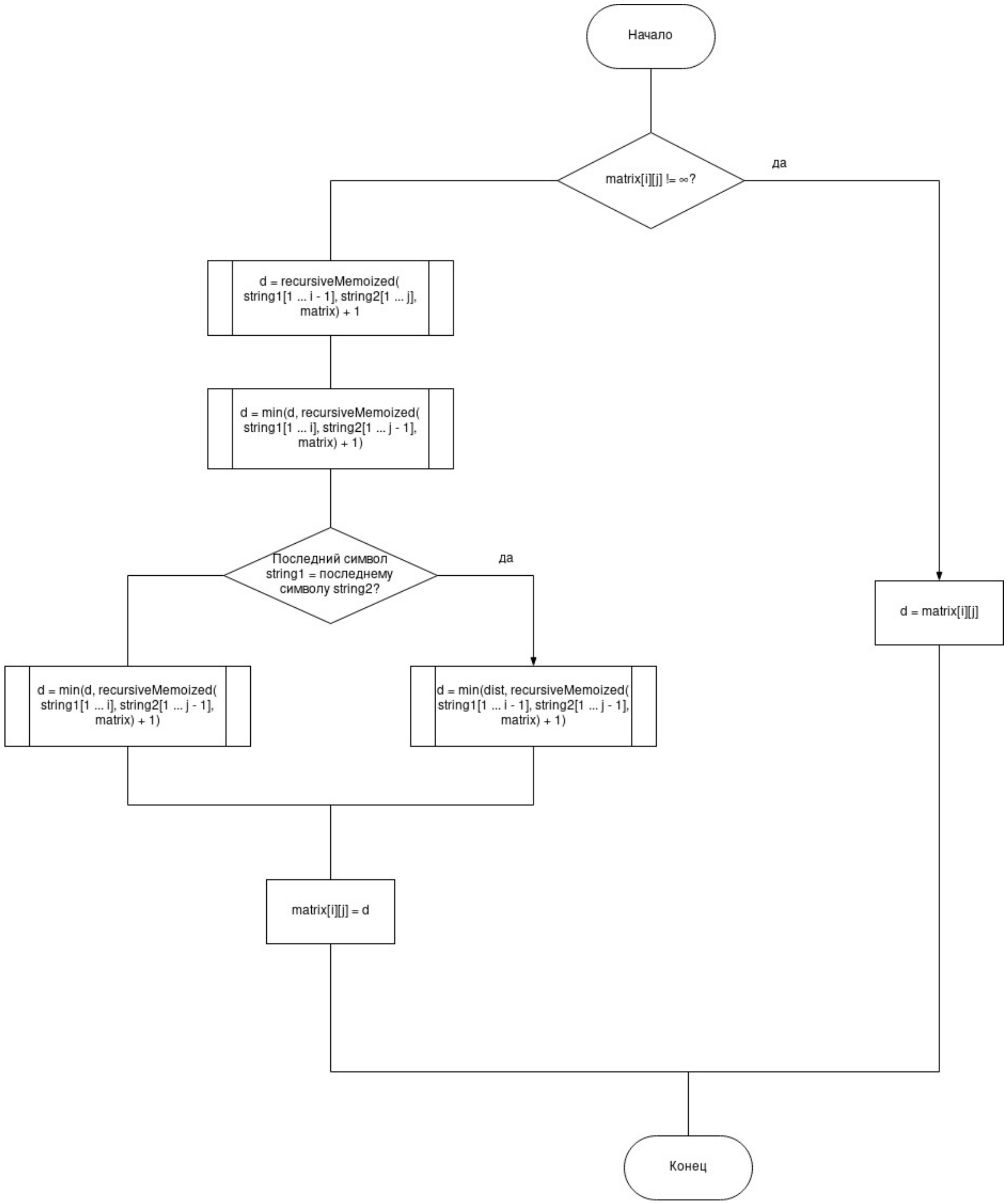

Рис. 2.2: Схема рекурсивного алгоритма с мемоизацией нахождения расстояния Левенштейна

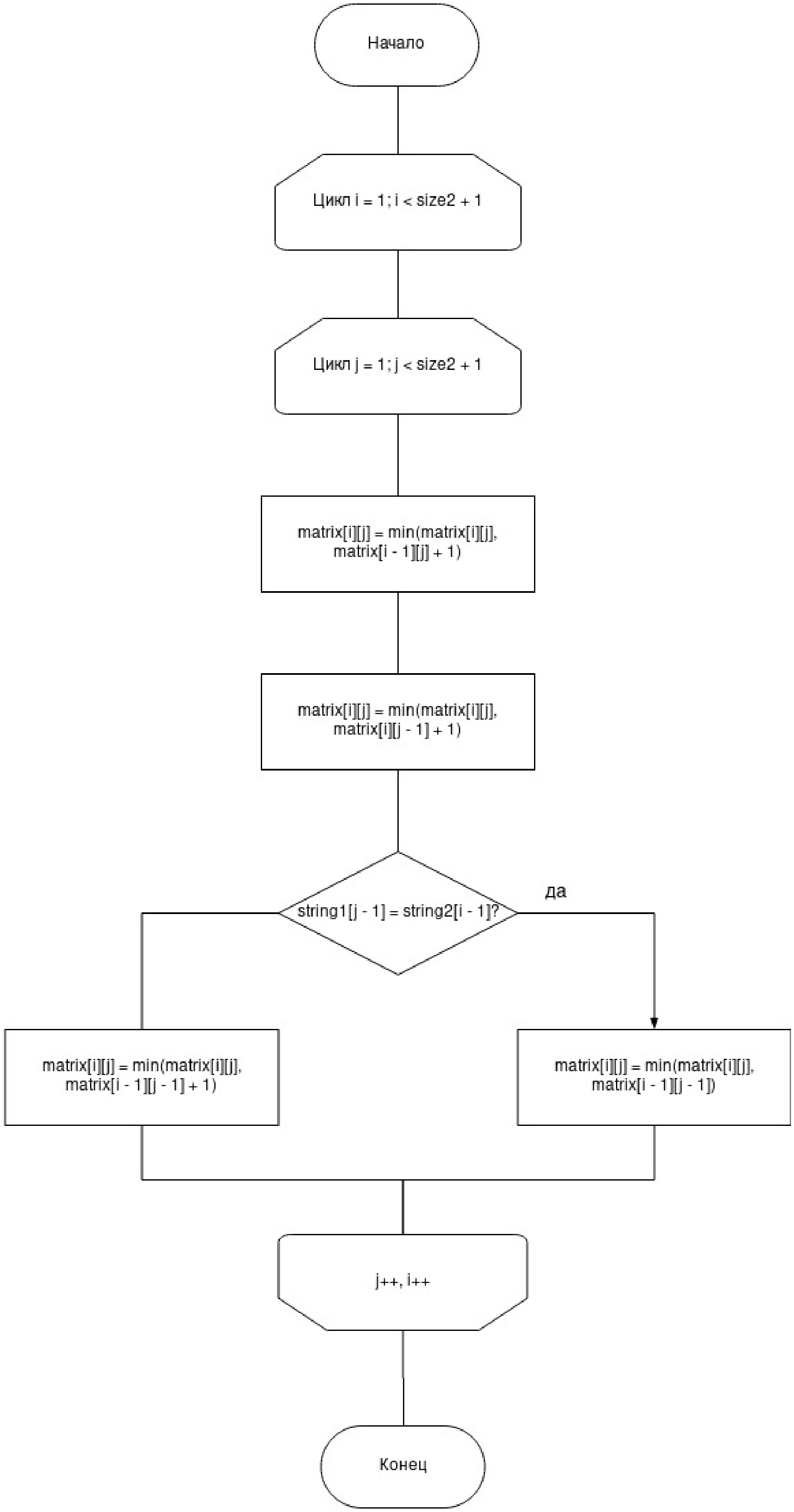

Рис. 2.3: Схема итеративного алгоритма нахождения расстояния Левенштейна

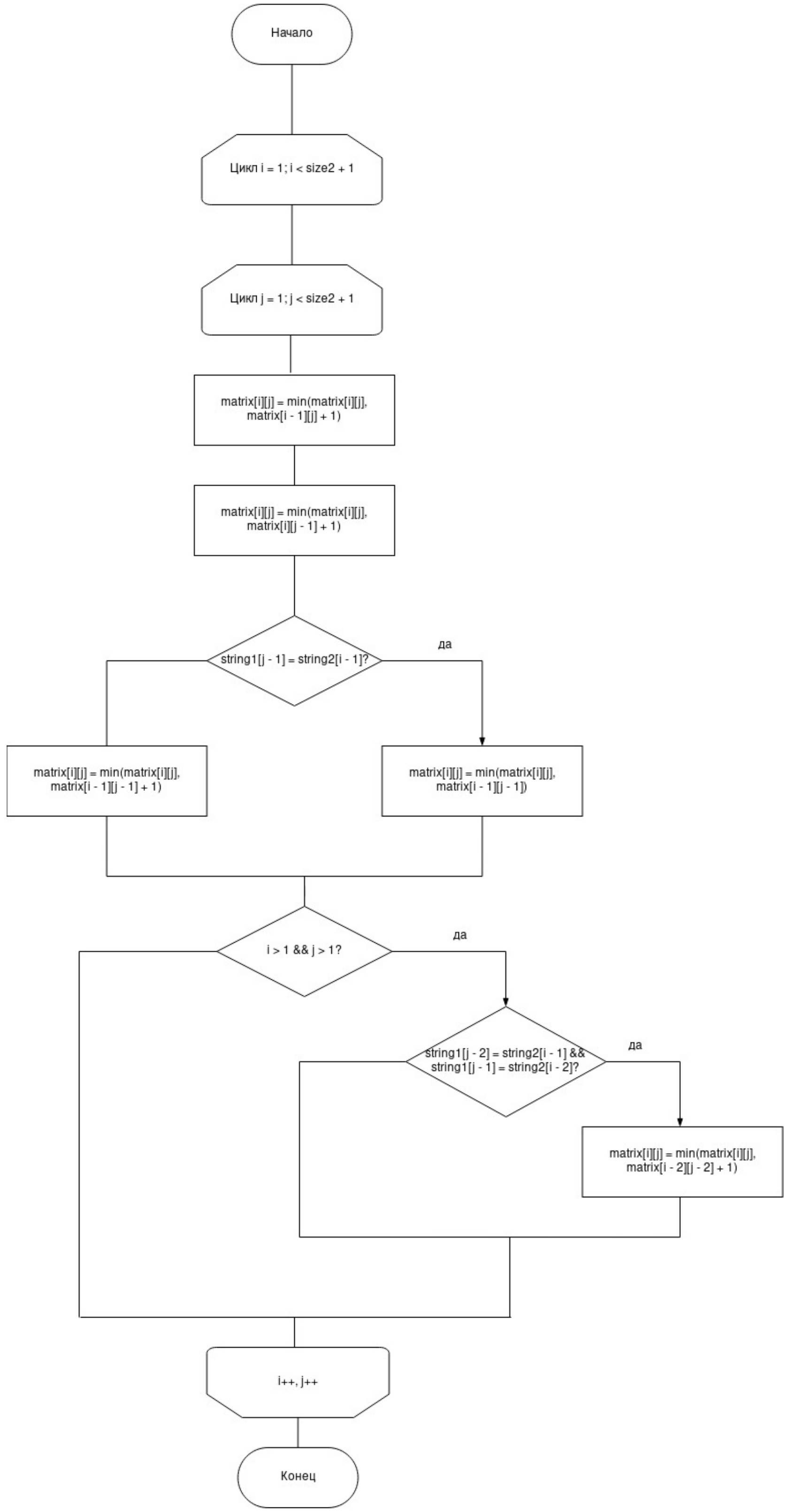

Рис. 2.4: Схема итеративного алгоритма нахождения расстояния Дамерау-Левенштейна

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- На вход ПО получает размеры 2 матриц, а также их элементы;
- На выходе — ПО печатает матрицу, которая является результатом умножения входных матриц.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Haskell [1]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа программирования.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.4 приведена реализация алгоритмов перемножения матриц.

Листинг 3.1: Функция умножения матриц обычным способом

```
baseMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
baseMultiplication m1 m2 = M.fromList (M.ncols m2) (M.nrows m1) $ V.toList $
    _multp m1 m2 1
  where
    _multp m1 m2 i
    | i == M.nrows m1 + 1 = V.fromList []
    | otherwise = P.foldl (\acc j ->
      V.zipWith (+) acc $ V.zipWith (*) (V.fromList $ P.take (M.nrows m1) $
          repeat (M.getElem i j m1)) (M.getRow j m2))
     (V.fromList $ P.take (M.ncols m2) $ repeat 0) [1..M.ncols m1] V.++
         _multp m1 m2 (i + 1)
```

Листинг 3.2: Функция умножения матриц с транспонированием

```
baseTMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
```

```
baseTMultiplication m1 m2 =  M.fromList (M.ncols m2) (M.nrows m1) $ V.toList $ _multp m1 (M.transpose m2) 1
  where
    _multp m1 m2 i
      | i == M.nrows m1 + 1 = V.fromList []
      | otherwise = P.foldl (\acc j ->
        V.snoc acc $ V.sum (V.zipWith (*) (M.getRow i m1) (M.getRow j m2))) (V.fromList []) [1..M.nrows m2]
        V.++ _multp m1 m2 (i + 1)
```

Листинг 3.3: Функция умножения матриц по Винограду

```
winogradMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
winogradMultiplication m1 m2 = res
  where
    a = M.nrows m1
    b = M.ncols m1
    c = M.ncols m2

    rows = V.generate a $ \i -> precalc $ M.getRow (i + 1) m1
    cols = V.generate c $ \j -> precalc $ M.getCol (j + 1) m2

    precalc v = P.foldl (\acc i ->
    acc - V.unsafeIndex v i * V.unsafeIndex v (i + 1)) 0 [0, 2 .. V.length v - 2]

    res = M.matrix a c $ \(i, j) ->
      V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1)
      + subcalc (M.getRow i m1) (M.getCol j m2)
      + if odd b then M.unsafeGet i b m1 * M.unsafeGet b j m2 else 0

    subcalc v1 v2 = P.foldl (\acc i ->
    acc + (V.unsafeIndex v1 (i + 1) + V.unsafeIndex v2 (i))
        * (V.unsafeIndex v1 (i) + V.unsafeIndex v2 (i + 1))) 0 [0, 2 .. V.length v1 - 2]
```

Листинг 3.4: Функция нахождения расстояния Дамерау-Левенштейна матрично

```
winogradOptimizedMultiplication :: (Num a) => Matrix a -> Matrix a -> Matrix a
winogradOptimizedMultiplication m1 m2 = res
  where
    a = M.nrows m1
    b = M.ncols m1
    c = M.ncols m2

    m1' = V.generate a $ \i -> M.getRow (i + 1) m1
    m2' = V.generate c $ \j -> M.getCol (j + 1) m2

    rows = V.generate a $ \i -> precalc $ V.unsafeIndex m1' i
    cols = V.generate c $ \j -> precalc $ V.unsafeIndex m2' j
```

```
    precalc v = P.foldl (\acc i ->
      acc - V.unsafeIndex v i * V.unsafeIndex v (i + 1)) 0 [0, 2 .. b - 2]

    res = if odd b
      then M.matrix a c $ \(i, j) ->
        let v1 = V.unsafeIndex m1' (i - 1)
            v2 = V.unsafeIndex m2' (j - 1)
        in V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1) + subcalc v1 v2 + V.last v1 * V.last v2
      else M.matrix a c $ \(i, j) ->
        let v1 = V.unsafeIndex m1' (i - 1)
            v2 = V.unsafeIndex m2' (j - 1)
        in V.unsafeIndex rows (i - 1) + V.unsafeIndex cols (j - 1) + subcalc v1 v2

    subcalc v1 v2 = P.foldl (\acc i ->
      acc + (V.unsafeIndex v1 (i + 1) + V.unsafeIndex v2 (i))
          * (V.unsafeIndex v1 (i) + V.unsafeIndex v2 (i + 1))) 0 [0, 2 .. b - 2]
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Все тесты пройдены успешно.

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix}1 & 2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}6 & 12 & 18\\ 6 & 12 & 18\\ 6 & 12 & 18\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 2\\ 1 & 2 & 2\end{pmatrix}$	$\begin{pmatrix}1 & 2\\ 1 & 2\\ 1 & 2\end{pmatrix}$	$\begin{pmatrix}5 & 10\\ 5 & 10\end{pmatrix}$
$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$
$\begin{pmatrix}1 & -2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}-1 & 2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}0 & 4 & 6\\ 4 & 12 & 18\\ 4 & 12 & 18\end{pmatrix}$

Таблица 3.1: Тестирование функций

3.5 Вывод

В данном разделе были разработаны исходные коды четырёх алгоритмов перемножения матриц: обычный алгоритм, алгоритм с транспонированием, алгоритм Копперсмита — Винограда, оптимизированный алгоритм Копперсмита — Винограда.

4 Исследовательская часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [2] Linux [3] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [4].

4.2 Время выполнения алгоритмов

Время выполнения алгоритм замерялось с помощью применения технологии профайлинга [5]. Данный инстрмуент даёт детальное описание количества вызовов и количества времени CPU, занятого каждой функцией.

В таблицах 4.1 и 4.2 представлены замеры времени работы для каждого из алгоритмов на чётных размерах матриц.

Таблица 4.1: Таблица времени выполнения алгоритмов при чётных размерах (в секундах)

Размер матрицы	С	ТС	КВ	ОКВ
100	0.482	0.120	0.169	0.158
200	X	X	X	1.253
300	X	X	X	4.063
400	X	X	X	9.909
500	X	X	X	19.357
600	X	X	X	33.265

4.3 Вывод

Тут какой то вывод..

Таблица 4.2: Таблица времени выполнения алгоритмов при нечётных размерах (в наносекундах)

Размер матрицы	C	TC	KB	OKB
10	30500400	1300	650	680
20	NaN	5100	2300	2500
30	NaN	11000	4800	5200
50	NaN	30000	12500	13500
100	NaN	115000	48000	55000
200	NaN	530000	249000	528000

Заключение

В рамках данной лабораторной работы:

1. Были изучены и реализованы 3 алгоритма перемножения матриц: обычный, Копперсмита – Винограда, оптимизированный Копперсмита – Винограда;
2. Был произведён анализ трудоёмкости алгоритмов на основе теоретических расчётов и выбранной модели вычислений;
3. Был сделан сравнительный анализ алгоритмов на основе экспериментальных данных;

Литература

[1] The Haskell purely functional programming language [Электронный ресурс]. Режим доступа: https://haskell.org/. Дата обращения: 16.09.2020.

[2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.09.2020.

[3] Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 20.09.2020.

[4] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html. Дата обращения: 20.09.2020.

[5] Profiling – Haskell profiling [Электронный ресурс]. Режим доступа: https://downloads.haskell.org/ghc/8.8.1/docs/html/user_guide/index.html. Дата обращения: 20.09.2020.