*s*

*Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования*

***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский институт)»***



***(МГТУ им. Н.Э. Баумана)***



ФАКУЛЬТЕТ Информатика и системы управления КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №1**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Расстояние Левенштейна**

Студент гр. ИУ7-56Б

Преподаватель

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **Хеламбаге Г.С.**

(Подпись, дата) (И.О. Фамилия)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** **Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

|  |  |  |  |
| --- | --- | --- | --- |
| [**Введение**](#page2) | | | **3** |
| **1.** | [**Аналитическая часть**](#page4) | | **4** |
|  | 1.1. | [Расстояние Левенштейна](#page4) | 4 |
|  | 1.2. | [Расстояние Дамерау-Левенштейна](#page6) | 6 |
|  | 1.3. | [Практическое применение](#page7) | 7 |
| **2.** | [**Конструкторская часть**](#page7) | | **7** |
|  | 2.1. | [Схемы алгоритмов](#page7) | 7 |
|  | 2.2. | Сравнительный анализ рекурсивной и нерекурсивной реализаций13 | |
| **3.** | **Технологическая часть** | | **14** |
|  | 3.1. | Требования к ПО | 14 |
|  | 3.2. | Средства реализации | 14 |
|  | 3.3. | Листинги кода | 14 |
| **4.** | **Экспериментальная часть** | | **17** |
|  | 4.1. | Примеры работы программы | 17 |
|  | 4.2. | Результаты функционального тестирования | 18 |
|  | 4.3. | Постановка эксперимента по по замеру времени | 18 |
|  | 4.4. | Сравнительный анализ на материале экспериментальных данных 19 | |
| **Заключение** | | | **21** |

Введение

Цель данной лабораторной работы: ознакомиться с понятиями “расстояние Левенштейна” и “расстояние Дамерау-Левенштейна”, изучить матричные и рекурсивные варианты алгоритмов, позволяющих вычислить эти расстояния, а также получить практические навыки в реализации предложенных алгоритмов и их анализе.

Требуется решить следующие задачи лабораторной работы.

1. Реализовать матричный вариант алгоритма нахождения расстояния Левенштейна.
2. Реализовать матричный вариант алгоритма нахождения расстояния Дамерау-Левенштейна.
3. Реализовать рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна.
4. Сравнить реализованные алгоритмы по памяти и по времени.
5. Описать проделанную работу и обосновать получившиеся результаты.

1. Аналитическая часть

1.1. Расстояние Левенштейна

Расстояние Левенштейна (редакционное расстояние) — метрика, позволяющая определить «схожесть» двух строк — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для конвертации одной строки в другую.

Каждая операция имеет свой вес (штраф).

Редакционным предписанием называется последовательность действий, необходимых для получения из первой строки вторую, и минимизирующую суммарную цену. Суммарная цена есть искомое расстояние Левенштейна .

Введем следующее обозначение операций:

D (delete) — удалить = 1

I (англ. insert) — вставить = 1

R (replace) — заменить = 1

M (match) — совпадение = 0

Для двух строк S1 и S2 расстояние Левенштейна можно посчитать по рекуррентной формуле:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 0, *если i*=0, *j* =0 |  |
|  |  | *i , если i*>0, *j* =0 |  |
| (1) *D*( *S* 1[ 1.. *i*] *, S* 2[ 1.. *j*])= |  | *j , если i*=0, *j*>0 |  |
|  | 1. *D*( *s* 1[1.. *i*] *, s* 2[ 1.. *j −*1])+1 |  |
|  | *min* |  |
|  | 2. *D*( *s* 1[1.. *i−* 1] *, s* 2[ 1.. *j*])+1 |  |
|  | { | {3. *D*(*s* 1[1. . *i−*1] *, s* 2[1. . *j −* 1])+(( *s* 1[ *i*]=*s* 2[ *j*])*?* 0 :1) |  |

Здесь, 1 - вставка символа, 2 - удаление символа, 3 - замена символа, при этом, если S1[i - 1] == S2[j - 1], то вес такой операции будет равен нулю, иначе единице.

Можно решать задачу нахождения расстояния Левенштейна рекурсивно в соответствии с формулами, обрабатывая подстроки, пока не будет

достигнут тривиальный случай (тривиальные случаи описаны в формуле до взятия минимума от трех случаев). Работать с подстроками затратно по объему памяти, также в рекурсивном алгоритме будут обрабатываться одинаковые случаи несколько раз, что неэффективно как по памяти, так и по времени.

Существует другое решение — матричное. Матрица размером (length(S1)

* 1)x((length(S2) + 1), где length(S) — операция определения длины строки S. Значение в ячейке [i, j] равно значению D(s1[1...i], s2[1...j]). Первая строка и первый столбец тривиальны. В соответствии с формулой получаем: A[i][j] = min (A[i-1][j] + 1, A[i][j-1] + 1, A[i-1][j-1] + m(s1[i], s2[j])), где m(s1[i], s2[j]) — это индикаторная функция, равная нулю при S1[i - 1] == S2[j - 1] и 1 в противном случае.

Результатом (расстоянием Левенштейна между двумя строками) будет ячейка матрицы с индексами i = length(S1) и j = length(S2) (правый нижний угол).

Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла в левый верхний, на каждом шаге ища минимальное из трёх значений:

• если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j);

• если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1);

• если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1) Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трех значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.

1.2. Расстояние Дамерау-Левенштейна

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, определенных в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов: T (transposition) — транспозиция = 1.

Для двух строк S1 и S2 расстояние Дамерау-Левенштейна можно посчитать по рекуррентной формуле:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | 0, *если i*=0, *j*=0 |  |
|  |  |  |  | *i , еслиi*>0, *j*=0 |  |
|  |  |  |  | *j , если i*=0, *j*>0 |  |
| (2) *D*( *S* 1[ 1.. *i*] *, S* 2[ 1 .. *j*])= |  |  |  | 1. *D*( *s* 1[ 1.. *i*] *, s* 2[ 1.. *j −*1])+1 |  |
|  |  |  | 2. *D*( *s* 1[ 1.. *i−*1] *, s* 2[ 1.. *j*])+1 |  |
|  | *min* |  |  |  |
|  | 3. *D*(*s* 1[1. .*i −*1] *, s* 2[1. . *j−* 1])+(( *s* 1[ *i* ]=*s* 2[ *j*])*?* 0 :1) | | |  |
|  | { | 4. *если* |  | *i , j*>1) *и*( *s* 1[ *i−*1]=*s* 2[ *j*]) *и*( *s* 1[ *i*]=*s* 2[ *j −*1]) |  |
|  | { | ( | *D*( *s* 1[1.. *i−* 2]*, s* 2[1.. *j −*2])+1,  , иначе |  |

Здесь, 1 - 3 имеют те же значения, что и при определении расстояния Левенштейна, а 4 - перестановка символов только в том случае, если S1[i - 1] = S2[j] и S1[i] = S2[j - 1].

Решения задачи нахождения расстояния Дамерау-Левенштейна аналогичны решениям задачи нахождения расстояния Левенштейна. При рекурсивном решении добавляется еще одна ветвь в дерево рекурсий, когда проверяется условие того, что два заключительных символа очередной подстроки равны двум другим из второй подстроки с учетом транспозиции. Для матричной реализации:

матрица размером (length(S1) + 1)x((length(S2) + 1). Значение в ячейке [i, j] равно значению D(s1[1...i], s2[1...j]). Первая строка и первый столбец тривиальны. В соответствии с формулой получаем значение в других

ячейках таблицы: A[i][j] = min (A[i-1][j] + 1, A[i][j-1] + 1, A[i-1][j-1] + m(s1[i], s2[j]), A[i - 2][j - 2] + 1 если s1[i] = s2[j-1]и s1[i-1] = s2[j] и i > 1 и j > 1). Результат также будет находится в правом нижнем углу.

Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла в левый верхний, на каждом шаге ища минимальное из трёх значений:

* если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j);
* если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1);
* если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1)
* если транспозиция возможна, то в минимум входит также (D(i-2, j-2) + 1)

1.3. Практическое применение

Данные алгоритмы применяются при поиске информации по запросу, с помощью них можно найти наиболее подходящее(имеющее наименьшее расстояние) к нему слово и заменить его в поисковой строке. Метод оценки редакционного расстояния активно применяется в биологии для оценки мутаций.

1. Конструкторская часть

2.1. Схемы алгоритмов

Рис. 2.1 - рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна.

Start R

Input str1,str2

Str1[i-1]= str2[j-1]

length2 == 0

Return length2

S = R(str1[:(len(str1)-1)],str2[:(len(str2)-1)]) + cost

d= R(str1[(len(str1)-1)],str2) + 1

i = R(str1,str2[:(len(str2)-1)]) + 1

Cost = 0

Cost =1

Return length1

Length1 == 0

length 1- length str1

length2 – length str2

End R

Return min(d,i,s)

J =1

Calculate previous row and current row

I < length2+ 1

Get the current row

i = 1

str1,str2 =str2,str1

length1,length2 = length2,length1

length1> length2

length1 = length str1

length2 = length str2

Input str1,str2

start

J += 1

i += 1

Current\_row =min(insert,delete,subsitution)

Subsitution += 1

str1[j-1] != str2[i-1]

Insert ,delete,subsitution

J < length1 + 1

2.2. Сравнительный анализ рекурсивной и нерекурсивной реализаций

Для реализации рекурсивного алгоритма поиска расстояния Дамерау-Левенштейна можно воспользоваться формулой (2) из первого раздела, приняв D за вызов рекурсивной функции. При каждом рекурсивном вызове необходимо передавать в функцию подстроки исходных строк, что явно затратно по памяти. Эту проблему возможно избежать, если занести строки в глобальные переменные и использовать индексы.

1. Технологическая часть

3.1. Требования к ПО

Программа на вход получает две строки символов. Результат работы программы: число - искомое расстояние. Для матричных реализаций дополнительно выводится получившаяся в результате работы программы матрица.

3.2. Средства реализации

* данной работе использовался язык программирования Python. Для замера времени использовалась дополнительный модуль time.

3.3. Листинги кода

В листинге 1 показана реализация алгоритма поиска расстояния Левенштейна.

def levens(word1,word2):

length1 ,length2 = len(word1), len(word2)

if length1 > length2 :

word1, word2 = word2, word1

length1 ,length2 = length2, length1

current\_row = range(length1 + 1)

for i in range(1,length2 + 1):

previous\_row,current\_row = current\_row,[i] + [0] \* length1

for j in range(1,length1 + 1):

insert, delete, subsitution = previous\_row[j] + 1,current\_row[j-1] + 1,previous\_row[j-1]

if word1[j -1] != word2[i - 1]:

subsitution += 1

current\_row[j] = min(insert, delete,subsitution)

return current\_row[length1]

В листинге 2 показана рекурсивная реализация алгоритма дистанционного поиска Дамерау-Левенштейна.

def levensRec(str1,str2):

length1 = len(str1)

if(length1 == 0):

return len(str2)

length2 = len(str2)

if(length2 == 0):

return len(str1)

if(str1[0] == str2[0]):

cost = 0

else:

cost = 1

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

(levensRec(str1,str2[1:]) + 1),

(levensRec(str1[1:],str2[1:]) + cost))

В листинге 3 показана модифицированная реализация алгоритма дистанционного поиска Дамерау-Левенштейна. Листинг 3

Листинг 3:

def levensAdj(str1,str2):

length1, length2 = len(str1), len(str2)

if (length1 < 2 or length2 < 2):

return levens(str1, str2)

if length1 > length2:

str1, str2 = str2, str1

length1, length2 = length2, length1

prev\_row = range(length1 + 1)

curr\_row = [1] + [0] \* length1

for i in range(1, length1 + 1):

insert, delete, subsitution = prev\_row[i] + 1, curr\_row[i - 1] + 1, prev\_row[i - 1]

if str1[i - 1] != str2[0]:

subsitution += 1

curr\_row[i] = min(insert, delete, subsitution)

for i in range(2, length2 + 1):

prev\_prev\_row = prev\_row; prev\_row = curr\_row; curr\_row = [i] + [0] \* length1

insert, delete, subsitution = prev\_row[1] + 1, curr\_row[0] + 1, prev\_row[0]

if (str1[0] != str2[i - 1]):

subsitution += 1

curr\_row[1] = min(insert, delete, subsitution)

for j in range(2, length1 + 1):

insert, delete, subsitution = prev\_row[j] + 1, curr\_row[j - 1] + 1, prev\_row[j - 1]

if (str1[j - 1] != str2[i - 1]):

subsitution += 1

if (str2[i - 1] == str1[j - 2] and str2[i - 2] == str1[j - 1]):

transp = prev\_prev\_row[j - 2] + 1

curr\_row[j] = min(insert, delete, subsitution, transp)

else:

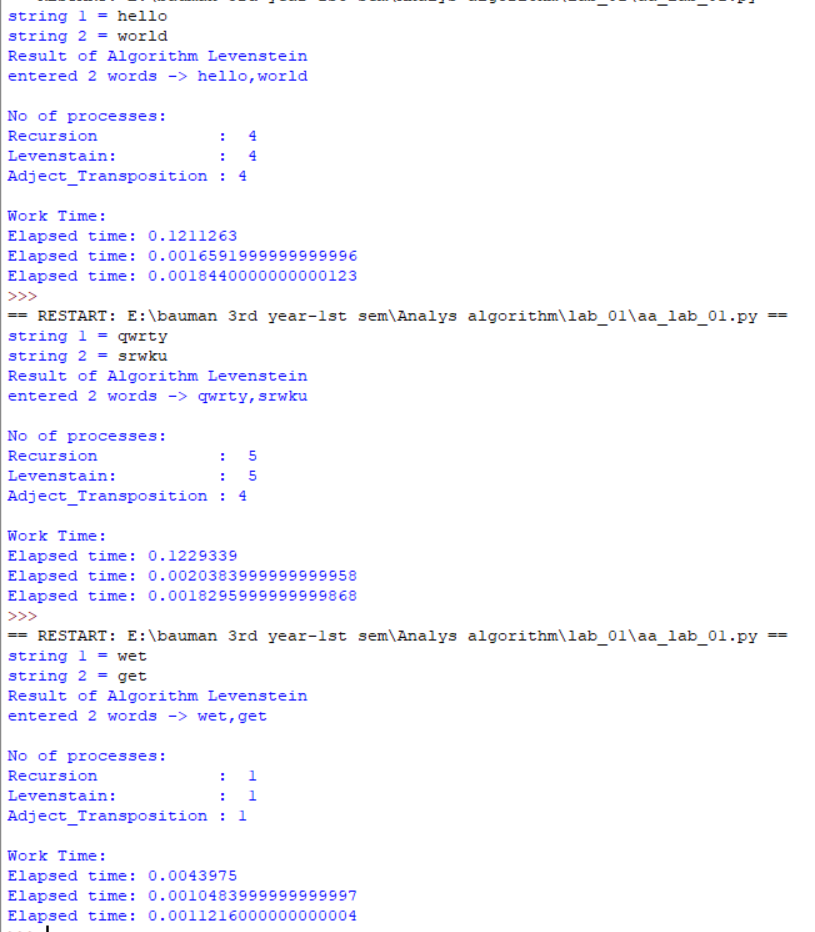
curr\_row[j] = min(insert, delete, subsitution)

return curr\_row[length1]

1. Экспериментальная часть

4.1. Примеры работы программы

Ниже представлены примеры работы программы:



4.2. Результаты функционального тестирования

* таблице 1 представлены результаты функционального тестирования программы.

Table 1

Тестовые случаи (классы эквивалентности)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | String 1 | String 2 | Ожидаемый результат | Полученный результат |
| 1 | sri | lanka | 5,5,5 | 5,5,5 |
| 2 | summer | summer | 0,0,0 | 0,0,0 |
| 3 | cat | way | 2,2,2 | 2,2,2 |
| 4 | life | sun | 4,4,4 | 4,4,4 |
| 5 | happy | angry | 4,4,4 | 4,4,4 |
| 6 | sort | crop | 4,4,4 | 4,4,3 |
| 7 | elephant | monkey | 8,8,8 | 8,8,8 |
| 8 | dog | cow | 2,2,2 | 2,2,2 |

В приведенной выше таблице столбцы «Ожидаемый результат» и «Полученный результат» содержат 3 числа - результаты функций:

1) алгоритм нахождения расстояния Левенштейна,

2) алгоритм рекурсивного поиска Дамерау-Левенштейна,

3) модифицированный алгоритм для расстояния Дамерау-Левенштейна.

4.3. Сравнительный анализ на основе экспериментальных данных

Ниже приведены графики зависимости времени, затраченного на работу алгоритмов (в секундах) от длины линий (см. рис. 1-2).

Рисунок 1 – Измерение времени выполнения программной реализации рекурсивного алгоритма Левенштейна и алгоритмов Левенштейна

Диаграмма 2

Измерение времени выполнения программной реализации рекурсивного алгоритма Левенштейна и модифицированных алгоритмов Дамарау Левенштейна.

4.4. Выводы из эксперимента

когда мы сравниваем результаты наших практических испытаний, там, где мы не хотим использовать модифицированный алгоритм levenstain, минимальное время, которое мы можем видеть, используя алгоритм levenstain.

когда это событие для использования модифицированного алгоритма damarau levenstain, мы видим, что минимальное время используется в модифицированном алгоритме.

Как правило, максимальное время получения рекурсивного алгоритма levenstain.

Заключение

Цель данной лабораторной работы была достигнута. В ходе работы были изучены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна (рекурсивный и итеративный). Выполнено сравнение рекурсивного и итеративного алгоритмов Левенштейна. Изучены зависимости времени выполнения алгоритмов от длин строк. Также были реализованы 3 описанных алгоритма нахождения расстояний Левенштейна и Дамерау-Левенштейна.