МГТУ им. Баумана

Лабораторная работа №1

По курсу: "Анализ алгоритмов"

Расстояние Левенштейна

Работу выполнила: Подвашецкий Дмитрий,
ИУ7-54Б

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

Москва, 2019

Оглавление

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 Дамерау-Левенштейна.

Задачами данной лабораторной являются:

1. изучение алгоритмов Левенштейна и Дамерау-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 временно́й 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:

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

где $m(a,b)$ равна нулю, если $a = b$ и единице в противном случае; $min\{\, a, b, c\}$ возвращает наименьший из аргументов. [1]

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

Расстояние Дамерау-Левенштейна вычисляется по следующей рекуррентной формуле:

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

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

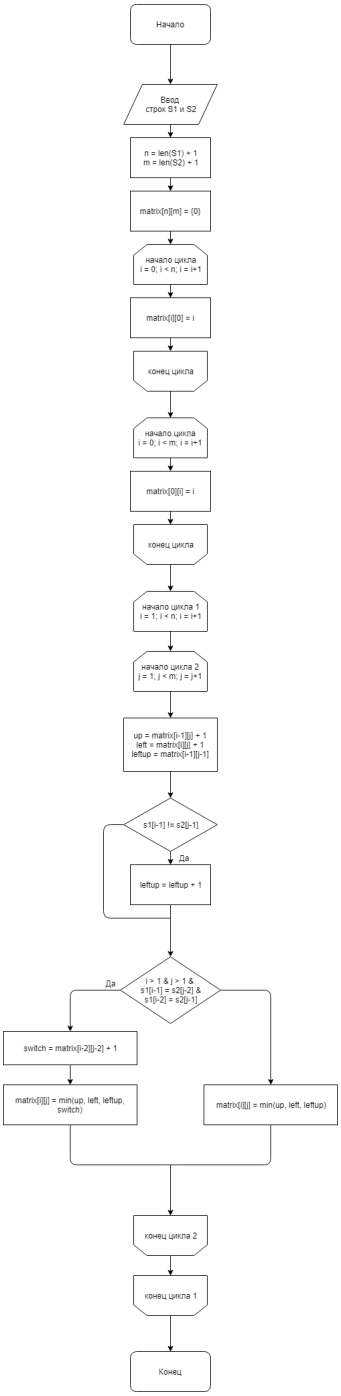

Схема 2. Матричный алгоритм Дамерау-Левенштейна

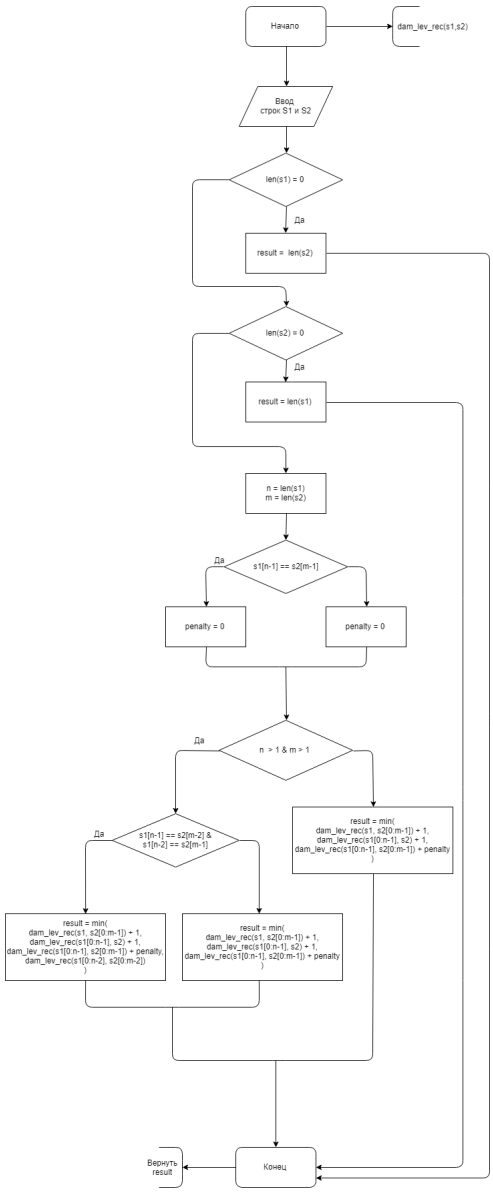

Схема 3. Рекурсивный алгоритм Дамерау-Левенштейна

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. [2]

Также производится усреднение времени работы улгоритмов. 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

```
lev_mtr_abs :: String -> String -> Matrix Int
lev_mtr_abs s1 s2 = do
    let a = length s2
    let b = length s1
    let mtr = fromLists [[j | j <- [i..a+i]] | i <- [0..b]]

    lev_mtr s2 s1 2 2 mtr

cmp :: Char -> Char -> Int
cmp a b = if a == b
            then 0
            else 1

calc_min :: String -> String -> Int -> Int -> Matrix Int ->
    Int
calc_min s1 s2 i j mtr = minimum [(getElem (i-1) j mtr) +
   1,
                                  (getElem i (j-1) mtr) +
                                     1,
                                  (getElem (i-1) (j-1) mtr)
                                      + (cmp (s1 !! (j-2))
                                     (s2 !! (i-2)))]


lev_mtr :: String -> String -> Int -> Int -> Matrix Int ->
   Matrix Int
lev_mtr "" s2 i j mtr = mtr
lev_mtr s1 "" i j mtr = mtr
lev_mtr s1 s2 i j mtr =
    if i >= length s2 + 2
        then lev_mtr s1 s2 2 (j+1) mtr
        else if j >= length s1 + 2
            then mtr
            else lev_mtr s1 s2 (i+1) j (setElem (calc_min
                s1 s2 i j mtr) (i,j) mtr)
```

Листинг 3.2: Функция нахождения расстояния Дамерау-Левенштейна рекурсивно

```
cmp :: Char -> Char -> Int
cmp a b = if a == b
             then 0
             else 1

dam_lev_rec :: String -> String -> Int
dam_lev_rec "" s2 = length s2
dam_lev_rec s1 "" = length s1
dam_lev_rec s1 s2 = do
    let l1 = length s1
    let l2 = length s2
    if (l1 > 2) && (l2 > 2)
        then if ((last s1) == (s2 !! (l2-2)) && (s1 !! (l1
           -2)) == (last s2))
                then minimum [(dam_lev_rec (init s1)) s2 +
                   1,
                              (dam_lev_rec s1 (init s2)) +
                                 1,
                              (dam_lev_rec (init s1) (init
                                 s2)) + (cmp (last s1) (
                                 last s2)),
                              (dam_lev_rec (init (init s1))
                                  (init (init s2))) + 1]

                else minimum [(dam_lev_rec (init s1) s2) +
                   1,
                              (dam_lev_rec s1 (init s2)) +
                                 1,
                              (dam_lev_rec (init s1) (init
                                 s2)) + (cmp (last s1) (
                                 last s2))]
        else minimum [(dam_lev_rec "" s2) + 1,
                      (dam_lev_rec s1 "") + 1,
                      (dam_lev_rec (init s1) (init s2)) + (
                         cmp (last s1) (last s2))]
```

Листинг 3.3: Функция нахождения расстояния Дамерау-Левенштейна матрично

```
dam_lev_mtr_abs :: String -> String -> Matrix Int
dam_lev_mtr_abs s1 s2 = do
```

```
    let a = length s2
    let b = length s1
    let mtr = fromLists [[j | j <- [i..a+i]] | i <- [0..b]]

    dam_lev_mtr s2 s1 2 2 mtr

cmp :: Char -> Char -> Int
cmp a b = if a == b
            then 0
            else 1

calc_min :: String -> String -> Int -> Int -> Matrix Int ->
    Int
calc_min s1 s2 i j mtr =
    if (i > 2) && (j > 2)
        then if ((s1 !! (j-2)) == (s2 !! (i-3))) && ((s1 !!
            (j-3)) == (s2 !! (i-2)))
            then minimum [(getElem (i-1) j mtr) + 1,
                          (getElem i (j-1) mtr) + 1,
                          (getElem (i-1) (j-1) mtr) + (cmp
                             (s1 !! (j-2)) (s2 !! (i-2))),
                          (getElem (i-2) (j-2) mtr) + 1]

            else minimum [(getElem (i-1) j mtr) + 1,
                          (getElem i (j-1) mtr) + 1,
                          (getElem (i-1) (j-1) mtr) + (cmp
                             (s1 !! (j-2)) (s2 !! (i-2)))]

        else minimum [(getElem (i-1) j mtr) + 1,
                      (getElem i (j-1) mtr) + 1,
                      (getElem (i-1) (j-1) mtr) + (cmp (s1
                         !! (j-2)) (s2 !! (i-2)))]

dam_lev_mtr :: String -> String -> Int -> Int -> Matrix Int
    -> Matrix Int
dam_lev_mtr "" s2 i j mtr = mtr
dam_lev_mtr s1 "" i j mtr = mtr
dam_lev_mtr s1 s2 i j mtr =
    if i >= length s2 + 2
        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 (c)	lev mtr (c)	dam lev mtr (c)
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

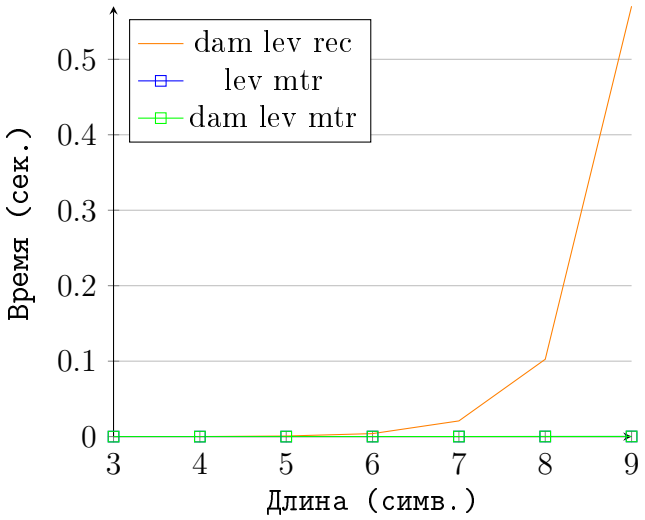

Рис. 1. График вермени работы всех трех алгоритмов.

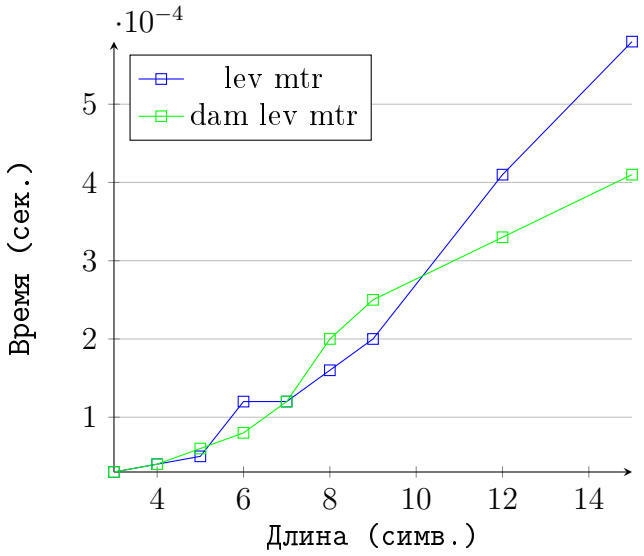

Рис. 2. График вермени работы матричных алгоритмов.

Матричные реализации отличаются друг от друга менее чем на один процент. В то время как рекурсивная уже при длине строк больше 5 работает медленее в десятки раз.

Для длин строк в 6 символов, рекурскивный алгоритм работает медленее в 33 раза, в 7 символов - 173 раза, в 8 символов - 640 раз. (Таблица 1.)

Заключение

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

Также мною получены практические навыки раелизации указанных алгоритмовв матричной и рекурсивных версиях.

При сравнении матричных версий алгоритмов Дамерау-Левенштейна и Левенштейна был сделан вывод, что данные два алгоритма отличаются менее чем на один процент.

Экспериментально было проверено различие во временной эффективности рекурсивной и нерекурсивной реализаций алгоритма Дамерау-Левенштейна. Рекурсивная версия данного алгоритма при длине строк более 5 работает в десятки раз медленнее. При длине строк в 6 символов - медленнее в 33 раз, в 7 - в 173 раза, в 8 - 640 раз.

Было произведено описание и обоснование полученных результатов.

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

Список литературы

1. Нечёткий поиск в тексте и словаре. 2011г. [Электронный ресурс] Режим доступа: https://habr.com/ru/post/114997/ Последнняя дата обращения: 01.10.2019

2. Документация Haskell по модулю Data.Time.Clock. [Электронный ресурс] Режим доступа: http://hackage.haskell.org/package/time-1.9.3/docs/Data-Time-Clock.html Последнняя дата обращения: 01.10.2019

3. Документация Haskell по модулю Data.Matrix. [Электронный ресурс] Режим доступа: https://hackage.haskell.org/package/matrix-0.2.2/docs/Data-Matrix.html Последнняя дата обращения: 01.10.2019

4. Р. В. Душкин. Справочник по языку Haskell. [Текст]/ Р. В. Душкин - 2008. - 542 с.