Отчет по лабораторной работе 1
По предмету “Анализ алгоритмов”
По теме “Расстояния Левенштейна и Дамерау-Левенштейна”

Фирсова Дарья ИУ7-56

2018

Введение

В лабораторной работе изучаются расстояние Левенштейна и расстояние Демерау-Левенштейна. Требуется применить метод динамического программирования, изучить работу алгоритма и получить практические навыки реализации алгоритмов. Задачи для лабораторной работы:

1. Изучение алгоритмов Левенштеи˘на и Дамерау-Левенштеи˘на нахождения расстояния между строками;

2. Применение метода динамического программирования для матричнои˘ реализации указанных алгоритмов;

3. Получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричнои˘ версии и одного из алгоритмов в рекурсивнои˘ версии;

4. Сравнительныи˘ анализ линеи˘нои˘ и рекурсивнои˘ реализации˘ выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);

5. Экспериментальное подтверждение различии˘ во временно´и˘ эффективности рекурсивнои˘ и нерекурсивнои˘ реализации˘ 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 выполненнои˘ лабораторнои˘ работе, выполненного как расче¨тно-пояснительная записка к работе.

1 Аналитическая часть

Алгоритмы имеют широкое применение: для исправления ошибок в слове при поисковых запросах, вводах текстов и распознавании текстов и речи, для сравнения белков.

1.1 Описание алгоритмов

Алгоритм находит редакционное расстояние - последовательность действий, для получения одного слова из другого. Пусть S_1 и S_2 — две строки (длиной $M*M$ и $N*N$ соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) $d(S_1, S_2)$ можно подсчитать по следующей рекуррентной формуле: $d(S_1, S_2) = D(M, N)$, где

$$D(i,j) = \begin{cases} 0, & \text{i} = 0, j = 0 \\ i, & \text{j} = 0, i > 0 \\ j, & \text{i} = 0, j > 0 \\ min \begin{cases} D(i, j-1) + 1, & Insert \\ D(i-1, j) + 1, & \text{j} > 0, i > 0; Delete \\ D(i-1, j-1) + m(S_1[i], S_2[j]) & Match\ or\ Replace \end{cases} \end{cases}$$

где $m(a, b)$ равна нулю, если $a = b$ единице в противном случае.
Для алгоритма Дамерау-Левенштейна существует возможность обмена элемента через один по диагонали.

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if} \min(i,j) = 0, \\ \min \begin{cases} d_{a,b}(i-1,j) + 1 \\ d_{a,b}(i,j-1) + 1 \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2) + 1 \end{cases} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\ \min \begin{cases} d_{a,b}(i-1,j) + 1 \\ d_{a,b}(i,j-1) + 1 \\ d_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise,} \end{cases}$$

Операция обмена учитывает специфику применения - ошибка в неверном порядке двух букв встречается чаще всего.

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов

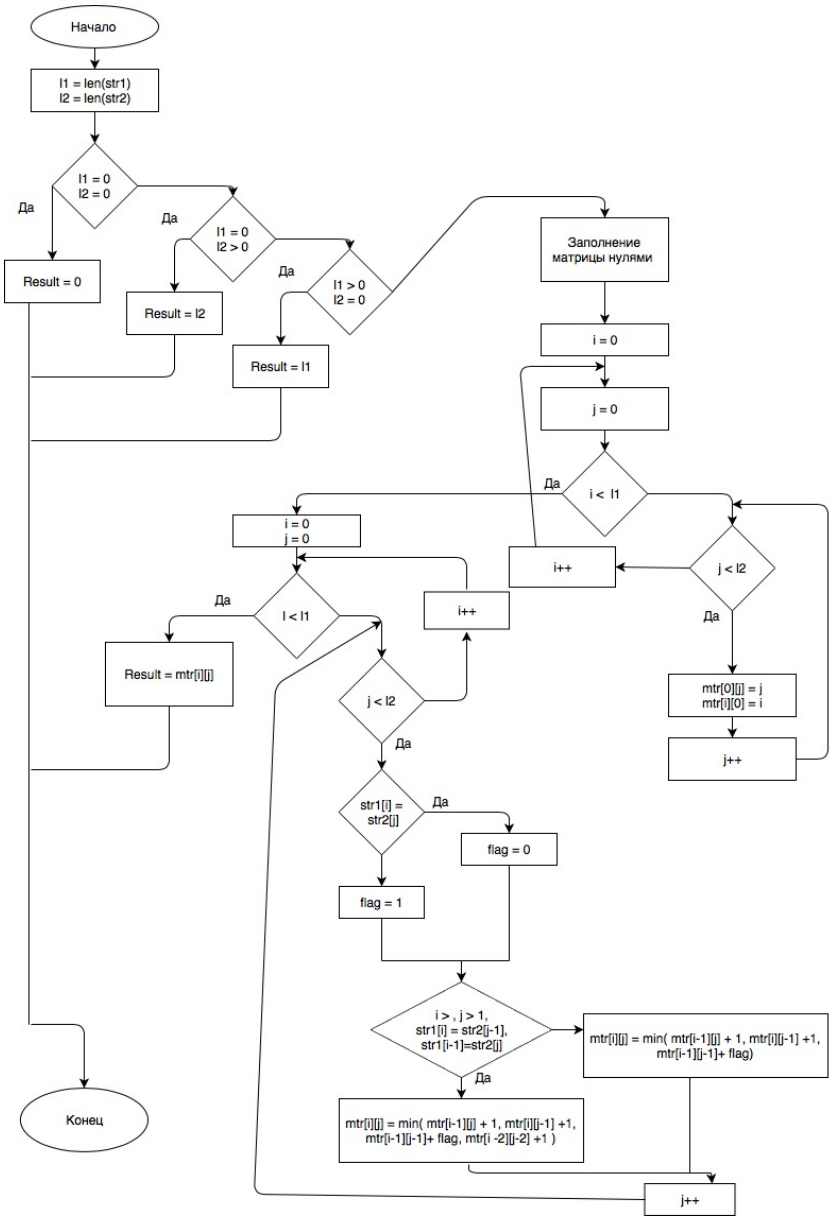


Рис. 1: Схема алгоритма Дамерау-Левенштейна

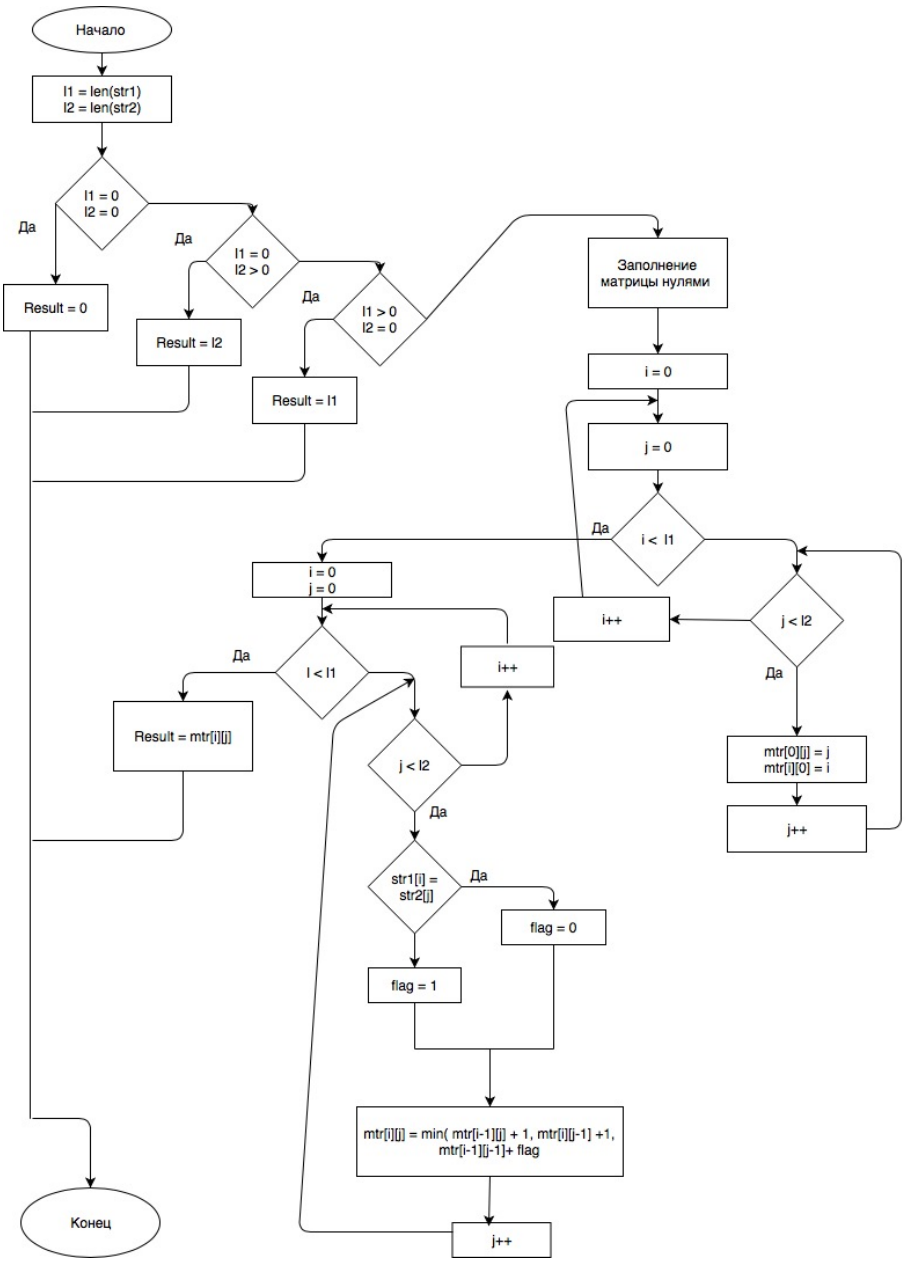


Рис. 2: Схема алгоритма Левенштейна

3 Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

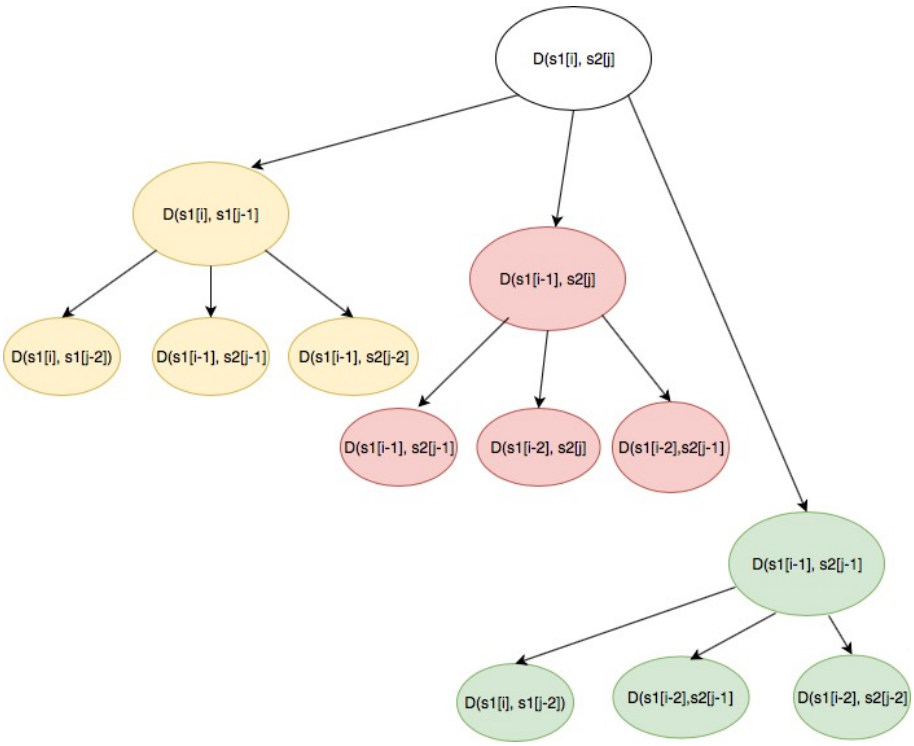


Рис. 3: Вложенность рекурсии

3.1 Требования к программному обеспечению

3.2 Средства реализации

В данной работе использовался язык Python 3.6, в среде Pycharm. Для измерения времени использовался модуль time.

3.3 Листинг кода

```
def levenstein(str1, str2):
    l1, l2 = len(str1), len(str2)

    if (l1 == 0) and (l2 == 0):
        return 0

    if (l1 == 0) and (l2 > 0):
        return l2
```

```
9
10      if (l1 > 0) and (l2 == 0):
11          return l1
12
13      mtr = [[0 for x in range(l2)] for y in range(l1)]
14      print(mtr)
15      for i in range(l1):
16          for j in range(l2):
17              mtr[0][j] = j
18              mtr[i][0] = i
19      print(mtr)
20
21      for i in range(1, l1):
22          for j in range(1, l2):
23              if str1[i] == str2[j]:
24                  flagmatch = 0
25              else:
26                  flagmatch = 1
27              mtr[i][j] = min(mtr[i - 1][j] + 1, mtr[i][j - 1] + 1,
    mtr[i - 1][j - 1] + flagmatch)
28
29      return mtr[i][j]
30
31
32  '''
33      for i in range(l1):
34          for j in range(l2):
35              print(mtr[i][j], end=' ')
36          print()
37  '''
38
39
40  def demerau(str1, str2):
41      l1, l2 = len(str1), len(str2)
42
43      if (l1 == 0) and (l2 == 0):
44          return 0
45
46      if (l1 == 0) and (l2 > 0):
47          return l2
48
49      if (l1 > 0) and (l2 == 0):
50          return l1
51
52      mtr = [[0 for x in range(l2)] for y in range(l1)]
```

```
    for i in range(l1):
        for j in range(l2):
            mtr[0][j] = j
            mtr[i][0] = i

    for i in range(1, l1):
        for j in range(1, l2):
            if str1[i] == str2[j]:
                flagmatch = 0
            else:
                flagmatch = 1

            if (i > 1) and (j > 1) and (str1[i] == str2[j - 1])
    and (str1[i - 1] == str2[j]):
                mtr[i][j] = min(mtr[i - 1][j] + 1, mtr[i][j - 1]
    + 1, mtr[i - 1][j - 1] + flagmatch,
                                mtr[i - 2][j - 2] + 1)
            else:
                mtr[i][j] = min(mtr[i - 1][j] + 1, mtr[i][j - 1]
    + 1, mtr[i - 1][j - 1] + flagmatch)

    return mtr[i][j]

    # mtr[i][j] = min(mtr[i - 1][j] + 1, mtr[i][j - 1] + 1, mtr[i
    - 1][j - 1] + flagmatch, mtr[])


'''
    for i in range(l1):
        for j in range(l2):
            print(mtr[i][j], end=' ')
        print()
'''



def recursion(str1, str2):
    l1, l2 = len(str1), len(str2)
    if l1 == 0 or l2 == 0:
        return max(l1, l2)
    if str1[-1] == str2[-1]:
        flagmatch = 0
    else:
        flagmatch = 1

```

```
94      result = min(
95          [recursion(str1[:-1], str2) + 1, recursion(str1, str2
        [:-1]) + 1, recursion(str1[:-1], str2[:-1]) + flagmatch])
96
97      return result
```

functions.py

4 Экспериментальная часть

В данном разделе будут приведены примеры работы программы для трех разных вариантов: стандартный алгоритм Левенштейна, алгоритм Дамерау-Левенштейна, и рекурсивный алгоритм Левенштейна. Приведет сравнительный анализ двух нерекурсивных алгоритмов при тестах от 100 до 1100 букв в слове. Тесты всех трех алгорит

4.1 Примеры работы

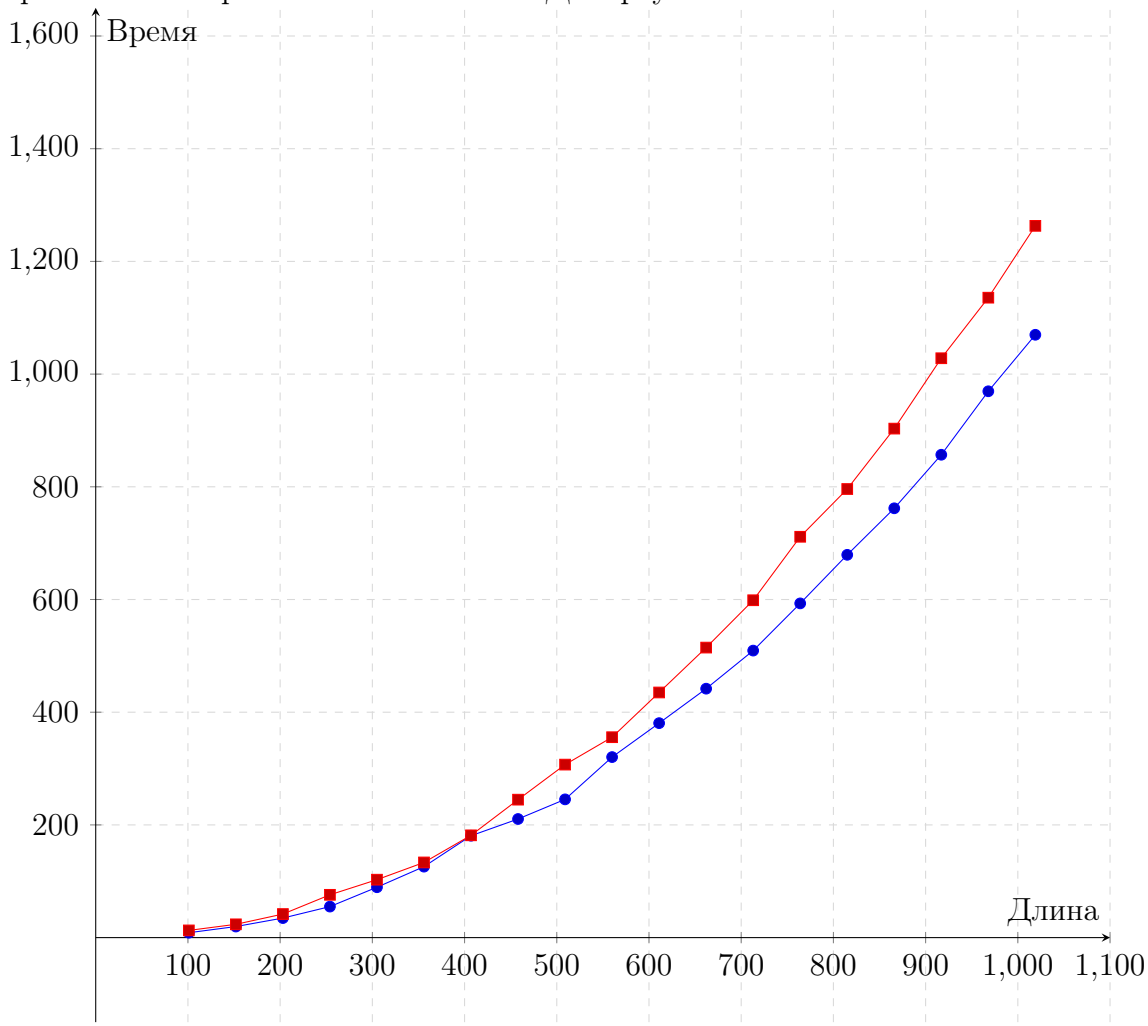
Входные данные	Левенштейн	Дамерау	Рекурсия
aaba, abab	2	2	2
qwerty , wqeryt	4	2	4
polynomial, exponential	6	6	6
tartar, otara	3	3	3

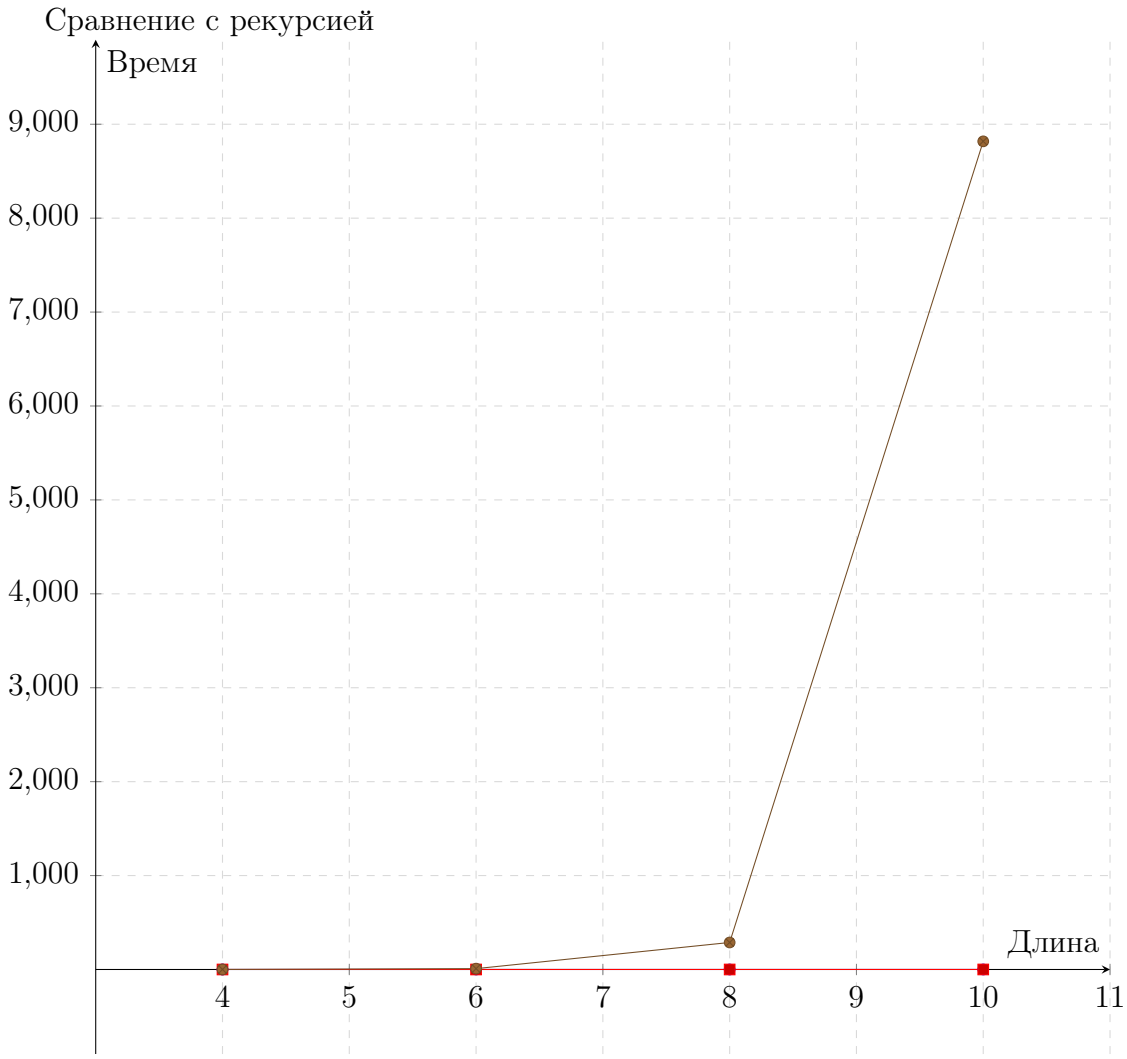
Пример результата работы матричной реализации для тестовых данных polynom, exponent

$$\begin{vmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 2 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 2 & 3 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 2 & 3 & 4 & 5 & 5 & 6 \\ 5 & 4 & 3 & 3 & 4 & 4 & 5 & 6 \\ 6 & 5 & 4 & 4 & 4 & 5 & 5 & 6 \\ 7 & 6 & 5 & 5 & 5 & 4 & 5 & 6 \\ 8 & 7 & 6 & 6 & 6 & 5 & 5 & 6 \end{vmatrix}$$

4.2 Сравнительный анализ

Сравнение алгоритма Левенштейна и Дамерау-Левенштейна





5 Вывод

В ходе лабораторной работы были проанализированы алгоритмы поиска редакционного расстояния. Приведена практическая реализация алгоритмов. Для составления отчета был изучен язык разметки и математических фунцкий Latex.

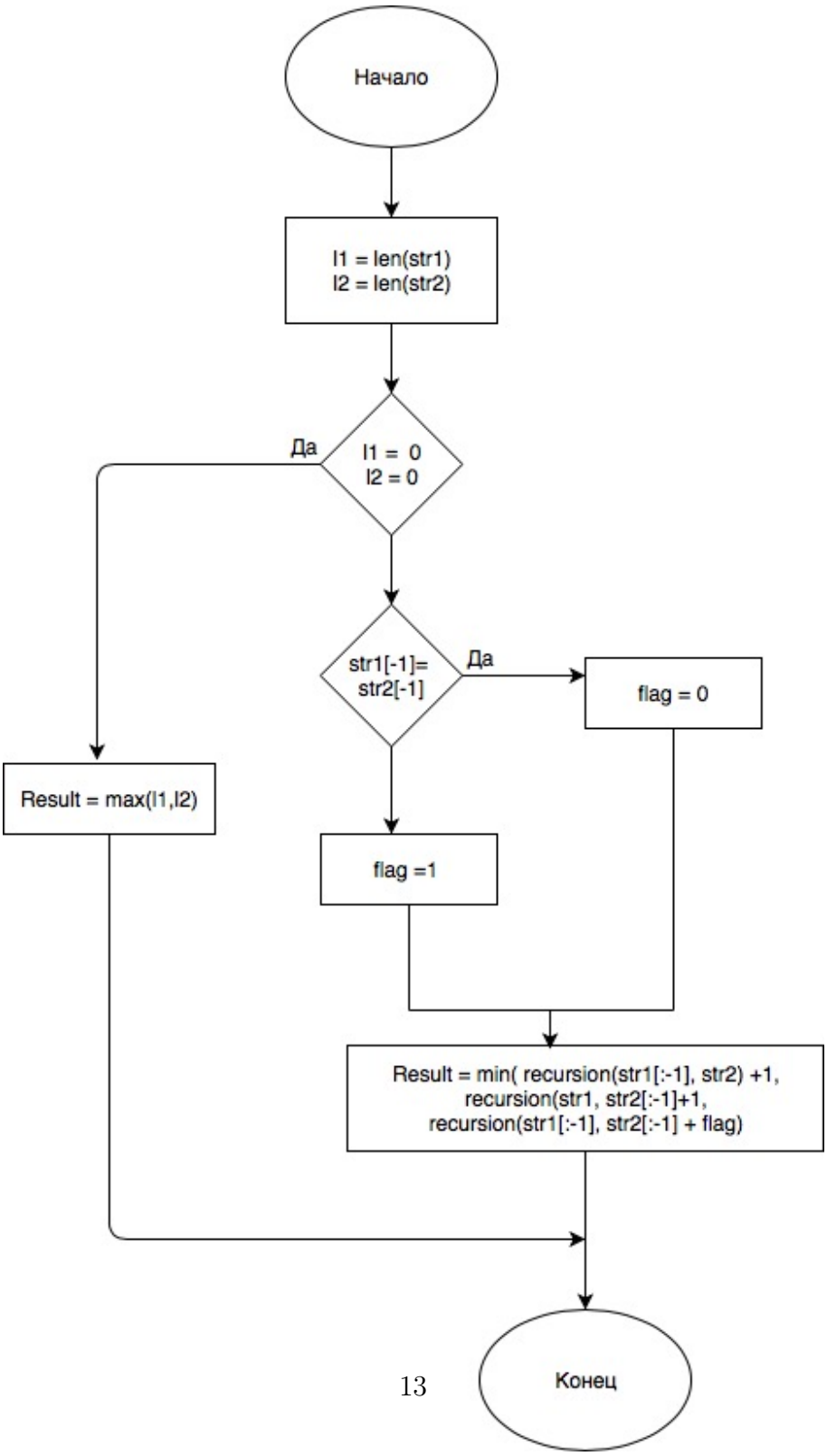


Рис. 4: Схема рекурсивного алгоритма Левенштейна