*Государственное образовательное учреждение высшего профессионального образования*

*«Московский Государственный Технический Университет имени Н. Э. Баумана»*

**ОТЧЕТ**

По лабораторной работе №1

По курсу «Анализ алгоритмов»

Тема: «Расстояние Левенштейна»

Студент: Лумбунов Д.В.

Группа: ИУ7-54

Преподаватель:

Москва, 2019

**Постановка задачи**

**Расстояние Левенштейна** (*редакционное расстояние*, *дистанция редактирования*) — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

**Расстояние Дамерау-Левенштейна** — это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

Рассмотреть и изучить понятия расстояния Левенштейна и расстояния Дамерау-Левенштейна. Реализовать два варианта алгоритма нахождения расстояния Левенштейна (рекурсивного и нерекурсивного вида). Сравнить их временные характеристики экспериментально. Реализовать алгоритм нахождения расстояния Дамерау-Левенштейна. На основании проделанной работы сделать выводы.

**Описание алгоритмов**

**Рекурсивный алгоритм (Левенштейн)**

Здесь и далее считается, что элементы строк нумеруются с первого, как принято в математике, а не с нулевого, как принято во многих языках программирования.

Пусть S1{\displaystyle S\_{1}} и S2 {\displaystyle S\_{2}}SSЫ— две строки (длиной M и N MЬ соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) d(S1{\displaystyle S\_{1}}, S2) можно подсчитать по следующей рекуррентной формуле

d(S1{\displaystyle S\_{1}}, S2) = D(M,N), где

(1)

Рекурсивный алгоритм напрямую реализует эту формулу.

Функция 1 составлена из следующих соображений:

1. Для перевода из пустой строки в пустую требуется ноль операций.
2. Для перевода из пустой строки в строку 𝑎 требуется |𝑎| операций. Аналогично, для перевода из строки 𝑎 в пустую требуется |𝑎| операций.
3. Для перевода из строки 𝑎 в строку 𝑏 требуется выполнить последовательно некоторое кол-во операций (удаление, вставка, замена) в некоторой последовательности. Как можно показать сравнением, последовательность проведения любых двух операций можно поменять, и, как следствие, порядок проведения операций не имеет никакого значения. Тогда цена преобразования из строки 𝑎 в строку 𝑏 может быть выражена как (полагая, что 𝑎′, 𝑏′ — строки 𝑎 и 𝑏 без последнего символа соответственно):
   1. Сумма цены преобразования строки 𝑎′ в 𝑏 и цены проведения операции удаления, которая необходима для преобразования 𝑎 в 𝑎′;
   2. Сумма цены преобразования строки 𝑎 в 𝑏′ и цены проведения операции вставки, которая необходима для преобразования 𝑏′ в 𝑏;
   3. Сумма цены преобразования из 𝑎′ в 𝑏′ и операции замены, предполагая, что 𝑎 и 𝑏 оканчиваются разные символы;
   4. Цена преобразования из 𝑎′ в 𝑏′, предполагая, что 𝑎 и 𝑏 оканчиваются на один и тот же символ.

Очевидно, что минимальной ценой преобразования будет минимальное

значение этих вариантов.

**Алгоритм Вагнера-Фишера (построчный)**

Прямая реализация формулы 1 может быть малоэффективна при больших 𝑖, 𝑗, т. к. множество промежуточных значений 𝐷(𝑖, 𝑗), 1 ≤ 𝑖 ≤ |𝑎|, 1 ≤ 𝑗 ≤ |𝑏| вычисляются заново множество раз подряд. Для оптимизации нахождения расстояния Левенштейна предложено использовать матрицу (двумерный массив) в целях хранения соответствующих промежуточных значений. В таком случае алгоритм представляет собой построчное заполнение матрицы 𝐴|𝑎|,|𝑏| значениями 𝐷(𝑖, 𝑗) по формуле 1.

Можно заметить, что при каждом заполнении новой строки значения предыдущей становятся ненужными. Поэтому можно провести оптимизацию по памяти и использовать дополнительно только одномерный массив размером min(|𝑎|, |𝑏|). Такой вариант алгоритма называется построчным и именно он реализован в данной работе в качестве нерекурсивного.

**Расстояние Дамерау-Левенштейна.**

Между двумя строками a, b определяется функцией da,b(|a|,|b|) как:

Где – индикаторная функция, равная нулю при и 1 в противном случае.

Формула выводится по тем же соображениям, что и формула 1.

Т. к. прямое применение этой формулы неэффективно, то аналогично действиям из предыдущего пункта производится добавление матрицы для хранения промежуточных значений рекурсивной формулы и оптимизация по памяти. В таком случае необходимо хранить одномерный массив длиной 3 min(|𝑎|, |𝑏|).

**Листинги.**

Расстояние Левенштейна:

def Levenshtein(str1, str2):

n = len(str1)

m = len(str2)

if not min(n,m):

return max(n,m)

return min(Levenshtein(str1, str2[:-1]) + 1,

Levenshtein(str1[:-1], str2) + 1,

Levenshtein(str1[:-1], str2[:-1]) + (str1[-1] != str2[-1]))

Алгоритм Вагнера-Фишера:

def Wagner\_Fischer(a, b):

n = len(a)

m = len(b)

if n > m:

a, b = b, a

n, m = m, n

current\_row = range(n + 1)

for i in range(1, m + 1):

previous\_row, current\_row = current\_row, [i] + [0] \* n

for j in range(1, n + 1):

add, delete, change = previous\_row[j] + 1, current\_row[j - 1] + 1, previous\_row[j - 1]

if a[j - 1] != b[i - 1]:

change += 1

current\_row[j] = min(add, delete, change)

return current\_row[n]

Расстояние Дамерау-Левенштейна:

def Damerau\_Levenshtein(s1, s2):

d = {}

n = len(s1)

m = len(s2)

for i in range(-1,n+1):

d[(i,-1)] = i+1

for j in range(-1,m+1):

d[(-1,j)] = j+1

for i in range(n):

for j in range(m):

cost = s1[i] != s2[j]

d[(i,j)] = min(

d[(i-1,j)] + 1, # delete

d[(i,j-1)] + 1, # insert

d[(i-1,j-1)] + cost, # substitute

)

if i and j and s1[i]==s2[j-1] and s1[i-1] == s2[j]:

d[(i,j)] = min (d[(i,j)], d[i-2,j-2] + cost) # transposition

return d[n-1,m-1]

**Тесты + сравнения**

Enter a: aa

Enter b: aaaa

Levenshtein: 2

Time in ticks: 0.0

Wagner\_Fischer: 2

Time in ticks (average by 1000): 0.0007279993057250976

Damerau\_Levenshtein: 2

Time in ticks (average by 1000): 0.0005459951400756837

…

Enter a: abcdaabb

Enter b: acbdaaba

Levenshtein: 3

Time in ticks: 6.370006942749023

Wagner\_Fischer: 3

Time in ticks (average by 1000): 0.0009100034713745116

Damerau\_Levenshtein: 2

Time in ticks (average by 1000): 0.0012739987850189209

…

Enter a: samestr

Enter b: samestr

Levenshtein: 0

Time in ticks: 2.1840022563934327

Wagner\_Fischer: 0

Time in ticks (average by 1000): 0.0005460038185119629

Damerau\_Levenshtein: 0

Time in ticks (average by 1000): 0.0010919989585876464

…

# Сравнение временных хар-к:

Enter a: ti

Enter b: ny

Levenshtein: 2

Time in ticks: 0.0

Wagner\_Fischer: 2

Time in ticks (average by 1000): 0.0003640083312988281

Damerau\_Levenshtein: 2

Time in ticks (average by 1000): 0.0003639996528625488

…

Levenshtein: 4

Time in ticks: 0.0

Wagner\_Fischer: 4

Time in ticks (average by 1000): 0.0008736104488372802

Damerau\_Levenshtein: 4

Time in ticks (average by 1000): 0.0004550234317779541

…

Enter a: usual

Enter b: string

Levenshtein: 6

Time in ticks: 0.3639996528625488

Wagner\_Fischer: 6

Time in ticks (average by 1000): 0.0007279993057250976

Damerau\_Levenshtein: 6

Time in ticks (average by 1000): 0.0007279993057250976

…

Enter a: significant

Enter b: stringtotry

Levenshtein: 10

Time in ticks: 891.1309880256653

Wagner\_Fischer: 10

Time in ticks (average by 1000): 0.001456002950668335

Damerau\_Levenshtein: 9

Time in ticks (average by 1000): 0.002366002082824707

…

**Таблицы измерений времени**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Levenshtein** | | | | |
| **Strlen2\strlen1** | **8** | **6** | **4** | **2** |
| **2** | **~0** | **~0** | **~0** | **~0** |
| **4** | **~0.0111** | **~0.0074** | **~0.0057** |  |
| **6** | **~0.1448** | **~0.0334** |  |  |
| **8** | **~0.9746** |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Wagner\_Fischer** | | | | |
| **Strlen2\strlen1** | **8** | **6** | **4** | **2** |
| **2** | **~0** | **~0** | **~0** | **~0** |
| **4** | **~0.0017** | **~0.0008** | **~0.0007** |  |
| **6** | **~0.0334** | **~0.1448** |  |  |
| **8** | **~0.001** |  |  |  |

**Выводы**

При использовании рекурсивного алгоритма потребление времени при увеличении размера данных растет экспоненциально. При длине строк strlen1=strlen2=8 уже требуется примерно 1 секунда для вычисления результата и это приблизительно в 1000 раз больше, чем требуется при применении алгоритма Вагнера-Фишера.

Применение рекурсивного алгоритма крайне неэффективно по времени, рекомендуется использовать алгоритм Вагнера-Фишера.

**Заключение**

В результате выполнения данной работы рассмотрены и изучены понятия расстояния Левенштейна и расстояния Дамерау-Левенштейна. Реализованы два варианта алгоритма нахождения расстояния Левенштейна (рекурсивного и нерекурсивного вида). Сравнены их временные характеристики как следствие проведённых экспериментов. Реализован алгоритм нахождения расстояния Дамерау-Левенштейна. Были сделаны выводы о временной эффективности рекурсивного и нерекурсивного алгоритмов.