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»

ОТЧЕТ

По лабораторной работе №1

По курсу: «Анализ алгоритмов»

Тема: «Расстояние Левенштейна и Дамерау-Левенштейна»

Студент: Ле Ни Куанг

Группа: ИУ7и-56Б

Преподаватель: Волкова Л. Л.

Строганов Ю. В.

Москва

2020

Оглавление

Введение

Расстояние Левенштейна - метрика, измеряющая разность между двумя последовательностями символов, или по-другому это минимальное количество односимвольных операций (вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую.

Расстояние Левенштейна применяется:

- для исправления ошибок в слове
- для сравнения текстовых файлов утилитой diff и ей подобными
- в биоинформатике для сравнения генов, хромосом и белков

Целью работы: изучение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна.

Задачи работы:

1. изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
2. применение метода динамического программирования для матричной реализации указанных алгоритмов;
3. получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
4. сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);

5. экспериментальное подтверждение различий во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;
6. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

В данном разделе будет приведено описание алгоритмов и формулы для нахождения расстояния Левенштейна и Дамерау-Левенштейна.

1.1 Описание алгоритмов

1.1.1 Расстояние Левенштейна

Расстояние Левенштейна определяет минимальное количество операций, необходимых для превращения одной последовательности символов в другую. Разрешенные действия:

- вставка (I - insert)
- удаление (D - delete)
- замена (R - replace)

Расстояние Левенштейна между двумя строками a, b задается выражением $lev_{a,b}(|a|, |b|)$ (1.1)

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & if\ min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_i)} \end{cases} & otherwise \end{cases} \tag{1.1}$$

где $1_{(a_i \neq b_i)} = \begin{cases} 0 & if\ a_i = b_i \\ 1 & otherwise \end{cases}$

$lev_{a,b}(i,j)$ - расстояние между первыми i символами строки a и первыми j символами строки b

1.1.2 Расстояние Дамерау-Левенштейна

Если к списку разрешённых операций расстояния Левенштейна добавить транспозицию (два соседних символа меняются местами), получается расстояние Дамерау — Левенштейна.

- + транспозицию (T - transposition)

Расстояние Дамерау-Левенштейна между двумя строками a, b задается выражением $d_{a,b}(|a|, |b|)$ (1.2)

$$d_{a,b}(i,j) = min \begin{cases} 0 & if\ i = j = 0 \\ d_{a,b}(i-1,j)+1 & if\ i > 0 \\ d_{a,b}(i,j-1)+1 & if\ j > 0 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_i)} & if\ i,j > 0 \\ \begin{cases} d_{a,b}(i-2,j-2)+1 \\ \\ +\infty \quad (i+j) \end{cases} & \begin{array}{l} if\ i,j > 1\ and \\ a[i] = b[j-1]\ and\ a[i-1] = b[j] \\ \\ \end{array} \end{cases} \tag{1.2}$$

где $1_{(a_i \neq b_i)} = \begin{cases} 0 & if\ a_i = b_i \\ 1 & otherwise \end{cases}$

Каждый рекурсивный вызов соответствует одному из случаев:

- $d_{a,b}(i-1,j)+1$ соответствует удалению символа (из a в b)
- $d_{a,b}(i,j-1)+1$ соответствует вставке (из a в b)
- $d_{a,b}(i-1,j-1)+1_{(a_i \neq b_i)}$ соответствие или несоответствие, в зависимости от совпадения символов
- $d_{a,b}(i-2,j-2)+1$ в случае перестановки двух последовательных символов

2 Конструкторский раздел

В данном разделе будет приведено описание схем алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна

2.1 Разработка алгоритмов

На рисунках показаны схемы алгоритмов Левенштейна рекурсивная, матричная, рекурсивная реализация с заполнением матрицы и схема алгоритма Дамерау–Левенштейна (матричная).

Примечание: я создаю таблицы и инициализирую значение вне этих функции

2.1.1 Схема алгоритма Левенштейна

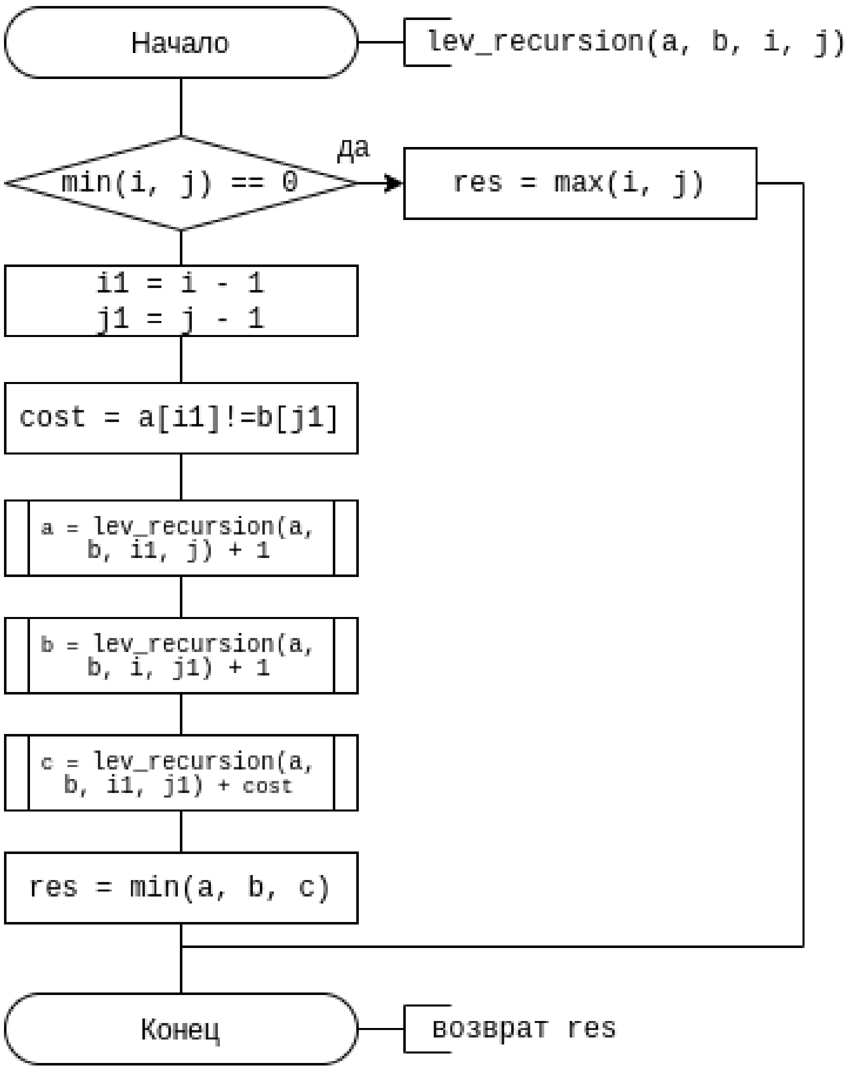

Рис. 2.1: Схема рекурсивного алгоритма Левенштейна

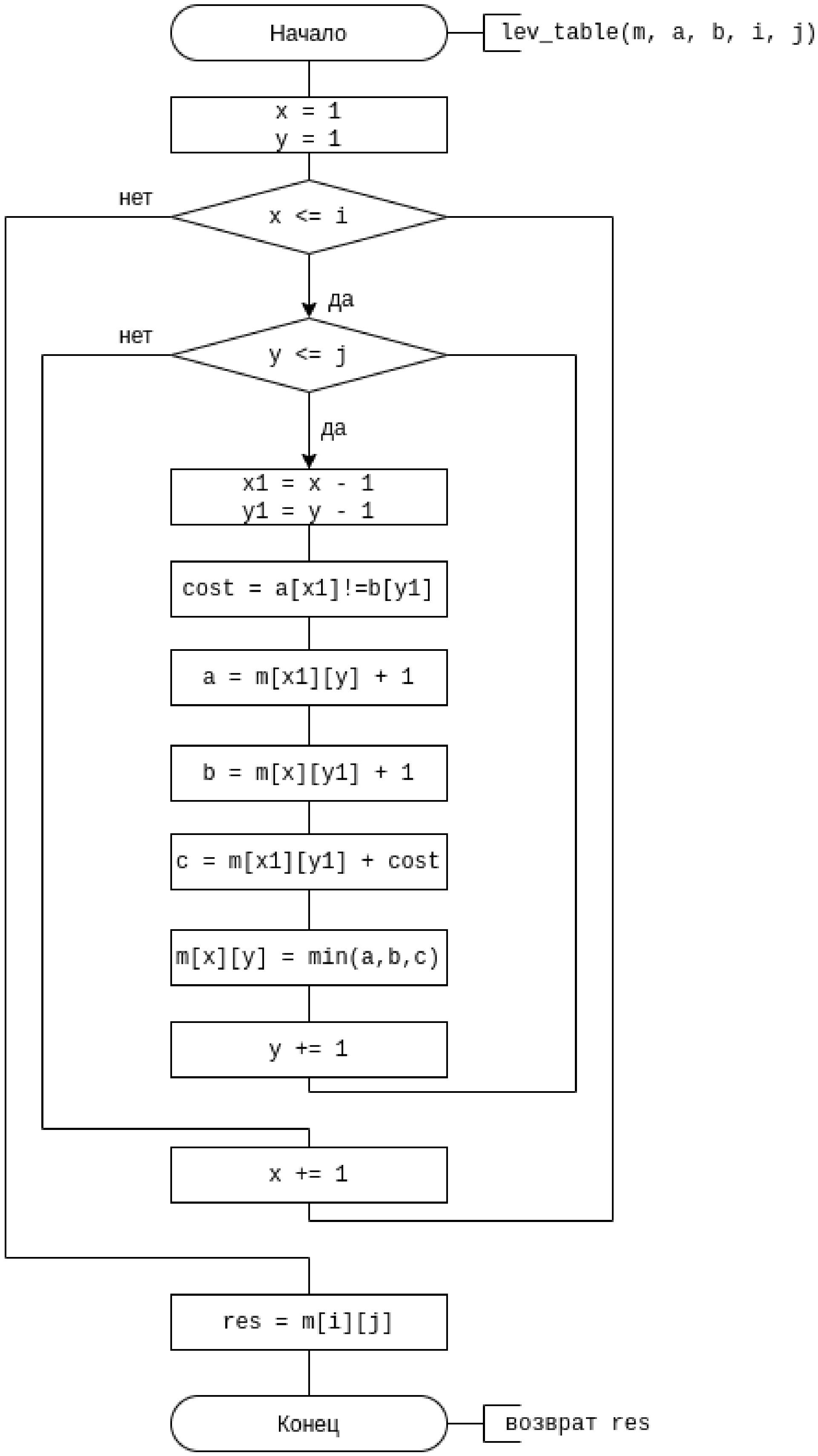

Рис. 2.2: Схема матричного алгоритма Левенштейна

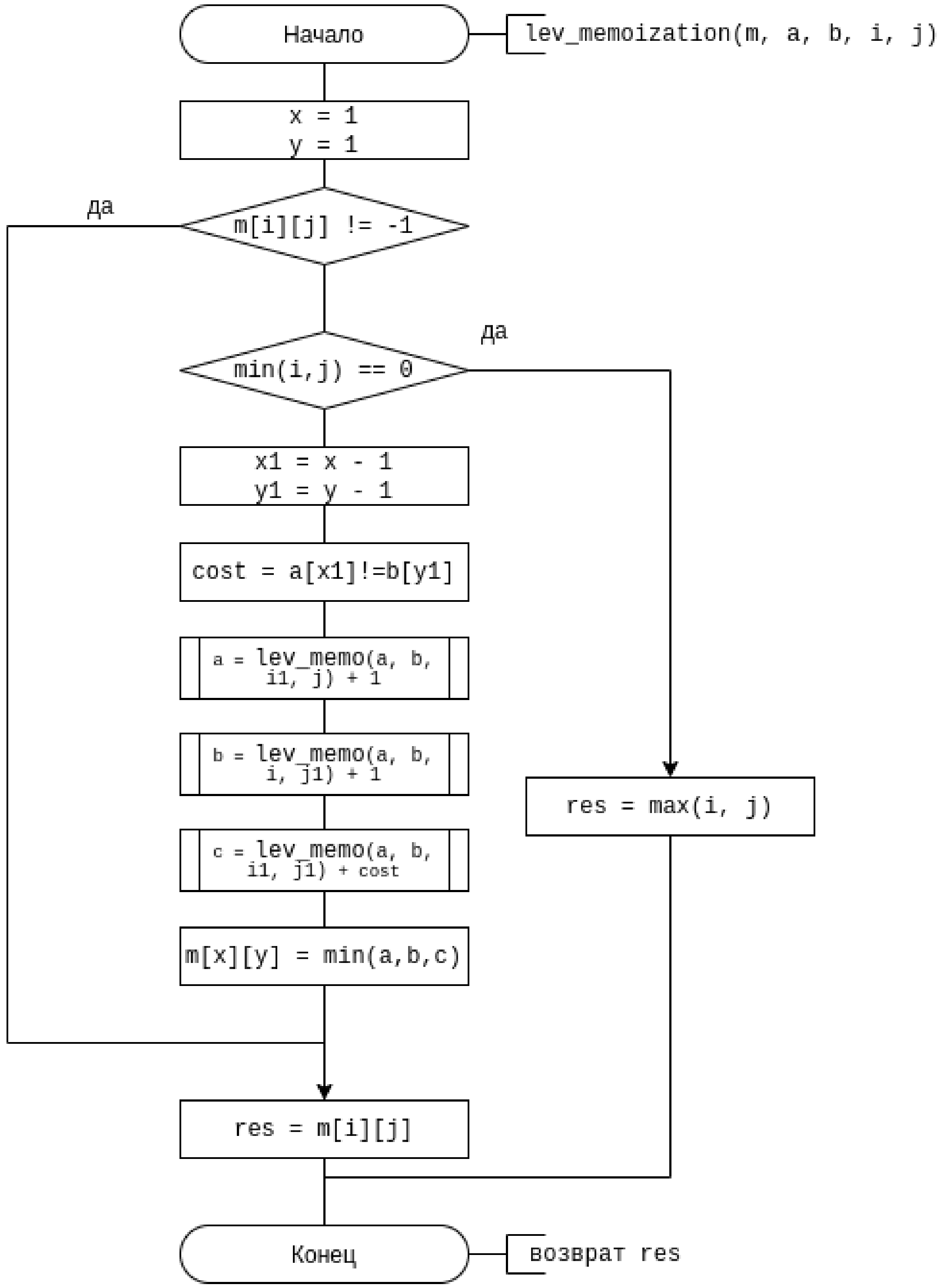

Рис. 2.3: Схема мемоизационного алгоритма Левенштейна

2.1.2 Схема алгоритма Дамерау — Левенштейна

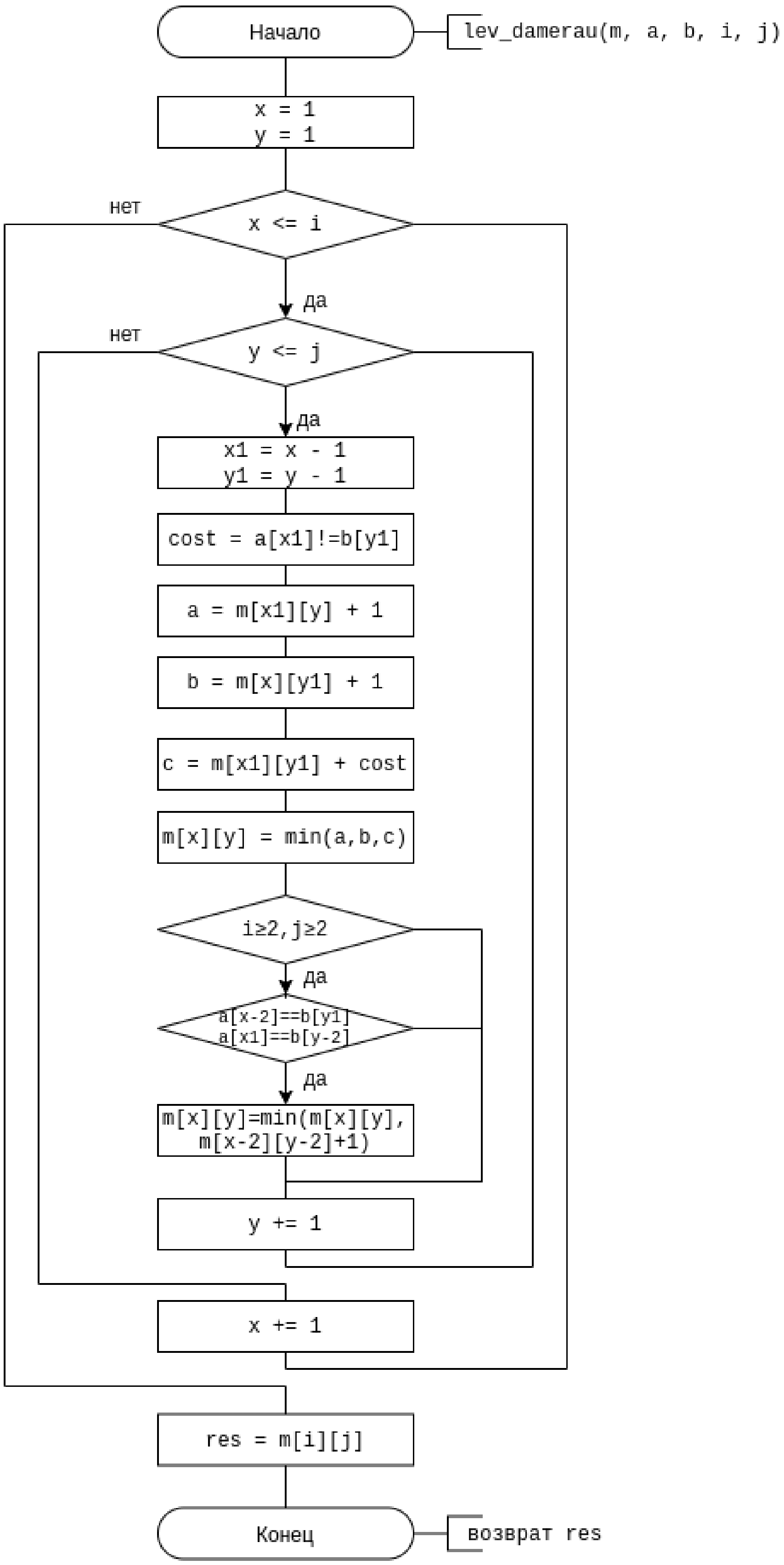

Рис. 2.4: Схема матричного алгоритма Дамерау — Левенштейна

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа создана в формате записной книжки, в интерактивном режиме пользователь вводит команду в соответствии с инструкциями. ПО должно иметь сравнение времени работы алгоритмов и должен быть хорошо протестирован. (ПО может быть легко использован программистом)

3.2 Средства реализации

Язык программирования: Python (IPython)

Библиотеки: unittest, timeit, matplotlib, ...

Редактор: Jupyter-Lab

Я использую эти инструменты потому, что они мощные, широко используемые и знакомые мне.

3.3 Листинг кода

Примечание: я создаю таблицы и инициализирую значение в классе Lev - interface

Листинг 3.1: Рекурсивная реализация алгоритма Левенштейна

```
def lev_recursion(a, b, i, j):
    if min(i, j) == 0:
        return max(i, j)

    i1 = i - 1
    j1 = j - 1
    return min(lev_recursion(a, b, i1, j) + 1,
               lev_recursion(a, b, i, j1) + 1,
               lev_recursion(a, b, i1, j1) + (0 if a[i1] == b[j1] else 1))
```

Листинг 3.2: Матричная реализация алгоритма Левенштейна

```python
def lev_table(m, a, b, i, j):
    for x in range(1, i+1):
        for y in range(1, j+1):
            x1 = x - 1
            y1 = y - 1
            m[x][y] = min(m[x1][y] + 1,
                          m[x][y1] + 1,
                          m[x1][y1] + (0 if a[x1] == b[y1] else 1))

    return m[i][j]
```

Листинг 3.3: Рекурсивная реализация алгоритма Левенштейна с заполнением матрицы

```python
def lev_memoization(m, a, b, i, j):
    if m[i][j] != -1:
        return m[i][j]

    if min(i, j) == 0:
        return max(i, j)

    i1 = i - 1
    j1 = j - 1
    r = min(lev_memoization(m, a, b, i1, j) + 1,
            lev_memoization(m, a, b, i, j1) + 1,
            lev_memoization(m, a, b, i1, j1) + (0 if a[i1] == b[j1] else
                1))

    m[i][j] = r
    return r
```

Листинг 3.4: Матричная реализация алгоритма Дамерау-Левенштейна

```python
def lev_damerau(m, a, b, i, j):
    for x in range(1, i+1):
        for y in range(1, j+1):
            x1 = x - 1
            y1 = y - 1
            m[x][y] = min(m[x1][y] + 1,
                          m[x][y1] + 1,
                          m[x1][y1] + (0 if a[x1] == b[y1] else 1))

            if i > 1 and j > 1 and a[x1]==b[y-2] and a[x-2]==b[y1]:
                m[x][y] = min(m[x][y], m[x-2][y-2] + 1)

    return m[i][j]
```

Листинг 3.5: Класс интерфейса

```
class Lev:
    def __init__(self):
        self.m = []

    def recursion(self, a, b):
        return lev_recursion(a, b, len(a), len(b))

    def table(self, a, b):
        i = len(a)
        j = len(b)
        self.m = [[x + y for x in range(j + 1)] for y in range(i + 1)]
        return lev_table(self.m, a, b, i, j)

    def memoization(self, a, b):
        i = len(a)
        j = len(b)
        self.m = [[-1 for _ in range(j + 1)] for _ in range(i + 1)]
        return lev_memoization(self.m, a, b, i, j)

    def damerau(self, a, b):
        i = len(a)
        j = len(b)
        self.m = [[x + y for x in range(j + 1)] for y in range(i + 1)]
        return lev_damerau(self.m, a, b, i, j)


    def debug(self, a, b):
        print("{:>8}\t{:>8}".format(a, b), end="\t")
        print('{:2d} {:2d} {:2d} {:2d}'.
              format(self.recursion(a, b), self.table(a, b),
                 self.memoization(a, b), self.damerau(a, b)))

    def print_table(self):
        if len(self.m):
            for x in range(len(self.m)):
                for y in range(len(self.m[0])):
                    print('{:3d} '.format(self.m[x][y]), end='')
                print()
        print()
```

3.4 Описание тестирования

В таблице 3.1 приведен функциональные тесты для алгоритмов вычисления расстояния Левенштейна и Дамерау — Левенштейна.

Строка 1	**Строка 2**	**Ожидаемый результат**
		0 0
abc	abc	0 0
abc	bc	1 1
de	def	1 1
abc	acb	2 1
	abcd	4 4
abcd		4 4
kitten	sitting	3 3
telo	ctolb	3 3
python	pyhton	2 1
pattern	state	5 4
writer	nation	6 5
large	already	6 5
trouble	foreign	7 6
fact	way	3 3
east	time	4 4
spend	move	5 5
prevent	player	5 5
way	top	3 3

Таблица 3.1: Функциональные тесты

4 Экспериментальный раздел

4.1 Примеры работы

На рисунке приведен пример работы программы. В соответствующем порядке будут: рекурсивная, матричная, рекурсивная реализация с заполнением матрицы Левенштейна и итеративная реализация Дамерау–Левенштейна.

```
   abc             bc         1  1  1  1
   abc            abc         0  0  0  0
  telo          ctolb         3  3  3  3
python         pyhton         2  2  2  1
kitten        sitting         3  3  3  3


'python', 'pyhton'

  0   1   2   3   4   5   6
  1   0   1   2   3   4   5
  2   1   0   1   2   3   4
  3   2   1   1   1   2   3
  4   3   2   1   2   2   3
  5   4   3   2   2   2   3
  6   5   4   3   3   3   2

 -1  -1  -1  -1  -1  -1  -1
 -1   0   1   2   3   4   5
 -1   1   0   1   2   3   4
 -1   2   1   1   1   2   3
 -1   3   2   1   2   2   3
 -1   4   3   2   2   2   3
 -1   5   4   3   3   3   2

  0   1   2   3   4   5   6
  1   0   1   2   3   4   5
  2   1   0   1   2   3   4
  3   2   1   1   1   2   3
  4   3   2   1   1   2   3
  5   4   3   2   2   1   2
  6   5   4   3   3   2   1
```

Рис. 4.1: Примеры работы алгоритмов нахождения растоянии Левенптейна и ДамерауЛевенштейна

4.2 Результаты тестирования

Использование фреймворка модульного тестирования: unittest

```
test_damerau (__main__.TestLevenshtein) ... ok
test_memoization (__main__.TestLevenshtein) ... ok
test_recursion (__main__.TestLevenshtein) ... ok
test_table (__main__.TestLevenshtein) ... ok

----------------------------------------------------
Ran 4 tests in 0.024s

OK
```

4.3 Постановка эксперимента по замеру времени и памяти

Операционная система - Ubuntu 20.04.1 LTS

Процессор - Intel® Core™ i5-7300HQ CPU @ 2.50GHz × 4

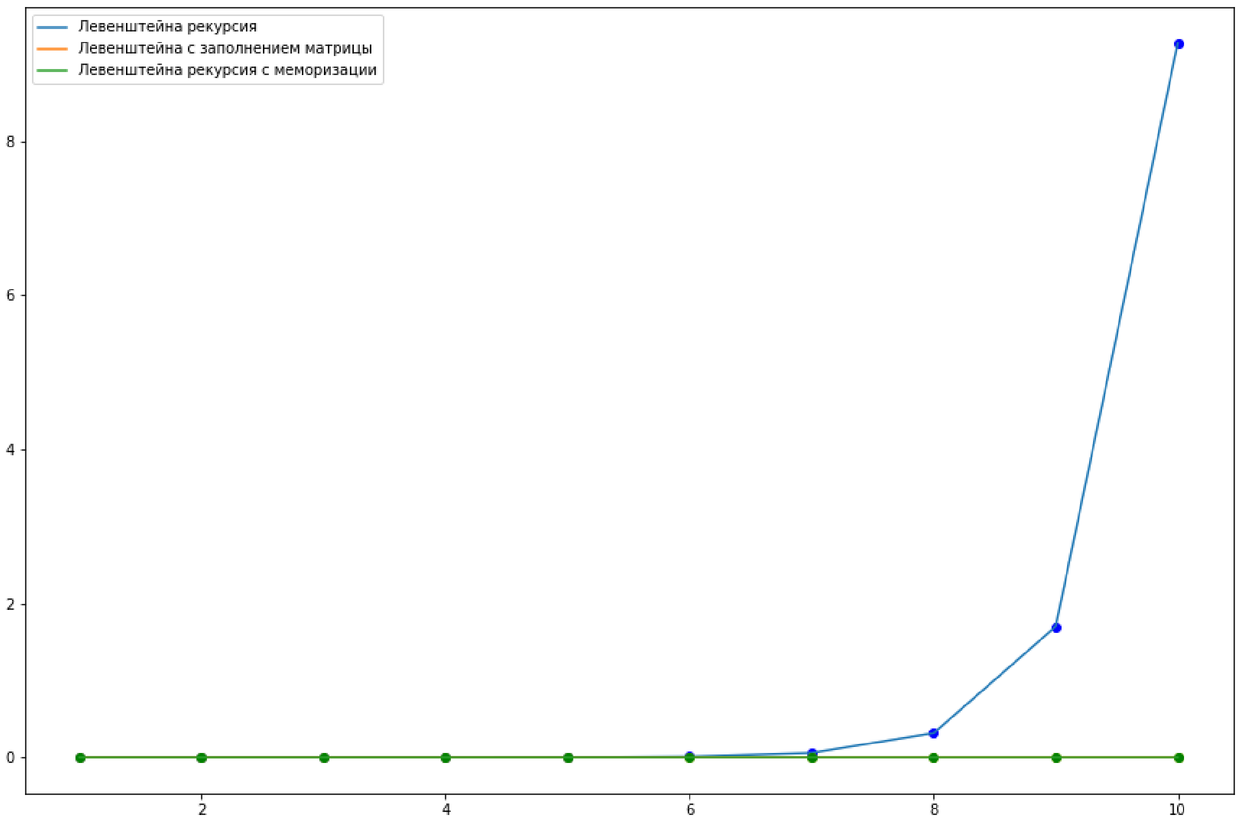

Рис. 4.2: Сравнение времени работы алгоритмов Левенштейна

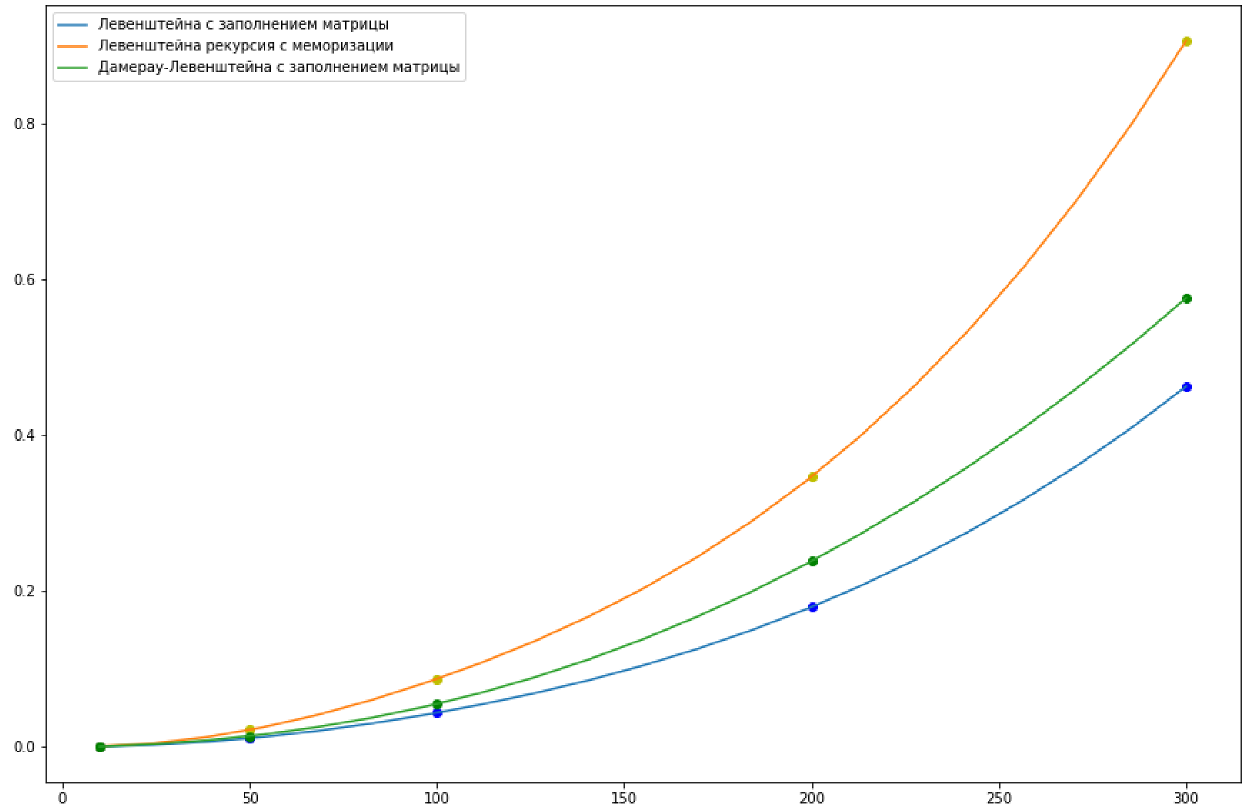

Рис. 4.3: Зависимость времени работы алгоритмов Левенштейна и Дамерау — Левенштейна

Использование памяти

Алгоритмы Левенштейна и Дамерау – Левенштейна не отличаются друг от друга с точки зрения использования памяти, следовательно, достаточно рассмотреть лишь разницу рекурсивной и матричной реализаций этих алгоритмов Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк. Максимальный расход памяти:

$$(len(S_1) + len(S_2)) \cdot (2 \cdot S(reference) + 7 \cdot S(number)) \tag{4.1}$$

где S – оператор вычисления размера, S_1, S_2 – строки

Использование памяти при итеративной реализации теоритически равно:

$$((len(S_1) + 1) \cdot (len(S_2) + 1) + 9) \cdot S(number) + 3 \cdot S(reference) \tag{4.2}$$

Заключение

Вывод лаборатории состоит в том, что существует множество алгоритмов, решающих одну и ту же проблему с очень разным временем, в частности, рекурсивный алгоритм может работать только с короткими строками, время быстро увеличивается с увеличением длины. В ходе работы был изучен метод динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна для нахождения расстояния между строками, получены практические навыки реализации указанных алгоритмов в матричной и рекурсивных версиях.

Литература

[1] В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.

[2] Гасфилд. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, 2003.

[3] Understanding the Levenshtein Distance Equation for Beginners
`https://medium.com/@ethannam/understanding-the-levenshtein-distance`

[4] unittest — Unit testing framework
`https://docs.python.org/3/library/unittest.html`