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: Кононенко С. Д.
Группа: ИУ7-51

Москва, 2017

Расстояние Левенштейна (также редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике — 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 между двумя строками **S1** и **S2** (длиной **M** и **N** соответственно) задаётся рекуррентно:

$\mathrm{d}(S_1, S_2) = D(M, N)$, где

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

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

Если к списку разрешённых операций добавить транспозицию (два соседних символа меняются местами), получается расстояние **Дамерау — Левенштейна.**
Расстояние Дамерау — Левенштейна между двумя строками **a** и **b** определяется функцией $\mathbf{d_{a,b}(|a|, |b|)}$ как:

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

где $1_{(ai \neq bj)}$ это индикаторная функция, равная нулю при ai = bj т 1 в противном случае

Алгоритм Левенштейна

```
int Levenstein_simple(const char* const  s1, const char* const s2)
{
    int len1 = strlen(s1);
    int len2 = strlen(s2);

    if (len1 == 0 || len2 == 0)
        return (max(len1, len2));

    int n = len1 + 1, m = len2 + 1;
    int** matr = allocate_matrix(n, m);

    for (register int i = 0; i <= len1; ++i)
        matr[i][0] = i;
    for (register int i = 0; i <= len2; ++i)
         matr[0][i] = i;

    for (register int i = 1; i < n; ++i)
        for (register int j = 1; j < m; ++j)
        {
            matr[i][j] = min3(matr[i][j - 1] + 1, matr[i - 1][j] + 1, matr[i-
1][j-1] +
                    (s1[i-1] == s2[j-1] ? 0 : 1));
        }

    int result = matr[len1][len2];
    free(matr);
    return result;
}
```

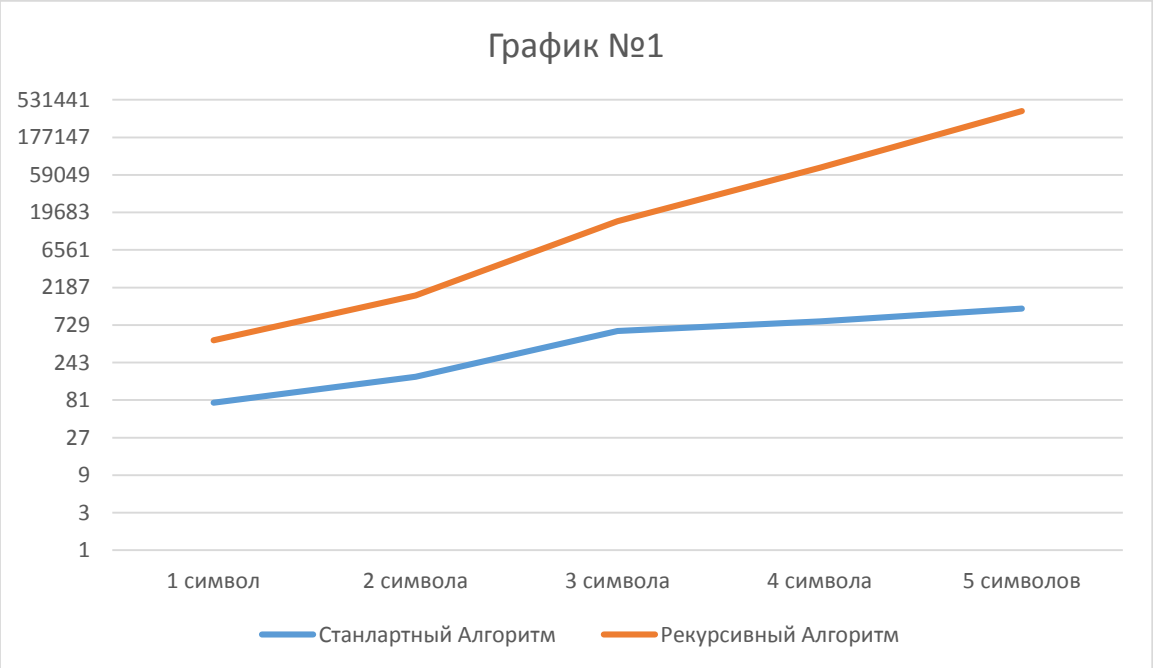
Алгоритм Левенштейна(рекурсивный)

```
static int l_r(const char * const s1, const char * const s2, const int i,
        const int j)
{
    //т.к. нумерация с 0
    if (i < 0 || j < 0)
        return max(i + 1, j + 1);

    return min3(l_r(s1, s2, i-1, j) +1 , l_r(s1, s2, i, j-1) + 1,
                l_r(s1, s2, i-1, j-1) + (s1[i] == s2[j] ? 0 : 1));
}
```

Алгоритм Левенштейна(модифицированный)

```
int Levenstein_Damer(const char* const  s1, const char* const s2)
{
    int len1 = strlen(s1);
    int len2 = strlen(s2);

    if (len1 == 0 || len2 == 0)
        return (max(len1, len2));

    int n = len1 + 1, m = len2 + 1;
    int** matr = allocate_matrix(n, m);

    for (register int i = 0; i <= len1; ++i)
        matr[i][0] = i;
    for (register int i = 1; i <= len2; ++i)
         matr[0][i] = i;

    for (register int i = 1; i < n; ++i)
        for (register int j = 1; j < m; ++j)
        {
            //проверка условия change
            if (i > 1 && j > 1 && s1[i-1] == s2[j-2] && s1[i-2] == s2[j-1])
                matr[i][j] = min4(matr[i][j - 1] + 1, matr[i - 1][j] + 1,
                    matr[i-1][j-1] + (s1[i-1] == s2[j-1] ? 0 : 1), matr[i-
2][j-2] + 1);
            else
                matr[i][j] = min3(matr[i][j - 1] + 1, matr[i - 1][j] + 1,
                    matr[i-1][j-1] + (s1[i-1] == s2[j-1] ? 0 : 1));
        }
    int result = matr[len1][len2];
    free(matr);
    return result;
}
```

Пример работы



Примечание. В графике по оси OY используется логарифмическая шкала.

Заключение

В результате выполнения лабораторной работы были экспериментально получены временные характеристики работы алгоритмов.