Государственное образовательное учреждение
высшего профессионального образования
«Московский Государственный Технический Университет
имени Н.Э. Баумана»

Отчет
По лабораторной работе №1
По курсу «Анализ Алгоритмов»
На тему «Расстояние Левенштейна»

Кизилов Дмитрий, ИУ7-54

Москва, 2017

Оглавление

Постановка задачи

Реализовать алгоритм поиска расстояния Левенштейна:

1. Базовый
2. Модифицированный
3. Рекурсивный

Описание алгоритма.

Расстояние Левенштейна (также редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Блок-схемы

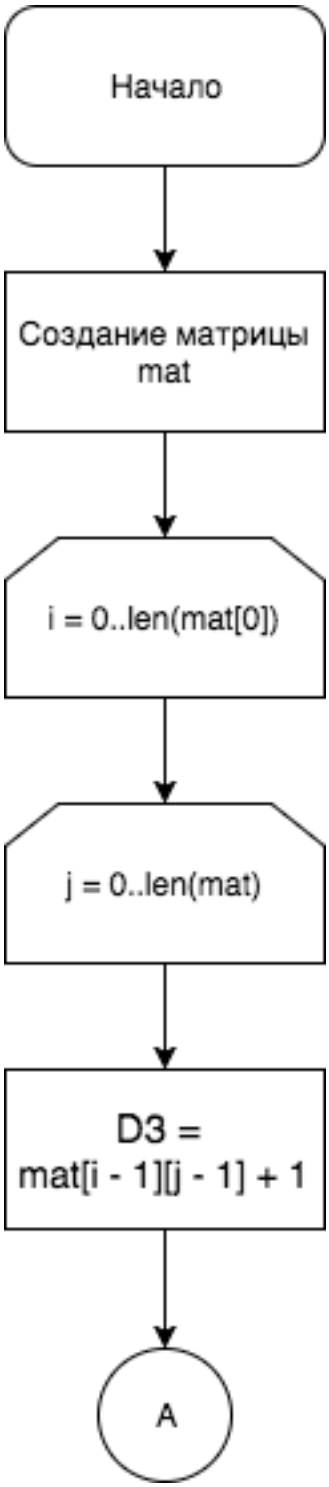

Рис. 1: Блок-схема первой части матричного базового алгоритма

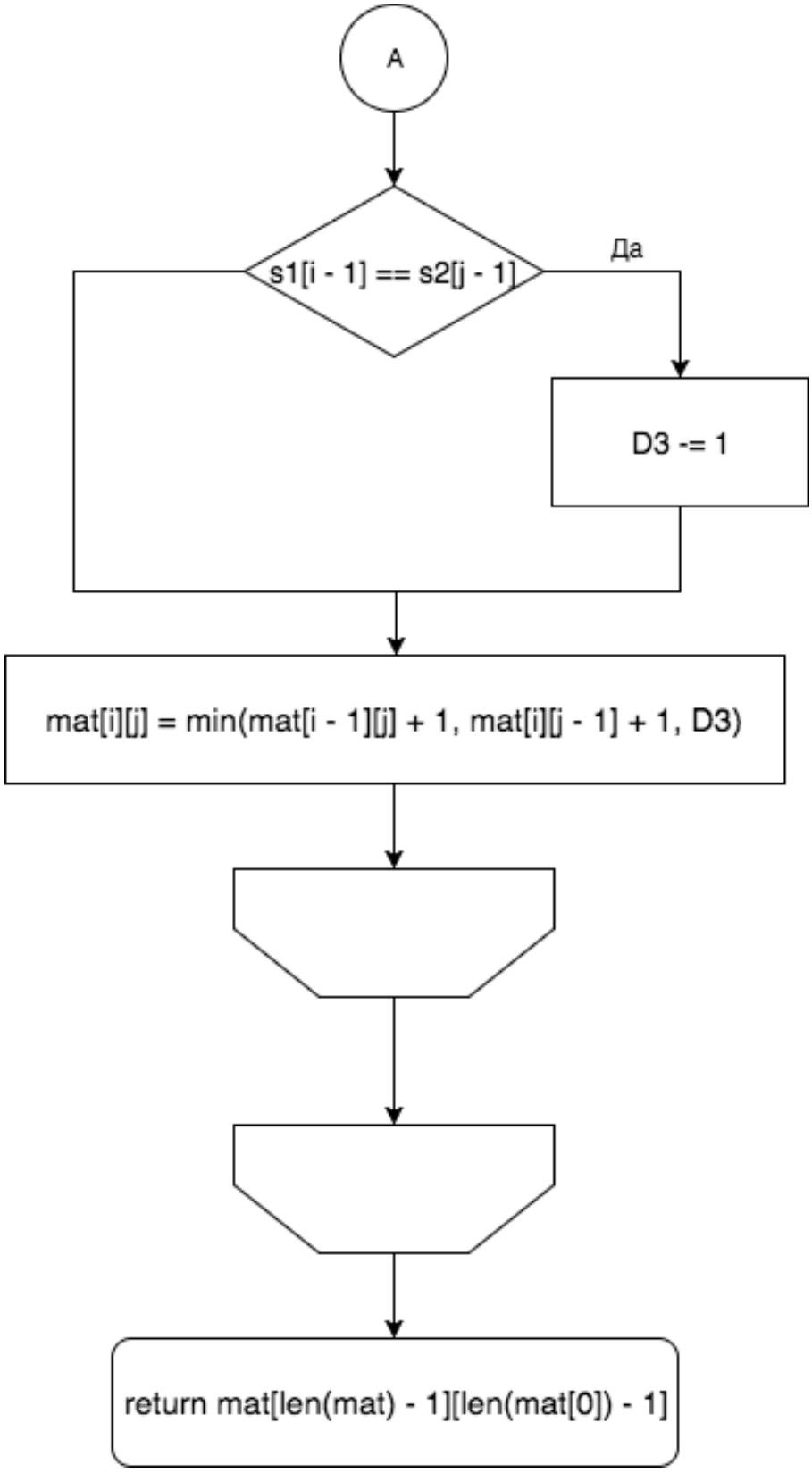

Рис. 2: Блок-схема второй части матричного базового алгоритма

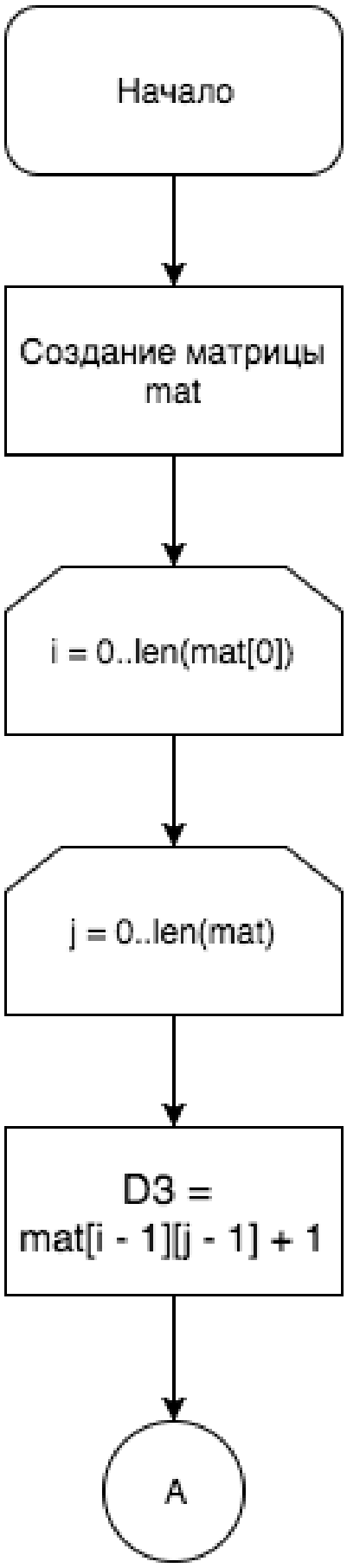

Рис. 3: Блок-схема рекурсивного алгоритма

Листинг

```
1  import timeit
2  import time
3
4
5  # Matrix format printing
6  def printMat(mat):
7      for i in range(0, len(mat)):
8          for x in mat[i]:
9              print("{:^3}".format(x), sep = '', end = '')
10         print()
11
12
13 # Matrix creation size len1xlen2 for Levenstein alg
14 def makeMatForLev(len1, len2):
15     mat = [[]] * len1
16     for i in range(0, len1):
17         mat[i] = [0] * len2
18
19     for i in range(0, len(mat)):
20         mat[i][0] = i
21     for i in range(0, len(mat[0])):
22         mat[0][i] = i
23
24     return mat
25
26
27 # Levenstein alg with matrix, basic
28 def LevBasic(s1, s2):
29     mat = makeMatForLev(len(s1) + 1, len(s2) + 1)
30
31     for j in range(1, len(mat[0])):
32         for i in range(1, len(mat)):
33
34             D3 = mat[i - 1][j - 1] + 1
35             if s1[i - 1] == s2[j - 1]:
36                 D3 -= 1
37             mat[i][j] = min(mat[i - 1][j] + 1, mat[i][j - 1] + 1, D3
                   )
38
39 #     printMat(mat)
40     return mat[len(mat) - 1][len(mat[0]) - 1]
41
42
43 # Levenstein alg with matrix, modifyed
44 def LevMod(s1, s2):
45     mat = makeMatForLev(len(s1) + 1, len(s2) + 1)
46
47     for j in range(1, len(mat[0])):
```

```
48          for i in range(1, len(mat)):
49
50              D3 = mat[i - 1][j - 1] + 1
51              if s1[i - 1] == s2[j - 1]:
52                  D3 -= 1
53
54              if (i > 1) and (j > 1):
55                  D4 = max(mat[i - 1][j] + 1, mat[i][j - 1] + 1, D3)
56                  if (s1[i - 1] == s2[j - 2]) and (s1[i - 2] == s2[j -
                        1]):
57                      D4 = mat[i - 2][j - 2] + 1
58
59                  mat[i][j] = min(mat[i - 1][j] + 1, mat[i][j - 1] +
                        1, D3, D4)
60
61              else:
62                  mat[i][j] = min(mat[i - 1][j] + 1, mat[i][j - 1] +
                        1, D3)
63
64  #     printMat(mat)
65      return mat[len(mat) - 1][len(mat[0]) - 1]
66
67
68  # Levenstein alg with matrix, recursive
69  def LevRecur(s1, s2):
70      if (len(s1) <= 1) and (len(s2) <= 1):
71          if (len(s1) == 0) and (len(s2) == 0):
72              return 0
73          elif len(s1) != len(s2):
74              return 1
75
76          if (s1[0] == s2[0]):
77              return 0
78          else:
79              return 1
80
81      if (len(s1) > 1) and (len(s2) > 1):
82          D3 = LevRecur(s1[:len(s1) - 1], s2[:len(s2) - 1])
83          if s1[len(s1) - 1] != s2[len(s2) - 1]:
84              D3 += 1
85      else:
86          D3 = -1
87
88      if len(s1) > 1:
89          D1 = LevRecur(s1[:len(s1) - 1], s2) + 1
90      else:
91          D1 = -1
92
93      if len(s2) > 1:
94          D2 = LevRecur(s1, s2[:len(s2) - 1]) + 1
```

```
95      else:
96          D2 = -1
97
98      if D1 < 0:
99          D1 = max(D2, D3)
100     if D2 < 0:
101         D2 = max(D1, D3)
102     if D3 < 0:
103         D3 = max(D1, D2)
104
105     return min(D1, D2, D3)
106
107
108 def main():
109     s1 = input('Enter s1: ')
110     s2 = input('Enter s2: ')
111     print()
112
113     for_output = (len(s2) + 1) * 3
114     rep_num = int(50)
115
116     basic = LevBasic(s1, s2)
117     print('\nBasic:', basic)
118     print('Basic time ({:d} reps):'.format(rep_num),
119           timeit.timeit('LevBasic(s1, s2)', setup='s1 = "{:s}"; s2 =
              "{:s}"'.format(s1, s2),
120                         number = rep_num, globals=globals()))
121
122     print('-' * for_output)
123     print()
124
125     modified = LevMod(s1, s2)
126     print('\nModified:', modified)
127     print('Modified time ({:d} reps)'.format(rep_num),
128           timeit.timeit('LevMod(s1, s2)', setup='s1 = "{:s}"; s2 =
              "{:s}"'.format(s1, s2),
129                         number = rep_num, globals=globals()))
130
131     print('-' * for_output)
132     print()
133
134     recur = LevRecur(s1, s2)
135     print('\nRecur:', recur)
136
137     print('Recur time ({:d} reps): {:f}'.format(int(rep_num),
138           timeit.timeit('LevRecur(s1, s2)', setup='s1 = "{:s}"; s2 =
              "{:s}"'.format(s1, s2),
139                         number = rep_num, globals=globals())))
140     print()
141
```

```
142
143  main ()
```

Тесты

Входные данные	Результат	Ожидаемый результат	Тест пройден
s1 = "Привет" s2 = "Приве" // Пропущена одна буква	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Да
s1 = "Привет" s2 = "Приветт" //Добавлена лишняя буква	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Да
s1 = "Привет" s2 = "Превет" //Одна из букв изменена	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Базовый: 1 Модифицированный: 1 Рекурсивный: 1	Да
s1 = "Привет" s2 = "Првиет" //Обмен местами 2 букв	Базовый: 2 Модифицированный: 1 Рекурсивный: 2	Базовый: 2 Модифицированный: 1 Рекурсивный: 2	Да
s1 = "Мобильник" s2 = "Моиблнег" //Комбинированный случай	Базовый: 5 Модифицированный: 4 Рекурсивный: 5	Базовый: 5 Модифицированный: 4 Рекурсивный: 5	Да
s1 = "\0" s2 = "\0" //Пустые строки	Базовый: 0 Модифицированный: 0 Рекурсивный: 0	Базовый: 0 Модифицированный: 0 Рекурсивный: 0	Да

Замеры времени

Входные данные	Базовый	Модифицированный	Рекурсивный
s1 = "Привет" s2 = "Приве" // Пропущена одна буква	0.00561	0.00331	0.18462
s1 = "Привет" s2 = "Приветт" //Добавлена лишняя буква	0.00574	0.00489	0.95220
s1 = "Привет" s2 = "Превет" //Одна из букв изменена	0.00536	0.00368	0.42712
s1 = "Привет" s2 = "Првиет" //Обмен местами 2 букв	0.00389	0.00403	0.43738
s1 = "Мобильник" s2 = "Моиблнег" //Комбинированный случай	0.00682	0.00713	2.93727

Замеры времени в секундах (среднее из 50 замеров)

Графики в данной задаче не дадут отчетливую картину зависимости времени от длинны слов, так как время зависит так же от степени отличия слов, и способа отличия (перестановка букв стоящих рядом, или вставка и удаление буквы).

Выводы

В результате проведенных испытаний алгоритма было установлено, что:

1. Модифицированный алгоритм даст меньший результат, если пара соседних букв переставлены местами, однако мы потеряем во времени из-за большего количества сравнений

2. Рекурсивный алгоритм всегда будет давать такой же результат, как базовый матричный. Однако рекурсивный алгоритм в несколько раз дольше, чем матричный.

Заключение

В ходе лабораторной работы были реализованы 3 алгоритма поиска расстояния Левенштейна: базовый и модифицированный матричные алгоритмы, и рекурсивный. Были получены навыки работы с матрицами и рекурсиями в Python, а так же работе с LaTeX.