Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №1

Отчет на тему:

Расстояние Левенштейна

Щербатюк Д.С.

ИУ7-54

Москва 2017

1 Расстояние Левенштейна

1.1 Постановка задачи

Реализовать алгоритм поиска расстояния Левенштейна, используя три алгоритма:

а) базовый
б) модифицированный
в) базовый через рекурсию

1.2 Алгоритм

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) $d(S_1, S_2)$ можно подсчитать по следующей рекуррентной формуле :

$$D(S_1[1..M], S_2[1..N] = min(D(S_1[1..M-1], S_2[1..N]) + 1^D, \tag{1.1}$$

$$D(S_1[1..M], S_2[1..N-1]) + 1^I, \tag{1.2}$$

$$D(S_1[1..M-1], S_2[1..N-1]) + \begin{cases} 0^M \ if \ S_1[M] = S_2[N], \\ 1^R \ else \end{cases}), \tag{1.3}$$

Где — разрешенные операции:
1. Замена символа $(R, replace)$ Штраф 1.
2. Вставка символа $(I, insert)$ Штраф 1.
3. Удаление символа $(D, delete)$ Штраф 1.
4. Совпадение символа $(M, match)$ Штраф 0.

В модифицированном алгоритме добавлена еще одна операция: 5. Перестановка символа $(X, exchange)$ Штраф 1.
В рекуррентную формулу добавляется еще один член минимума:

$$D(S_1[1..M-1], S_2[1..N-1]) + \begin{cases} 1^X \ if \ S_1[M-1] = S_2[N], \\ 0 \ else \end{cases} \tag{1.4}$$

Алгоритм можно реализовать с помощью матрицы, двигаясь построчно или по столбцам, рассматривая «квадрат» значений:

	Пустая строка	M	A
Пустая строка	0	1	2
M	1	0 +1	1 +1
E	2	1 +1	1 = min(2,2,1)

1.3 Листинг кода

base.py (файл с базовыи алгоритмом)

```python
def distance(s1, s2):
    l1 = len(s1)
    l2 = len(s2)

    row1 = [x for x in range(l2 + 1)]  # we need only two rows
    row2 = [1]  # the first row and column will be like [0, 1, ... n] and intersect at 0


    for i in range(1, l1 + 1):  # loop through rows
        for j in range(1, len(row1)):  # loop through column
            if s1[i - 1] != s2[j - 1]:  # if symbols doesn't match
                row2.append(min(row1[j] + 1,  # there are three variants
                                row2[j - 1] + 1,
                                row1[j - 1] + 1))
            else:
                row2.append(row1[j - 1])  # if match
            row1 = row2  # change rows
            row2 = [i + 1]

    return row1[-1]  # return the lower right value matrix


if __name__ == "__main__":
    print(distance("ма", "ам"))
```

base_with_rec.py (файл с базовым алгоритмом через рекурсию)

```python
def distance(s1, s2):
    l1 = len(s1)
    l2 = len(s2)
    if l1 == 1 and l2 == 1:  # if s1 and s2 is symbols
        if s1 == s2:  # and they match
            return 0
        else:
            return 1
    else:
            if (l1 > l2 == 1) or (l2 > l1 == 1):  # but if one of str is not a symbols
                return abs(l1 - l2) + 1  # return distance for N inserts + penalty

    t = 0
    if s1[-1] != s2[-1]:  # if the last symbols of strings aren't match
        t = 1
```

```
16
17      return min(distance(s1[:l1 - 1], s2) + 1,
18                 distance(s1, s2[:l2 - 1]) + 1,
19                 distance(s1[:l1 - 1], s2[:l2 - 1]) + t)
20
21  if __name__ == "__main__":
22      print(distance("метра", "матрица"))
```

modified.py (файл с модифицированным алгоритмом)

```
1   def distance(s1, s2):
2       d = None
3       l1 = len(s1)
4       l2 = len(s2)
5
6       row1 = [x for x in range(l2 + 1)]   # we need only two rows
7       row2 = [1] # the first row and column will be like [0, 1, ... n] and
            intersect at 0
8
9
10      for i in range(1, l1 + 1):  # loop through rows
11          for j in range(1, len(row1)):  # loop through column
12              if s1[i - 1] != s2[j - 1]:  # if symbols doesn't match
13                  if j > 1 and s2[j - 2] == s1[i - 1]:  # and we can change
                        current symbol and previous symbol in s2
14                      row2.append(min(row1[j] + 1,  # there are four variants
15                                      row2[j - 1] + 1,
16                                      row1[j - 1] + 1,
17                                      row1[j - 2] + 1))
18                  else:  # there are three variants
19                      row2.append(min(row1[j] + 1,
20                                      row2[j - 1] + 1,
21                                      row1[j - 1] + 1))
22              else:
23                  row2.append(row1[j - 1])   # if match
24          row1 = row2  # change rows
25          row2 = [i + 1]
26
27      return row1[-1]  # return the lower right value matrix
28
29
30  if __name__ == "__main__":
31      print(distance("метра", "матрица"))
```

release.py (файл с пользовательским вводом)

```
1   import base as bs
2   import modified as md
3   import base_with_rec as rec
```

```
4
5  if __name__ == "__main__":
6      print("Введите два слова через пробел:")
7      s = input()
8      s1, s2 = s.split()
9      print("Расстояние Левенштейна между двумя введенными словами: ",
                          bs.distance(s1, s2),
10             md.distance(s1, s2),
11             rec.distance(s1, s2))
```

tests.py (файл тестов)

```
1  import base as bs
2  import modified as md
3  import base_with_rec as rec
4  from itertools import combinations
5  import time
6
7  def test():
8      l = ["Январь",
9          "Февраль",
10         "Март",
11         "Апрель",
12         "Май",
13         "Июнь",
14         "Июль",
15         "Август",
16         "Сентябрь",
17         "Октябрь",
18         "Ноябрь",
19         "Декабрь"]
20     t1, t2, t3 = 0, 0, 0
21     times = 0
22     print("Расстояние Левенштейна между строками:\n")
23     for i in combinations(l, 2):
24         times += 1
25
26         start = time.time()
27         a = bs.distance(i[0], i[1])
28         stop = time.time()
29
30         t1 += (stop - start)
31
32         start = time.time()
33         b = md.distance(i[0], i[1])
34         stop = time.time()
35
36         t2 += (stop - start)
```

```
37
38          start = time.time()
39          c = rec.distance(i[0], i[1])
40          stop = time.time()
41
42          t3 += (stop - start)
43
44          print(i[0], "   ", i[1], ":  ", a, " ", b, " ", c, " ")
45
46      print("Среднее время:\n")
47      print("Базовый: {0:.10f}".format(t1 / times))
48      print("Модифицированный:  {0:.10f}".format(t2 / times))
49      print("Базовый с рекурсией:  {0:.10f}".format(t3 / times))
50
51  if __name__ == "__main__":
52      test()
```

Таблица 1.1 — Тестовые данные.

Ввод		Вывод	Ожидаемое	Результат
S_1	S_2			
Январь	Февраль	4 4 4	4 4 4	V
Январь	Март	4 4 4	4 4 4	V
Январь	Апрель	5 5 5	5 5 5	V
Январь	Май	5 5 5	5 5 5	V
Январь	Июнь	5 5 5	5 5 5	V
Январь	Июль	5 5 5	5 5 5	V
Январь	Август	6 6 6	6 6 6	V
Январь	Сентябрь	5 5 5	5 5 5	V
Январь	Октябрь	5 5 5	5 5 5	V
Январь	Ноябрь	4 4 4	4 4 4	V
Январь	Декабрь	4 4 4	4 4 4	V
Февраль	Март	6 6 6	6 6 6	V
Февраль	Апрель	4 4 4	4 4 4	V
Февраль	Май	6 6 6	6 6 6	V
Февраль	Июнь	6 6 6	6 6 6	V
Февраль	Июль	5 5 5	5 5 5	V
Февраль	Август	6 6 6	6 6 6	V
Февраль	Сентябрь	6 6 6	6 6 6	V
Февраль	Октябрь	6 6 6	6 6 6	V
Февраль	Ноябрь	6 6 6	6 6 6	V
Февраль	Декабрь	5 5 5	5 5 5	V
Март	Апрель	5 5 5	5 5 5	V
Март	Май	2 2 2	2 2 2	V
Март	Июнь	4 4 4	4 4 4	V
Март	Июль	4 4 4	4 4 4	V
Март	Август	5 5 5	5 5 5	V
Март	Сентябрь	7 7 7	7 7 7	V
Март	Октябрь	6 6 6	6 6 6	V
Март	Ноябрь	5 5 5	5 5 5	V
Март	Декабрь	5 5 5	5 5 5	V
Апрель	Май	6 6 6	6 6 6	V
Апрель	Июнь	5 5 5	5 5 5	V
Апрель	Июль	4 4 4	4 4 4	V
Апрель	Август	5 5 5	5 5 5	V
Апрель	Сентябрь	7 7 7	7 7 7	V
Апрель	Октябрь	6 6 6	6 6 6	V
Апрель	Ноябрь	5 5 5	5 5 5	V

Заключение

Реализован алгоритм Левенштейна, позволяющий решать множество прикладных задач: автоматического исправления ошибок в слове, сравнения файлов, а в биоинформатике генов и хромосом. Проведено сравнение 3-х реализаций алгоритмов, выявлены их слабые места. Алгоритм с рекурсией является самым медленным, его стоит заменить базовым или модифицированным. Базовый и модифицированный сильно по скорости в данной реализации не различаются.