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 Фам Минь Хиеу

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л., Строганов Д.В.

Москва — 2023 г.

Оглавление

Введение

Целью данной лабораторной работы является описание, реализация и исследование алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна.

Расстояние Левенштейна – это минимальное количество односимвольных операций (вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую.

Расстояние Дамерау-Левенштейна – модификация расстояния Левештейна. Это минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую.

Расстояния Левенштейна и Дамерау-Левенштейна широко применяются для решения задач компьютерной лингвистики (исправление ошибок в слове, автоматическое распознавание отсканированного текста или речи), биоинформатики (для сравнения генов) и других.

В рамках выполнения лабораторной работы необходимо решить следующие задачи:

— описать расстояния Левенштена и Дамерау-Левенштейна;

— построить схемы алгоритмов следующих методов: нерекурсивный метод поиска расстояния Левенштейна, нерекурсивный метод поиска Дамерау-Левенштейна, рекурсивный метод поиска Дамерау-Левенштейна, рекурсивный с кешированием метод поиска Дамерау-Левенштейна;

— создать ПО, реализующее перечисленные выше алгоритмы;

— сравнить алгоритмы определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна

1.1 Нерекурсивный алгоритм нахождения расстояния Левенштейна

Расстояние Левенштейна между двумя строками — это минимальное количество операций вставки, удаления и замены, необходимых для превращения одной строки в другую.

Каждая операция имеет свою цену (штраф). Редакционным предписанием называется последовательность действий, необходимых для получения из первой строки второй, и минимизирующих суммарную цену. Суммарная цена есть искомое расстояние Левенштейна.

Введем следующие обозначения операций с указанием соответсвующих штрафов w:

— D (англ. delete) — удаление ($w(a, \lambda) = 1$);

— I (англ. insert) — вставка ($w(\lambda, b) = 1$);

— R (англ. replace) — замена ($w(a, b) = 1,\ a \neq b$);

— M (англ. match) — совпадение ($w(a, a) = 0$).

Пусть S_1 и S_2 — две строки длиной M и N соответственно, тогда расстояние Левенштейна можно подсчитать по рекуррентной формуле 1.1.

$$D(i,j) = \begin{cases} 0, \text{ если i} = 0, \text{ j} = 0, \\ i, \text{ если j} = 0, \text{ i} > 0, \\ j, \text{ если i} = 0, \text{ j} > 0, \\ \min\{ \\ \quad D(i, j-1) + 1 \\ \quad D(i-1, j) + 1 \\ \quad D(i-1, j-1) + m(a[i], b[j]) \\ \}, \text{ если i} > 0, \text{ j} > 0, \end{cases} \tag{1.1}$$

где m определяется следующим образом:

$$m(a,b) = \begin{cases} 0 & \text{если a} = \text{b}, \\ 1 & \text{иначе.} \end{cases}. \tag{1.2}$$

С ростом i, j прямая рекурсивная реализация формулы (1.1) становится малоэффективной по времени исполнения, так как множество промежуточных значений $D(i,j)$ вычисляются не по одному разу. Для решения этой проблемы можно использовать матрицу для хранения соответствующих промежуточных значений.

Чтобы исключить повторные вычисления, будет использоваться матрица размером $(length(S1)+1)$x$((length(S2)+1)$, где $length(S)$ – длина строки S. Значение в ячейке $[i,j]$ равно значению $D(S1[1...i], S2[1...j])$.

Вся таблица (за исключением первого столбца и первой строки) заполняется в соответствии с формулой (1.3).

$$A[i][j] = min \begin{cases} A[i-1][j] + 1 \\ A[i][j-1] + 1 \\ A[i-1][j-1] + m(S1[i], S2[j]) \end{cases}. \tag{1.3}$$

Функция m определена как:

$$m(S1[i], S2[j]) = \begin{cases} 0, & \text{если } S1[i-1] = S2[j-1], \\ 1, & \text{иначе} \end{cases}. \tag{1.4}$$

В результате расстоянием Левенштейна будет ячейка матрицы с индексами $i = length(S1)$ и $j = length(S2)$ при учете, что индексы начинаются с 0.

1.2 Нерекурсивный алгоритм поиска Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна – это минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна с добавлением операции транспозиции (перестановки) соседних символов.

Расстояние Дамерау-Левенштейна может быть найдено по формуле:

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), \text{ если } \min(i,j) = 0, \\ \min\{ \\ \quad d_{a,b}(i,j-1) + 1, \\ \quad d_{a,b}(i-1,j) + 1, \\ \quad d_{a,b}(i-1,j-1) + m(a[i],b[j]), \text{ иначе} \\ \quad \begin{cases} d_{a,b}(i-2,j-2) + 1, \text{ если } i,j > 1; \\ \qquad a[i] = b[j-1]; \\ \qquad b[j] = a[i-1] \\ \qquad \infty, \text{ иначе} \end{cases} \\ \} \end{cases}, \tag{1.5}$$

1.3 Рекурсивный алгоритм поиска Дамерау-Левенштейна

Рекурсивный алгоритм поиска расстояния Дамерау-Левенштейна решает ту же задачу, что его нерекурсивная версия. Отличие лишь в том, что в данном случае не используется матрица, так как все значения вычисляются рекурсивно.

1.4 Рекурсивный с кешированием алгоритм поиска Дамерау-Левенштейна

Рекурсивный алгоритм заполнения можно оптимизировать по времени выполнения с использованием кеша. В качестве кеша используется матрица. Суть данной оптимизации заключается в параллельном заполнении матрицы при выполнении рекурсии. В случае, если рекурсивный алгоритм выполняет прогон для данных, которые еще не были обработаны, результат нахождения расстояния заносится в матрицу. В случае, если обработанные ранее данные встречаются снова, для них расстояние не находится и алгоритм переходит к следующему шагу.

Вывод

В данном разделе были рассмотрены алгоритмы поиска расстояния Левенштейна и расстояния Дамерау-Левенштейна. В частности были приведены рекурентные формулы работы алгоритмов, объяснена разница между расстоянием Левенштейна и расстоянием Дамерау-Левенштейна.

2 Конструкторская часть

В этом разделе будут приведены требования к вводу и программе, а также схемы алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна.

2.1 Требования к вводу

На вход подаются две строки, причем буквы верхнего и нижнего регистров считаются различными.

2.2 Требования к программе

При вводе двух пустых строк программа не должна аварийно завершатся. Вывод программы - число (расстояние Левенштейна или Дамерау-Левенштейна), для алгоритмов, использующих матрицу для расчёта требуется вывести заполненную матрицу.

2.3 Разработка алгоритма поиска расстояния Левенштейна

На рисунке 2.1 приведена схема нерекурсивного алгоритма нахождения расстояния Левенштейна.

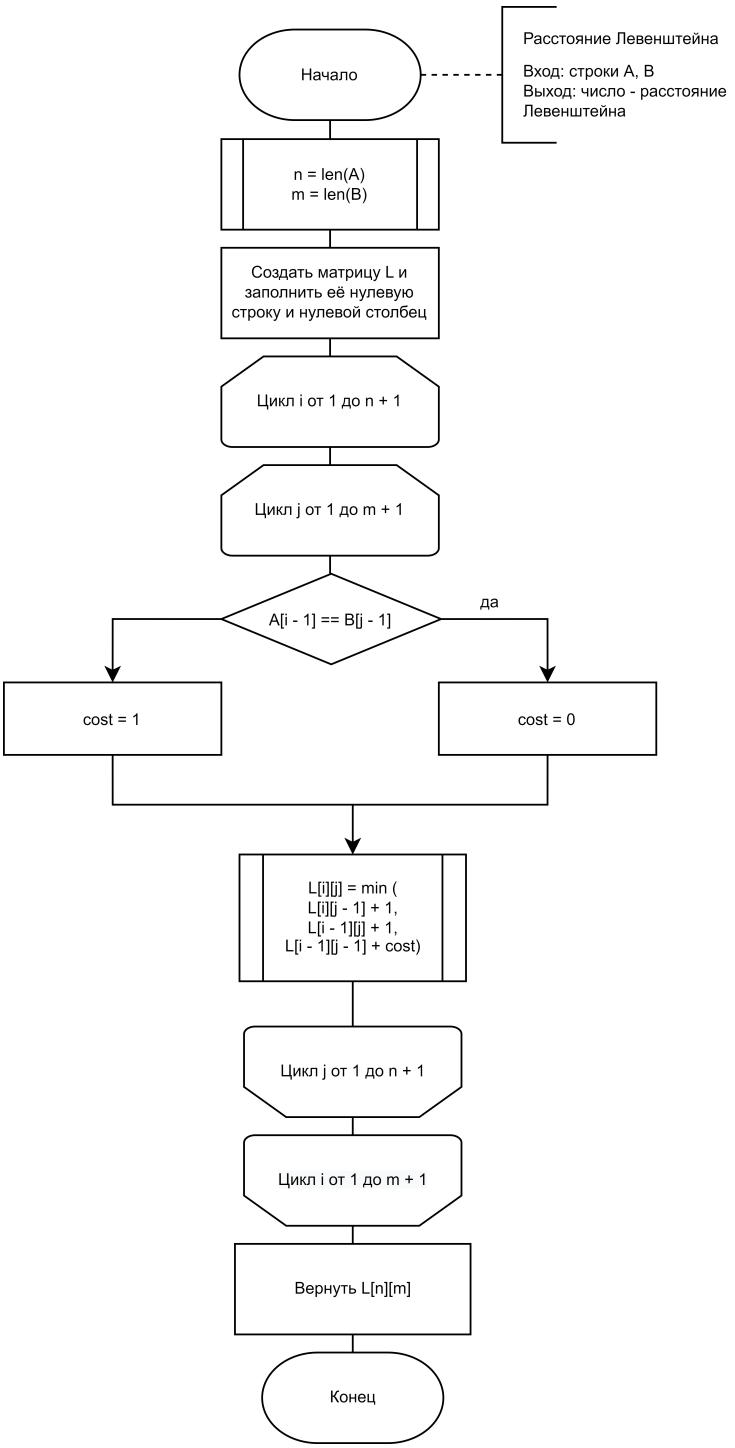

Рисунок 2.1 – Схема нерекурсивного алгоритма нахождения расстояния Левенштейна

2.4 Разработка алгоритмов поиска расстояния Дамерау-Левенштейна

На рисунке 2.2 приведена схема нерекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна.

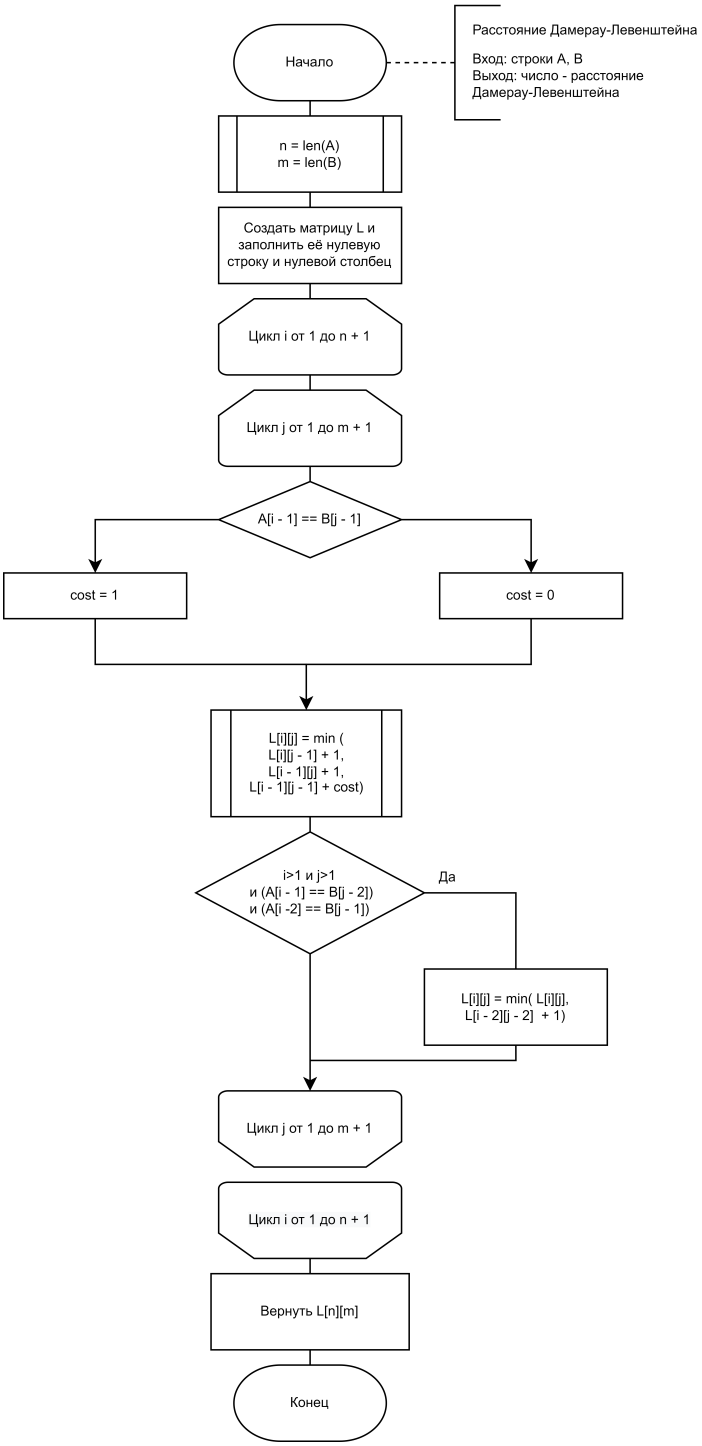

Рисунок 2.2 – Схема нерекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна

На рисунке 2.3 приведена схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна.

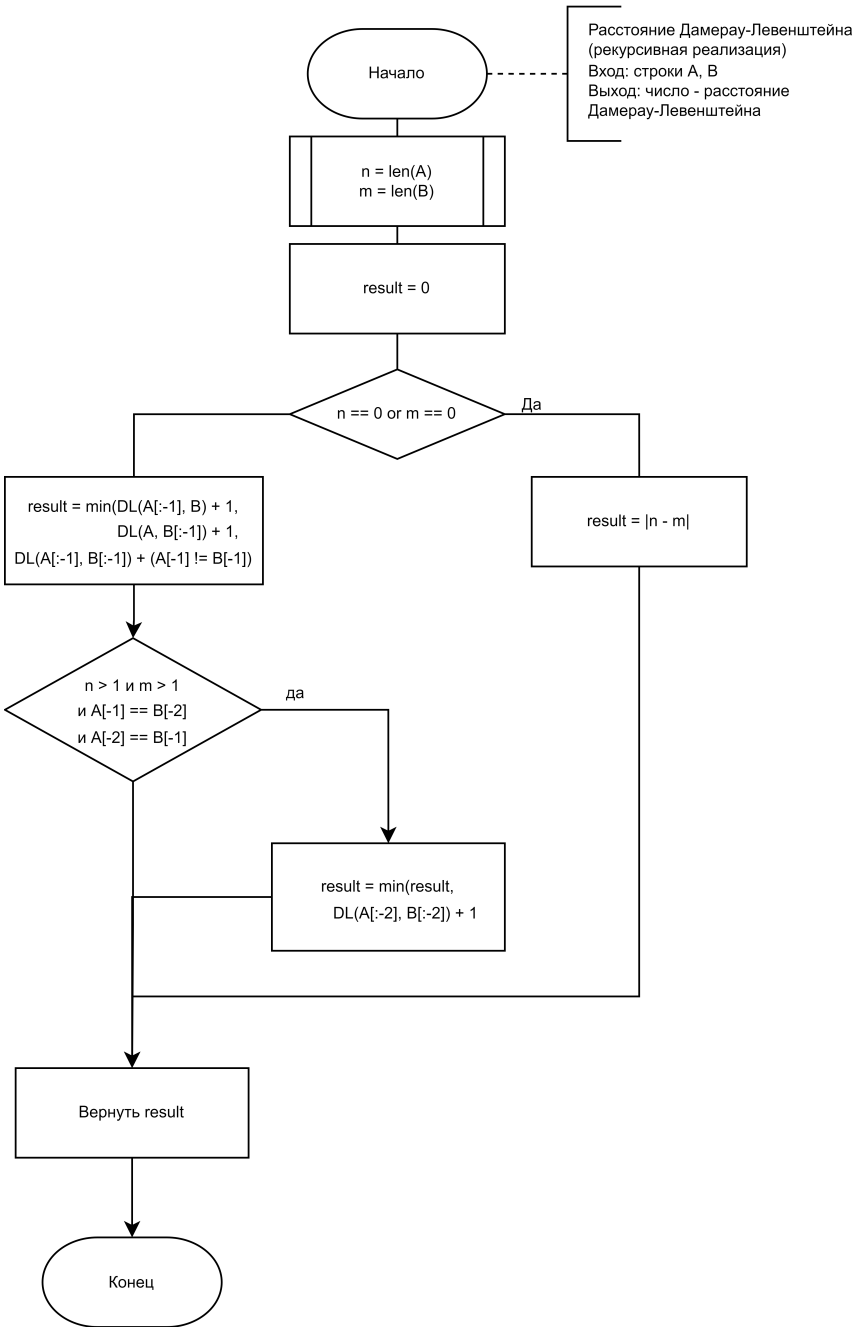

Рисунок 2.3 – Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна

На рисунке 2.4 приведена схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна с использованием кеша в виде матрицы.

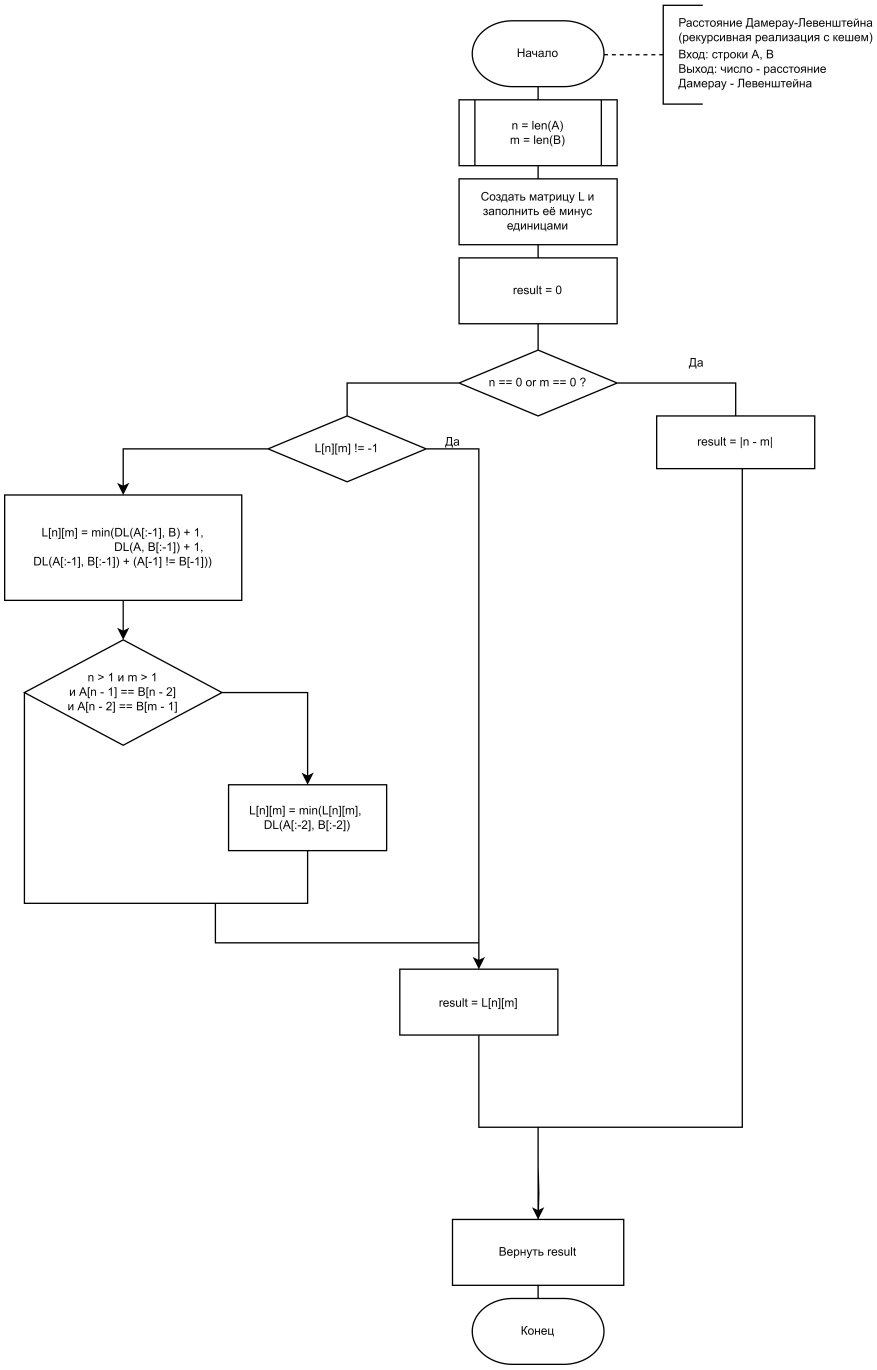

Рисунок 2.4 – Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна с использованием кеша в виде матрицы

Вывод

Перечислены требования к вводу и программе, а также были построены схемы требуемых алгоритмов.

3 Технологическая часть

В данном разделе будут приведены требования к ПО, средства реализации и листинги кода.

3.1 Требования к ПО

Программа принимает две строки с учетом регистров. В качестве результата возвращается число, равное редакторскому расстоянию. Необходимо реализовать возможность сравнения по времени и по памяти.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран ЯП Python.

Данный язык имеет все небходимые инструменты для решения поставленной задачи.

3.3 Сведения о модулях программы

Программа состоит из трех модулей:

1) main.py — точка входа;

2) algorythms.py — хранит реализацию алгоритмов;

3) compare.py — хранит реализацию системы замера времени.

3.4 Листинг кода

В листингах 3.1, 3.2, 3.3, 3.4 приведены реализаций алгоритмов нахождения расстояния Левенштейна и Дамерау–Левенштейна.

```
def levenshteinDistance(A, B, output = True):
  n = len(A)
  m = len(B)
  L = createMatrix(n + 1, m + 1)

  for i in range(1, n + 1):
    for j in range(1, m + 1):
      if A[i - 1] == B[j - 1]:
        L[i][j] = L[i - 1][j - 1]
      else:
        L[i][j] = min(L[i - 1][j],
                L[i][j - 1],
                L[i - 1][j - 1]
                ) + 1

  if (output):
    printMatrix(L, A, B)

  return L[n][m]

```

Листинг 3.1 – Функция нахождения расстояния Левенштейна нерекурсивным методом

```
def damerauLevenshteinDistance(A, B, output = True):
  n = len(A)
  m = len(B)
  L = createMatrix(n + 1, m + 1)

  for i in range(1, n + 1):
    for j in range(1, m + 1):
      if A[i - 1] == B[j - 1]:
        L[i][j] = L[i - 1][j - 1]
      else:
        L[i][j] = min(
                L[i - 1][j],
                L[i][j - 1],
                L[i - 1][j - 1]
                ) + 1

      if (i > 1 and j > 1 and A[i - 1] == B[j - 2] and A[i - 2] == B[j -
    1]):
        L[i][j] = min(L[i][j], L[i - 2][j - 2] + 1)

  if (output):
    printMatrix(L, A, B)

```

```
  return L[n][m]
```

Листинг 3.2 – Функция нахождения расстояния Дамерау–Левенштейна нерекурсивным методом

```
def damerauLevenshteinDistanceRecursive(A, B, output = True):
  n = len(A)
  m = len(B)

  if ((n == 0) or (m == 0)):
    return abs(n - m)

  D = min(
      damerauLevenshteinDistanceRecursive(A[:-1], B) + 1,
      damerauLevenshteinDistanceRecursive(A, B[:-1]) + 1,
      damerauLevenshteinDistanceRecursive(A[:-1], B[:-1]) + (A[-1] != B
    [-1])
      )

  if (n > 1 and m > 1 and A[-1] == B[-2] and A[-2] == B[-1]):
    D = min(D, damerauLevenshteinDistanceRecursive(A[:-2], B[:-2]) + 1)

  return D
```

Листинг 3.3 – Функция нахождения расстояния Дамерау–Левенштейна с использованием рекурсии

```
def recursiveWithCache(A, B, n, m, L):
  if (L[n][m] != -1):
    return L[n][m]

  if n == 0 or m == 0:
    L[n][m] = abs(n - m)
    return L[n][m]

  L[n][m] = min(
        recursiveWithCache(A, B, n - 1, m, L) + 1,
        recursiveWithCache(A, B, n, m - 1, L) + 1,
        recursiveWithCache(A, B, n - 1, m - 1, L) + (A[n - 1] != B[m - 1])
        )
  if (n > 1 and m > 1 and
  A[n - 1] == B[m - 2] and
  A[n - 2] == B[m - 1]):

    L[n][m] = min(L[n][m], recursiveWithCache(A, B, n - 2, m - 2, L) + 1)

  return L[n][m]

```

```
22 def damerauLevenshteinDistanceRecurciveCache(A, B, output = True):
23   n = len(A)
24   m = len(B)
25   L = createMatrix(n + 1, m + 1)
26 
27   for i in range(n + 1):
28     for j in range(m + 1):
29       L[i][j] = -1
30 
31   recursiveWithCache(A, B, n, m, L)
32 
33   if (output):
34     printMatrix(L, A, B)
35 
36   return L[n][m]
```

Листинг 3.4 – Функция нахождения расстояния Дамерау–Левенштейна рекурсивным методом с использованием кеша

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна. Все тесты пройдены.

Таблица 3.1 – Функциональные тесты

	Входные данные		Ожидаемый результат	
№	Строка 1	Строка 2	Левенштейн	Дамерау-Л.
1	"пустая строка"	"пустая строка"	0	0
2	"пустая строка"	x	1	1
3	bmstu	"пустая строка"	5	5
4	"пустая строка"	iu7	3	3
5	lll	lll	0	0
6	something	sommer	6	6
7	united	unique	3	2
8	summer	service	3	3
9	abcdef	acbdef	2	1

Вывод

Были разработаны и протестированы алгоритмы нахождения расстояния Левенштейна нерекурсивно, нахождения расстояния Дамерау – Левенштейна нерекурсивно, рекурсивно, а также рекурсивно с кешированием.

4 Исследовательская часть

В данном разделе приводятся результаты замеров затрат реализаций алгоритмов по памяти и времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры:

— Операционная система Window 10 Home Single Language;

— Память 8 Гб;

— Процессор 11th Gen Intel(R) Core(TM) i7-1165G7 2.80 ГГц, 4 ядра.

4.2 Время выполнения реализаций алгоритмов

Процессорное время реализаций алгоритмов замерялось при помощи функции process_time() из библиотеки time языка Python. Данная функция возвращает количество секунд, прошедших с начала эпохи, типа float.

Замеры времени для каждой длины слов проводились 5000 раз. В качестве результата взято среднее время работы алгоритма на данной длине слова. При каждом запуске алгоритма, на вход подавались случайно сгенерированные строки, днины строк (Length).

Результаты замеров приведены на рисунке 4.1 (в микросекундах).

```
Замер времени для алгоритмов:

Length | Levenshtein | Lev Domerau | Recursive | Cache
     0 |    0.003125 |    0.000000 |  0.003125 | 0.000000
     1 |    0.003125 |    0.000000 |  0.000000 | 0.003125
     2 |    0.000000 |    0.003125 |  0.003125 | 0.003125
     3 |    0.003125 |    0.006250 |  0.021875 | 0.006250
     4 |    0.006250 |    0.009375 |  0.109375 | 0.012500
     5 |    0.009375 |    0.012500 |  0.565625 | 0.018750
     6 |    0.015625 |    0.015625 |  3.109375 | 0.028125
     7 |    0.018750 |    0.018750 | 17.643750 | 0.056250
```

Рисунок 4.1 – Результаты замеров времени (в микросекундах)

Сранение время выполнения между алгоритмами Левенштейн и Дамерау-Леиенштейн (нерекурсивно) на рисунке 4.2

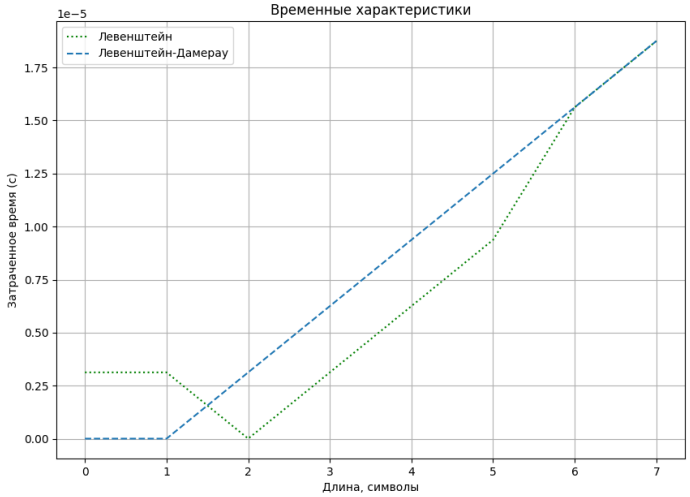

Рисунок 4.2 – Результаты сравнения Левенштейн и Дамерау-Левенштейн (в микросекундах)

Сранение время выполнения между тремя алгоритмами Дамерау-Леиенштейн на рисунке 4.3

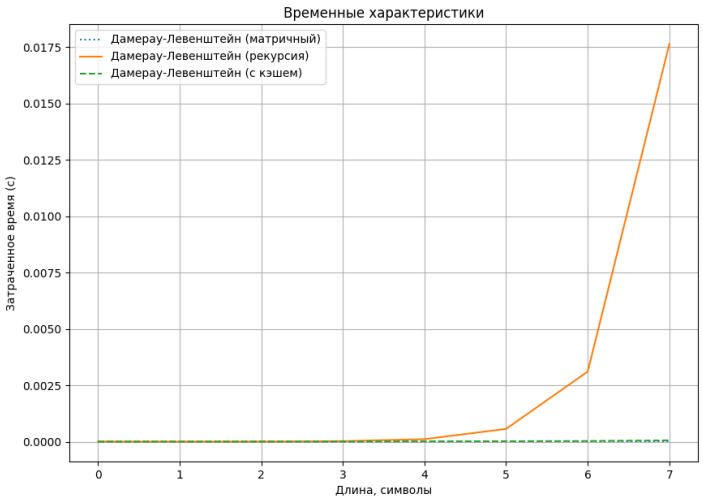

Рисунок 4.3 – Результаты сравнения трех алгоритмов Дамерау-Левенштейн (в микросекундах)

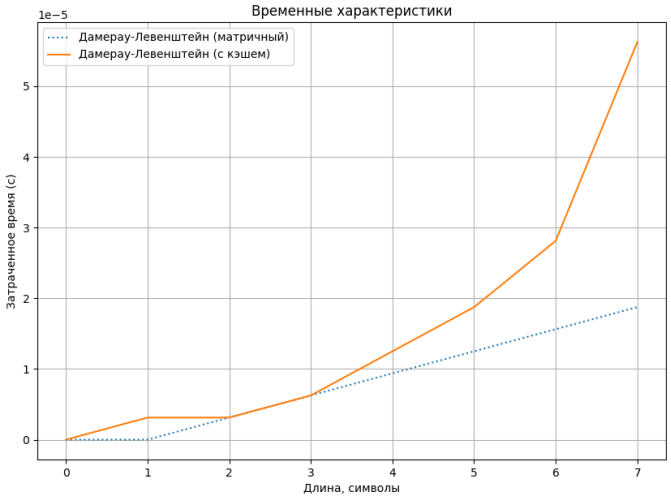

Рисунок 4.4 – Результаты сравнения Дамерау-Левенштейна(матричный) и Дамерау-Левенштейна(с кэшем)

Вывод

В результате замеров можно прийти к выводу, что матричная реализация алгоритмов нахождения расстояний заметно выигрывает по времени при росте строк.

4.3 Затраты памяти выполнения реализаций алгоритмов

Пусть длина строки S1 — n, длина строки S2 — m, тогда затраты памяти на приведенные алгоритмы будут следующими.

Матричный алгоритм поиска расстояния Левенштейна:

— строки S1, S2 — (m + n) · sizeof(char),

— матрица — ((m + 1) · (n + 1)) · sizeof(int),

— длины строк — 2 · sizeof(int),

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк.

Рекурсивный алгоритм поиска расстояния Дамерау – Левенштейна (для каждого вызова):

— строки S1, S2 — (m + n) · sizeof(char),

— длины строк — 2 · sizeof(int),

— размер аргументов функции — 2 · 24,

— размер адреса возврата — 4.

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк. Таким образом, общая затраченная память в рекурсивном алгоритме будет равна $m + n + (2 \cdot 4 + 2 \cdot 24 + 4) \cdot (n + m) = 61 \cdot m + 61 \cdot n$.

Рекурсивный алгоритм поиска расстояния Дамерау – Левенштейна с использованием кеша (для каждого вызова):

— Для всех вызовов память для хранения самой матрицы — ((m + 1) · (n + 1)) · sizeof(int),

— строки S1, S2 — (m + n) · sizeof(char),

— длины строк — 2 · sizeof(int),

— вспомогательные переменные — $1 \cdot \text{sizeof(int)}$,

— размер аргументов функции — $2 \cdot 24$,

— ссылка на матрицу — 8 байт,

— размер адреса возврата — 4.

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк. Таким образом, общая затраченная память в рекурсивном алгоритме будет равна $m + n + 4 \cdot (m + 1) \cdot (n + 1) + (2 \cdot 4 + 4 + 4 + 2 \cdot 24) \cdot (n + m) = 4 \cdot m \cdot n + 69 \cdot m + 69 \cdot n + 4$.

Матричный алгоритм поиска расстояния Дамерау – Левенштейна:

— строки S1, S2 — $(\text{m} + \text{n}) \cdot \text{sizeof(char)}$,

— матрица — $((\text{m} + 1) \cdot (\text{n} + 1)) \cdot \text{sizeof(int)}$,

— длины строк — $2 \cdot \text{sizeof(int)}$,

— вспомогательные переменные — $3 \cdot \text{sizeof(int)}$.

Таким образом, общая затраченная память в рекурсивном алгоритме будет равна $m + n + 4 \cdot (m + 1) \cdot (n + 1) + 6 \cdot 4 = 4 \cdot m \cdot n + 5 \cdot m + 5 \cdot n + 28$.

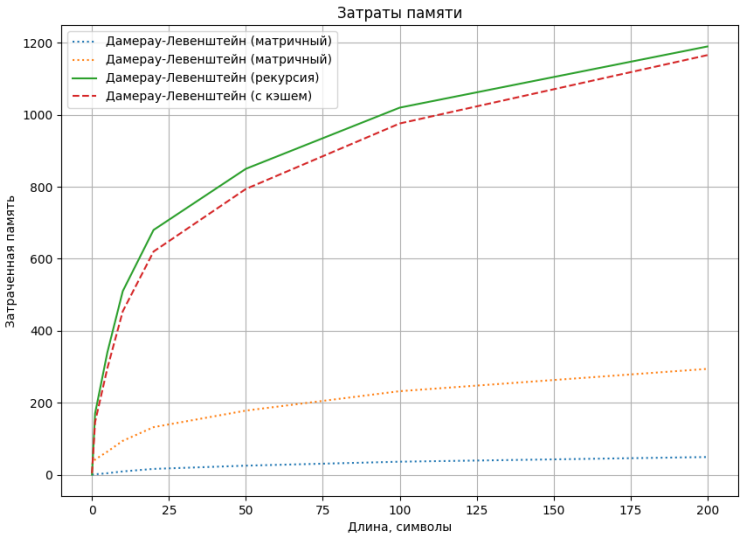

Рисунок 4.5 – Результаты замеров памяти(в байтах)

4.4 Вывод

В результате можно прийти к выводу, что матричная реализация алгоритмов нахождения расстояний заметно выигрывает по памяти при росте строк.

Заключение

В ходе выполнения лабораторной работы были решены все задачи:

— описаны алгоритмы нахождения расстояний Левенштейна и Дамерау-Левенштейна;

— реализованы алгоритмы поиска расстояния Левенштейна и расстояния Дамерау-Левенштейна без рекурсии;

— реализованы рекурсивные алгоритмы поиска расстояния Дамерау-Левенштейна с и без матрицы-кеша;

— проведен сравнительный анализ линейной и рекурсивной реализаций алгоритмов определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);

— подготовлен отчет о лабораторной работе.

Экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализаций для различных длин строк. Было получено, что рекурсивная реализация алгоритмов без кеширования в 3-4 раза проигрывает по памяти нерекурсивной реализации. Однако ее можно улучшить, добавив кеширование, и получить небольшой (в 1.5 раза) выигрыш по памяти по сравнению с нерекурсивными алгоритмами. Анализ временных затрат показал, что для длинных строк (10 символов и больше) рекурсивная реализация алгоритмов работает в 1.5 раза дольше, чем нерекурсивная.

Список использованных источников

1. Вычисление расстояния Левенштейна [Электронный ресурс]. – Режим доступа: https://foxford.ru/wiki/informatika/vychislenie-rasstoyaniya-levenshteyna. – Дата доступа: 14.09.2023.

2. Расстояние Левенштейна [Электронный ресурс]. – Режим доступа: https://vc.ru/newtechaudit/129654-rasstoyanie-levenshteyna-dlya-poiska-opechatok-v-dannyh-klienta. – Дата доступа: 14.09.2023.

3. time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions - дата доступа: 14.09.2023).