

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №12
дисциплины «Алгоритмизация»
Вариант 14

Выполнил:
Касимов Асхаб Арсенович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических
наук, доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Ход работы

Задание. Выполнить алгоритм Левенштейна – расстояние редактирования.

Для поиска расстояния редактирования используются 2 способа:

1) Алгоритм динамического программирования сверху вниз:

Инициализация

создать двумерный массив $D[0 \dots n, 0 \dots m]$
инициализировать все ячейки значением ∞

Функция EDITDISTTD(i, j)

```
если  $D[i, j] = \infty$ :  
    если  $i = 0$ :  $D[i, j] \leftarrow j$   
    иначе если  $j = 0$ :  $D[i, j] \leftarrow i$   
    иначе:  
         $ins \leftarrow EDITDISTTD(i, j - 1) + 1$   
         $del \leftarrow EDITDISTTD(i - 1, j) + 1$   
         $sub \leftarrow EDITDISTTD(i - 1, j - 1) + \text{diff}(A[i], B[j])$   
         $D[i, j] \leftarrow \min(ins, del, sub)$ 
```

Рисунок 1 – Алгоритм 1

Реализация данного алгоритма представлена в файле levinshtein.cpp на строчках 9-25.

2) Алгоритм динамического программирования снизу вверх:

Дин. прог. снизу вверх

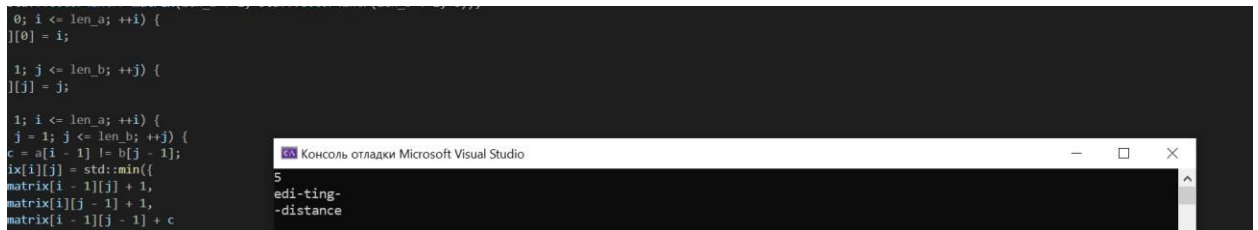
Функция EDITDISTBU($A[1 \dots n], B[1 \dots m]$)

```
создать массив  $D[0 \dots n, 0 \dots m]$   
для  $i$  от 0 до  $n$ :  
     $D[i, 0] \leftarrow i$   
для  $j$  от 0 до  $m$ :  
     $D[0, j] \leftarrow j$   
• для  $i$  от 1 до  $n$ :  
    • для  $j$  от 1 до  $m$ :  
        •  $c \leftarrow \text{diff}(A[i], B[j])$   
         $D[i, j] \leftarrow \min(D[i-1, j] + 1, D[i, j-1] + 1, D[i-1, j-1] + c)$   
вернуть  $D[n, m]$ 
```

Рисунок 2 – Алгоритм 2

Реализация данного алгоритма представлена в файле levinshtein.cpp на строчках 28-49.

Также был реализован, согласно видеоролику, алгоритм восстановления матрицы в файле levinshtein.cpp на строчках 52-76.



```
0; i <= len_a; ++i) {  
    j[0] = i;  
  
    1; j <= len_b; ++j) {  
        j[j] = j;  
  
        1; i <= len_a; ++i) {  
            j = 1; j <= len_b; ++j) {  
                c = a[i - 1] != b[j - 1];  
                ix[i][j] = std::min({  
                    matrix[i - 1][j] + 1,  
                    matrix[i][j - 1] + 1,  
                    matrix[i - 1][j - 1] + c  
                });  
            }
```

Рисунок 3. Результат работы алгоритма

Вывод: В ходе выполнения лабораторной работы был изучен алгоритм Левенштейна, который используется для нахождения расстояния редактирования. Также были изучены различные методы поиска расстояния редактирования. Один из них основан на рекурсивном подходе, при котором для вычисления верхних значений используются все нижние, а заполнение матрицы происходит по порядку снизу вверх.