Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет имени Н.Э. Баумана»

**ОТЧЕТ**

По лабораторной работе №1

По курсу «Анализ алгоритмов»

Тема: «Алгоритм Левенштейна»

Студент: Жарова Е.А.

Группа ИУ7-51

Москва, 2017

# Оглавление

Постановка задачи....1
Описание алгоритма....1
Реализация....3
1) Базовый алгоритм....3
2) Модифицированный алгоритм....4
3) Рекурсивный алгоритм....5
Примеры работы (тесты)....5
Сравнение: модифицированного и базового....5
1) Сравнение по времени....5
2) Сравнение по памяти....6
Заключение....6

# Постановка задачи

Для превращения одной строки в другую используется алгоритм Левенштейна: происходит поиск редакционного расстояния между двумя строками. Мы рассмотрим следующие его реализации: с помощью матрицы, рекурсивный и модифицированный алгоритмы. Так же необходимо сравнить базовый и модифицированный алгоритмы Левенштейна.

# Описание алгоритма

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

Расстояние Левенштейна и его обобщения активно применяется:

- для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи).
- для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы.
- в биоинформатике для сравнения генов, хромосом и белков.

Допускаются следующие операции, каждая операция имеет свою стоимость:

| № | Операция | Стоимость операции |
|---|---|---|
| 1 | Вставка символа (I, Insert) | 1 |
| 2 | Удаление символа (D, delete) | 1 |
| 3 | Замена символа (R, replace) | 1 |
| 4 | Совпадение символа (M, match) | 0 |

Например, чтобы получить из строки «исчо» строку «ещё», мы 3 раза используем операцию замены и 1 раз – операцию удаления:

| | | | |
|---|---|---|---|
| И | С | Ч | О |
| Е | Щ | Ё | |
| R | R | R | D |

Пусть S1 и S2 — две строки (длиной M и N соответственно), тогда редакционное расстояние (расстояние Левенштейна) d(S1, S2) можно подсчитать по следующей формуле:

d(S1, S2) = D(M, N), где

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ \min\{ & \\ \quad D(i, j-1) + 1, & \\ \quad D(i-1, j) + 1, & j > 0, i > 0 \\ \quad D(i-1, j-1) + m(S1[i], S2[j]) & \\ \} & \end{cases}$$

где m(a, b) равна нулю, если a = b и единице в противном случае.

Здесь шаг по I символизирует удаление (D) из первой строки, по j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Модифицированный алгоритм Левенштейна

Существует несколько вариантов реализации алгоритма. В данной работе используется модифицированный алгоритм, который называется «Расстояние Дамерау-Левенштейна» (Дамерау показал, что большая часть ошибок, при наборе текста человеком, связана с транспозициями).

Для вычисления редакционного расстояния вводится дополнительная операция – перестановка (транспозиция) символов.

Модифицированная формула выглядит следующим образом:

$$d_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} d_{a,b}(i-1,j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \\ d_{a,b}(i-2,j-2)+1 \end{cases} & \text{if } i,j > 1 \text{ and } a_i = b_{j-1} \text{ and } a_{i-1} = b_j \\ \min \begin{cases} d_{a,b}(i-1,\ j)+1 \\ d_{a,b}(i,j-1)+1 \\ d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise,} \end{cases}$$

где $1_{(a_i \neq b_j)}$ это индикаторная функция, равная нулю при $a_i = b_j$ и 1 в противном случае.

$d_{a,b}(i-1,j)+1$ - соответствует удалению символа (из $a$ в $b$),

$d_{a,b}(i,j-1)+1$ - соответствует вставке (из $a$ в $b$),

$d_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)}$ - соответствие или несоответствие, в зависимости от совпадения символов,

$d_{a,b}(i-2,j-2)+1$ - в случае перестановки двух последовательных символов.

# Реализация

1) Базовый алгоритм

```
public static int BaseLev(string s1, string s2)
{
    int LengthS1 = s1.Length;
    int LengthS2 = s2.Length;
    int delete, insert, replace;
    int[,] Matrix = new int[LengthS1+1, LengthS2+1];

    // Операция удаления
    for (int q = 0; q <= LengthS2; q++)
    {
        Matrix[0, q] = q;
    }

    // Операция вставки
    for (int k = 0; k <= LengthS1; k++)
    {
        Matrix[k, 0] = k;
    }

    for (int i = 1; i <= LengthS1; i++)
    {
        for (int j = 1; j <= LengthS2; j++)
```

```
            {
                // Вычисление редакционного расстояния Левенштейна
                delete = Matrix[i, j - 1] + 1;
                insert = Matrix[i - 1, j] + 1;
                replace = Matrix[i - 1, j - 1];
                if (s1[i-1] != s2[j-1])
                {
                    replace += 1;
                }

                Matrix[i, j] = Math.Min(delete, Math.Min(insert, replace));
            }
        }

        return Matrix[LengthS1, LengthS2];
    }
```

2) Модифицированный алгоритм

```
    public static int ModfiedLev(string s1, string s2)
    {
        int LengthS1 = s1.Length;
        int LengthS2 = s2.Length;
        int delete, insert, replace, transp;
        int[,] Matrix = new int[LengthS1 + 1, LengthS2 + 1];

        // Операция удаления
        for (int k = 0; k <= LengthS1; k++)
        {
            Matrix[k, 0] = k;
        }

        // Операция вставки
        for (int q = 0; q <= LengthS2; q++)
        {
            Matrix[0, q] = q;
        }

        for (int i = 1; i <= LengthS1; i++)
        {
            for (int j = 1; j <= LengthS2; j++)
            {
                // Вычисление редакционного расстояния Дамерау – Левенштейна
                if ((i > 1) && (j > 1))
                {
                    delete = Matrix[i, j - 1] + 1;
                    insert = Matrix[i - 1, j] + 1;
                    replace = Matrix[i - 1, j - 1];
                    transp = Matrix[i - 2, j - 2] + 1;
                    if (s1[i - 1] != s2[j - 1])
                    {
                        replace += 1;
                    }
                    Matrix[i, j] = Math.Min(transp,
                                   Math.Min(delete,
                                   Math.Min(insert, replace)));
                }
                else
                {
                    delete = Matrix[i, j - 1] + 1;
                    insert = Matrix[i - 1, j] + 1;
                    replace = Matrix[i - 1, j - 1];
                    if (s1[i - 1] != s2[j - 1])
                    {
                        replace += 1;
```

```
                    }
                    Matrix[i, j] = Math.Min(delete, Math.Min(insert, replace));
                }
            }
        }
        return Matrix[LengthS1, LengthS2];
    }
```

3) Рекурсивный алгоритм

```
    // Рекурсивный алгоритм Левенштейна
    public static int DistRecLev(string s1, int i, string s2, int j)
    {
        if (i == 0)
        {
            return j;
        }

        if (j == 0)
        {
            return i;
        }

        int tmp = Math.Min(DistRecLev(s1, i - 1, s2, j),
                           DistRecLev(s1, i, s2, j - 1)) + 1;
        int t = 0;

        if (s1[i-1]!=s2[j-1])
        {
            t = 1;
        }

        tmp = Math.Min(DistRecLev(s1, i-1, s2, j-1) + t, tmp);
        return tmp;
    }

    public static int FindDistRecLev(string s1, string s2)
    {
        return DistRecLev(s1, s1.Length, s2, s2.Length);
    }
```

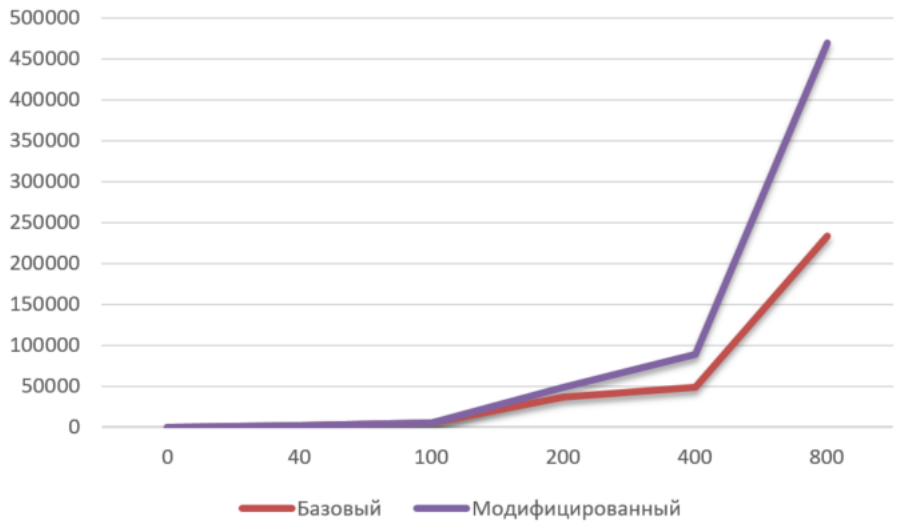
# Примеры работы (тесты)

| Строка S1 | Строка S2 | Результат | |
|---|---|---|---|
| | | Базовый | Модифицированный |
| EGHIJKL | EHIXKML | 3 | 3 |
| asdfgh | asfdgh | 2 | 1 |
| abcdefigh | abcdiefgh | 2 | 2 |
| cdef | def | 1 | 1 |
| метра | матрица | 3 | 3 |
| mbn | nm | 3 | 2 |
| c | d | 1 | 1 |

# Сравнение: модифицированного и базового

1) Сравнение по времени

Провели сравнение базового и модифицированного алгоритмов. Время работы от длины входных строк зависит квадратично.




По горизонтали – длина входных строк в символах, по вертикали – среднее количество тиков.

В модифицированных алгоритмах осуществляется дополнительная проверка, поэтому их время работы соответственно выше, чем в базовой реализации.

2) Сравнение по памяти

В базовом и модифицированном алгоритмах используется матрица размерности M×N. Следовательно, общие затраты по памяти равны O(n×m). А модифицированный алгоритм ещё должен хранить индексы вхождений для каждого символа, которые могут быть поданы на вход в строках.

Рекурсивный алгоритм использует память за счет стека вызовов функций. Так как спуск в рекурсию осуществляется для каждой подстроки входных строк, затрачиваемое количество времени и памяти для этого алгоритма значительно превышает показатели работы нерекурсивной версии алгоритма.

# Заключение

В течение выполнения работы был изучен алгоритм Левенштейна поиска редакционного расстояния между строками. Были реализованы базовый, модифицированный и рекурсивный алгоритмы. Далее было произведено сравнение модифицированного и базового алгоритмов Левенштейна.