

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №1

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

Расстояние Левенштейна

Работу выполнила: Подвасецкий Дмитрий,
ИУ7-54Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение	2
1 Аналитическая часть	4
2 Конструкторская часть	6
3 Технологическая часть	10
3.1 Выбор ЯП	10
3.2 Замеры времени	10
3.3 Сведения о модулях программы	10
4 Исследовательская часть	15
Заключение	17

Введение

Расстояние Левенштейна - минимальное количество операций вставки, удаления, замены, необходимых для превращения одной строки в другую.

Расстояние Левенштейна в основном применяется для:

1. для нахождения объектов или записей в поисковых системах;
2. в базах данных, при поиске с неполно-заданным или неточно заданным именем;
3. для исправления ошибок при вводе текста;
4. для исправления ошибок в результате автоматического распознавания отсканированного текста или речи;

Целью данной лабораторной работы является изучение алгоритмов Левенштейна и Дamerau-Левенштейна.

Задачами данной лабораторной являются:

1. изучение алгоритмов Левенштейна и Дamerau-Левенштейна нахождения расстояния между строками;
2. получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
3. экспериментальное подтверждение различий во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма;

4. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 | Аналитическая часть

Задача по нахождению расстояния Левенштейна заключается в поиске минимального количества операций вставки, удаления, замены, необходимых для превращения одной строки в другую.

Действия обозначаются так:

1. D — удаление,
2. I — вставка,
3. R — замена,
4. M — совпадение.

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

$$D(i, j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ \min(& j > 0, i > 0 \\ D(i, j - 1) + 1, \\ D(i - 1, j) + 1, \\ D(i - 1, j - 1) + m(S_1[i], S_2[j]) \\), \end{cases}$$

где $m(a, b)$ равна нулю, если $a = b$ и единице в противном случае; $\min\{a, b, c\}$ возвращает наименьший из аргументов.

При вычислении расстояния Дамерау-Левенштейна добавляется еще одна операция - транспозиция, т.е. перестановка двух соседних элементов.

Расстояние Дамерау-Левенштейна вычисляется по следующей рекуррентной формуле:

$$D(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0 \\ \min(\\ D(i, j - 1) + 1, \\ D(i - 1, j) + 1, \\ D(i - 1, j - 1) + m(S_1[i], S_2[j]) \\ D(i - 2, j - 2) + 1, & \text{if } i, j > 1 \text{ and } a_i = b_{j-1}, a_{i-1} = b_j \\ \min(\\ D(i, j - 1) + 1, \\ D(i - 1, j) + 1, \\ D(i - 1, j - 1) + m(S_1[i], S_2[j]) \\) & \text{otherwise} \end{cases}$$

2 | Конструкторская часть

Требования к вводу:

1. Запрос на работу в тестовом режиме
2. Две строки, строчные и заглавные буквы считаются различными

Требования к программе:

1. Корректный ввод, корректный вывод, программа не должна аварийно завершаться

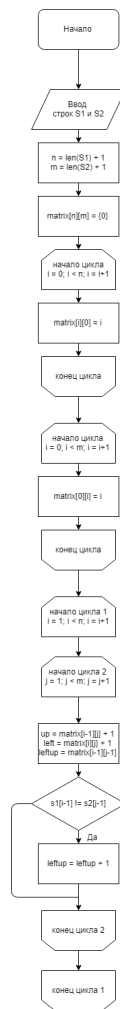


Схема 1. Матричный алгоритм Левенштейна

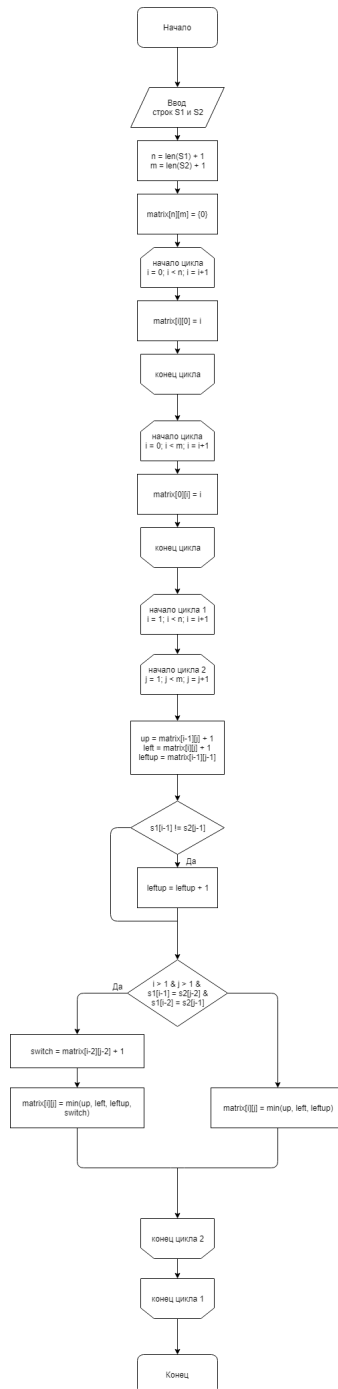


Схема 2. Матричный алгоритм Дамерау-Левенштейна

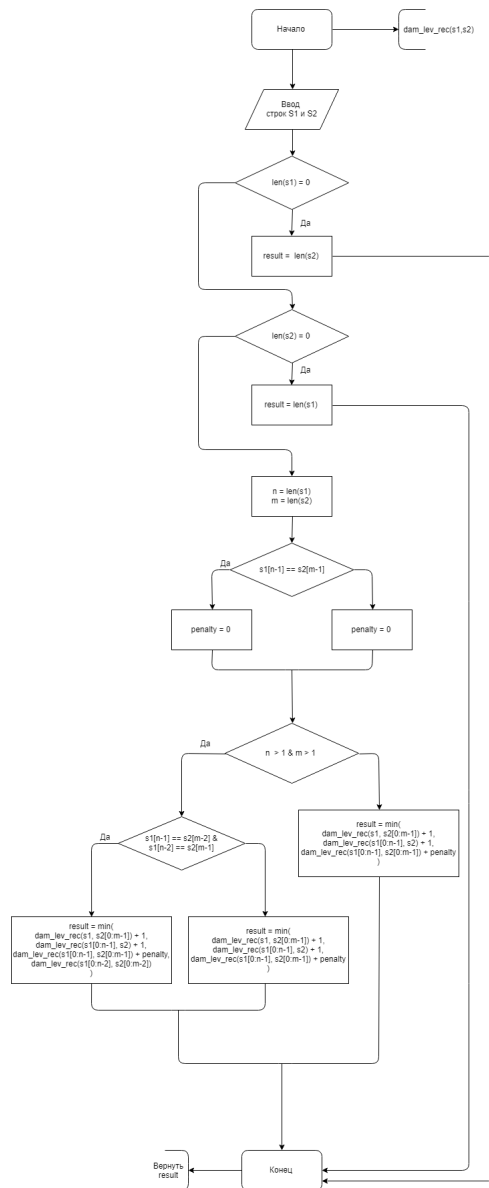


Схема 3. Рекурсивный алгоритм Дамарау-Левенштейна

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран Haskell, для ознакомления с ним.

3.2 Замеры времени

Замер времени работы алгоритмов производился при помощи функций `getCurrentTime`, `diffUTCTime` из библиотеки `Data.Time.Clock`.

Также производится усреднение времени работы алгоритмов. Для этого время считается для 12 вызовов, и после делится на 12.

3.3 Сведения о модулях программы

Программа состоит из:

- `main.hs` - главный файл программы
- `dam lev matrix.hs` - файл с функцией вычисления расстояния Дамерау-Левенштейна матрично (Листинг 3.3)
- `lev matrix.hs` - файл с функцией вычисления расстояния Левенштейна матрично (Листинг 3.1)
- `dam lev rec.hs` - файл с функцией вычисления расстояния Дамерау-Левенштейна рекурсивно (Листинг 3.2)
- `tests.hs` - файл с тестами

Листинг 3.1: Функция нахождения расстояния Левенштейна матрично

```

1 lev_mtr_abs :: String -> String -> Matrix Int
2 lev_mtr_abs s1 s2 = do
3     let a = length s2
4     let b = length s1
5     let mtr = fromLists [[j | j <- [i..a+i]] | i <- [0..b]]
6
7     lev_mtr s2 s1 2 2 mtr
8
9 cmp :: Char -> Char -> Int
10 cmp a b = if a == b
11             then 0
12             else 1
13
14 calc_min :: String -> String -> Int -> Int -> Matrix Int ->
15           Int
16 calc_min s1 s2 i j mtr = minimum [(getElem (i-1) j mtr) +
17                                     1,
18                                     (getElem i (j-1) mtr) +
19                                     1,
20                                     (getElem (i-1) (j-1) mtr)
21                                     + (cmp (s1 !! (j-2))
22                                     (s2 !! (i-2)))]
23
24 lev_mtr :: String -> String -> Int -> Int -> Matrix Int ->
25         Matrix Int
26 lev_mtr "" s2 i j mtr = mtr
27 lev_mtr s1 "" i j mtr = mtr
28 lev_mtr s1 s2 i j mtr =
29     if i >= length s2 + 2
30     then lev_mtr s1 s2 2 (j+1) mtr
31     else if j >= length s1 + 2
32     then mtr
33     else lev_mtr s1 s2 (i+1) j (setElem (calc_min
34                                     s1 s2 i j mtr) (i,j) mtr)

```

Листинг 3.2: Функция нахождения расстояния Дамерау-Левенштейна рекурсивно

```

1 cmp :: Char -> Char -> Int
2 cmp a b = if a == b
3           then 0
4           else 1
5
6 dam_lev_rec :: String -> String -> Int
7 dam_lev_rec "" s2 = length s2
8 dam_lev_rec s1 "" = length s1
9 dam_lev_rec s1 s2 = do
10     let l1 = length s1
11     let l2 = length s2
12     if (l1 > 2) && (l2 > 2)
13     then if ((last s1) == (s2 !! (l2-2)) && (s1 !! (l1
14         -2)) == (last s2))
15         then minimum [(dam_lev_rec (init s1)) s2 +
16             1,
17             (dam_lev_rec s1 (init s2)) +
18             1,
19             (dam_lev_rec (init s1) (init
20                 s2)) + (cmp (last s1) (
21                     last s2)),
22             (dam_lev_rec (init (init s1))
23                 (init (init s2))) + 1]
24     else minimum [(dam_lev_rec (init s1) s2) +
25         1,
26         (dam_lev_rec s1 (init s2)) +
27         1,
28         (dam_lev_rec (init s1) (init
29             s2)) + (cmp (last s1) (
30                 last s2))]
31     else minimum [(dam_lev_rec "" s2) + 1,
32         (dam_lev_rec s1 "") + 1,
33         (dam_lev_rec (init s1) (init s2)) + (
34             cmp (last s1) (last s2))]

```

Листинг 3.3: Функция нахождения расстояния Дameraу-Левенштейна матрично

```

1 dam_lev_mtr_abs :: String -> String -> Matrix Int
2 dam_lev_mtr_abs s1 s2 = do

```

```

3      let a = length s2
4      let b = length s1
5      let mtr = fromLists [[j | j <- [i..a+i]] | i <- [0..b]]
6
7      dam_lev_mtr s2 s1 2 2 mtr
8
9  cmp :: Char -> Char -> Int
10 cmp a b = if a == b
11           then 0
12           else 1
13
14 calc_min :: String -> String -> Int -> Int -> Matrix Int ->
15           Int
16 calc_min s1 s2 i j mtr =
17     if (i > 2) && (j > 2)
18     then if ((s1 !! (j-2)) == (s2 !! (i-3))) && ((s1 !!
19              (j-3)) == (s2 !! (i-2)))
20     then minimum [(getElem (i-1) j mtr) + 1,
21                  (getElem i (j-1) mtr) + 1,
22                  (getElem (i-1) (j-1) mtr) + (cmp
23                    (s1 !! (j-2)) (s2 !! (i-2))),
24                  (getElem (i-2) (j-2) mtr) + 1]
25     else minimum [(getElem (i-1) j mtr) + 1,
26                  (getElem i (j-1) mtr) + 1,
27                  (getElem (i-1) (j-1) mtr) + (cmp
28                    (s1 !! (j-2)) (s2 !! (i-2)))]
29     else minimum [(getElem (i-1) j mtr) + 1,
30                  (getElem i (j-1) mtr) + 1,
31                  (getElem (i-1) (j-1) mtr) + (cmp (s1
32                    !! (j-2)) (s2 !! (i-2)))]
33
34 dam_lev_mtr :: String -> String -> Int -> Int -> Matrix Int
35             -> Matrix Int
36 dam_lev_mtr "" s2 i j mtr = mtr
37 dam_lev_mtr s1 "" i j mtr = mtr
38 dam_lev_mtr s1 s2 i j mtr =
39     if i >= length s2 + 2
40     then dam_lev_mtr s1 s2 2 (j+1) mtr

```

```
37     else if j >= length s1 + 2
38         then mtr
39         else dam_lev_mtr s1 s2 (i+1) j (setElem (
            calc_min s1 s2 i j mtr) (i,j) mtr)
```

4 | Исследовательская часть

Был проведен замер времени работы каждого из алгоритмов.

Таблица. 1. Сравнение времени работы.

Длина (симв.)	dam lev rec (с)	lev mtr (с)	dam lev mtr (с)
3	0.00003	0.00003	0.00003
4	0.0001	0.00004	0.00004
5	0.0008	0.00005	0.00006
6	0.04	0.00012	0.00008
7	0.02083	0.00012	0.00012
8	0.10250	0.00016	0.00020
9	0.57036	0.00020	0.00025

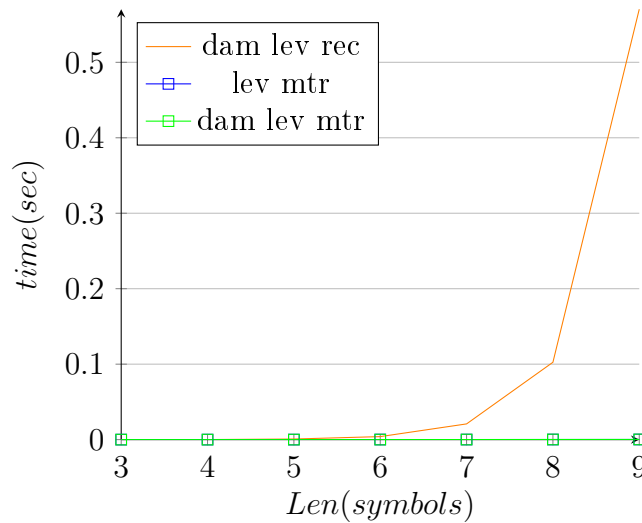


Рис. 1. График времени работы всех трех алгоритмов.

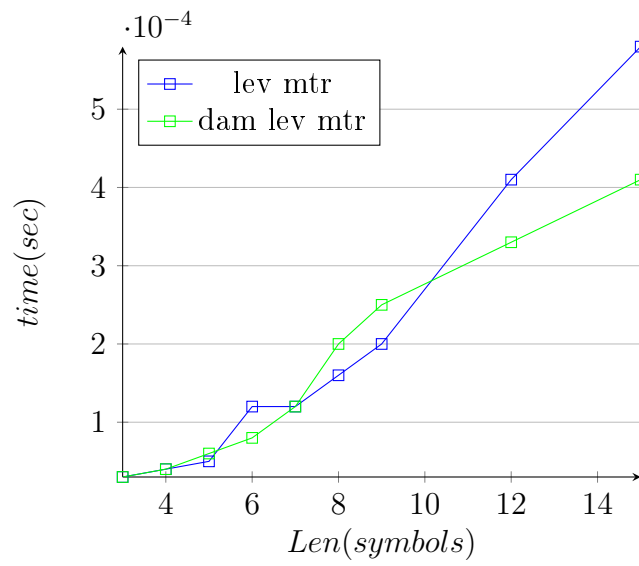


Рис. 2. График времени работы матричных алгоритмов.

Матричные реализации практически не отличаются друг от друга по времени. В то время как рекурсивная уже при длине строк больше 5 работает медленнее в десятки раз.

Заключение

Мною были изучены алгоритмы Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками.

Также мною получены практические навыки реализации указанных алгоритмов в матричной и рекурсивных версиях.

Экспериментально было проверено различие во временной эффективности рекурсивной и нерекурсивной реализаций алгоритма Дамерау-Левенштейна.

Было произведено описание и обоснование полученных результатов.

В результате можно сделать вывод, что практически всегда предпочтительнее использовать матричный вариант.