Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по курсу «Анализ алгоритмов»

на тему:

«Редакционные расстояния»

Студент Рунов К. А.

Группа ИУ7-54Б

Преподаватели Волкова Л. Л., Строганов Д. В.

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Редакционные расстояния — расстояние Левенштейна и его модификация — расстояние Дамерау — Левенштейна — метрики сходства между двумя символьными последовательностями. Расстоянием в этих метриках считается минимальное число односимвольных преобразований (удаления, вставки, замены или транспозиции), необходимых для преобразования одной последовательности символов в другую.

Редакционные расстояния применяются

— для исправления ошибок в слове поискового запроса;

— в формах заполнения информации на сайтах;

— для распознавания рукописных символов;

— в базах данных. [1]

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов поиска расстояний Левенштейна и Дамерау — Левенштейна.

Для достижения поставленной цели нужно решить следующие задачи:

1) описать алгоритмы поиска расстояний Левенштейна и Дамерау — Левенштейна;

2) обосновать выбор средств реализации алгоритмов;

3) реализовать алгоритмы:

   — итеративный алгоритм нахождения расстояния Левенштейна,

   — итеративный алгоритм нахождения расстояния Дамерау — Левенштейна,

   — рекурсивный алгоритм нахождения расстояния Дамерау — Левенштейна,

   — рекурсивный с кешированием алгоритм нахождения расстояния Дамерау — Левенштейна;

4) провести сравнительный анализ алгоритмов по критериям:

— используемое процессорное время,

— максимальная затрачиваемая память;

5) описать и проанализировать полученные результаты в отчёте.

1 Аналитическая часть

1.1 Расстояние Левенштейна

1.1.1 Итеративный алгоритм нахождения расстояния Левенштейна

1.2 Расстояние Дамерау — Левенштейна

1.2.1 Итеративный алгоритм нахождения расстояния Дамерау — Левенштейна

1.2.2 Рекурсивный алгоритм нахождения расстояния Дамерау — Левенштейна

1.2.3 Рекурсивный с кешированием алгоритм нахождения расстояния Дамерау — Левенштейна

2 Конструкторская часть

3 Технологическая часть

4 Исследовательская часть

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А. Погорелов Д., М. Таразанов А. Сравнительный анализ алгоритмов редакционного расстояния Левенштейна и Дамерау-Левенштейна // Синергия Наук. 2019. URL: — Режим доступа: https://elibrary.ru/item.asp?id=36907767 (дата обращения 27.10.2023).

ПРИЛОЖЕНИЕ А

Листинг 1 – Создание матрицы для последующего использования в реализациях алгоритмов нахождения расстояний Левенштейна и Дамерау — Левенштейна

```
size_t **create_matrix(size_t n_rows, size_t n_columns)
{
    if (n_rows < 1 || n_columns < 1)
        return NULL;

    size_t *mem = (size_t*)malloc(n_rows * n_columns *
       sizeof(size_t));

    if (mem == NULL)
        return NULL;

    size_t **matrix = (size_t**)malloc(n_rows * sizeof(size_t*));

    if (matrix == NULL)
    {
        free(mem);
        return NULL;
    }

    for (size_t i = 0; i < n_rows; i++)
    {
        matrix[i] = mem + i * n_columns;
        matrix[i][0] = i;
    }

    for (size_t j = 1; j < n_columns; j++)
        matrix[0][j] = j;

    return matrix;
}
```

Листинг 2 – Освобождение памяти из-под созданной матрицы

```
void free_matrix(size_t **matrix, size_t *first_row)
{
    free(first_row);
    free(matrix);
```

```
}
```

Листинг 3 – Нахождение минимального числа из трёх и возврат указателя на него

```
size_t *min3(size_t *a, size_t *b, size_t *c)
{
    size_t *result;
    size_t min = std::min(*a, std::min(*b, *c));

    if (min == *a)
        result = a;
    else if (min == *b)
        result = b;
    else
        result = c;

    return result;
}
```

Листинг 4 – Нахождение минимального числа из четырёх и возврат указателя на него

```
size_t *min4(size_t *a, size_t *b, size_t *c, size_t *d)
{
    size_t *result;
    size_t min = std::min(std::min(*a, *b), std::min(*c, *d));

    if (min == *a)
        result = a;
    else if (min == *b)
        result = b;
    else if (min == *c)
        result = c;
    else
        result = d;

    return result;
}
```

Листинг 5 – Часть реализации итеративного алгоритма нахождения расстояния Левенштейна, участвующая в замерах времени выполнения реализаций исследуемых алгоритмов

```
size_t lev_ifm_helper(size_t **matrix, const wchar_t *s1, size_t
    len1, const wchar_t *s2, size_t len2)
{
    size_t result = 0;
    bool replace_skip_cond;
    size_t insert_cost, delete_cost, replace_cost, *who;

    for (size_t i = 1; i < len1; i++)
    {
        for (size_t j = 1; j < len2; j++)
        {
            insert_cost = matrix[i - 1][j] + 1;
            delete_cost = matrix[i][j - 1] + 1;
            replace_skip_cond = (s1[i] == s2[j]);
            replace_cost = matrix[i - 1][j - 1] +
                (replace_skip_cond ? 0 : 1);
            who = min3(&insert_cost, &delete_cost, &replace_cost);
            matrix[i][j] = *who;
        }
    }

    result = matrix[len1 - 1][len2 - 1];

    return result;
}
```

Листинг 6 – Реализация итеративного алгоритма нахождения расстояния Левенштейна

```
size_t levenshtein_iterative_full_matrix(const wchar_t *str1,
    size_t len1, const wchar_t *str2, size_t len2)
{
    if (len1 == 0) return len2;
    if (len2 == 0) return len1;

    ++len1;
    ++len2;
    const wchar_t *s1 = str1 - 1;
    const wchar_t *s2 = str2 - 1;
```

```
10
11      size_t **matrix = create_matrix(len1, len2);
12
13      if (matrix == NULL) return -1;
14
15      size_t result = lev_ifm_helper(matrix, s1, len1, s2, len2);
16
17      free_matrix(matrix, matrix[0]);
18
19      return result;
20  }
```

Листинг 7 – Часть реализации итеративного алгоритма нахождения расстояния Дамерау — Левенштейна, участвующая в замерах времени выполнения реализаций исследуемых алгоритмов

```
1  size_t damlev_ifm_helper(size_t **matrix, const wchar_t *s1,
       size_t len1, const wchar_t *s2, size_t len2)
2  {
3      size_t result = 0;
4      bool replace_skip_cond, swap_cond;
5      size_t insert_cost, delete_cost, replace_cost, swap_cost,
           *who;
6
7      for (size_t i = 1; i < len1; i++)
8      {
9          for (size_t j = 1; j < len2; j++)
10         {
11             insert_cost = matrix[i - 1][j] + 1;
12             delete_cost = matrix[i][j - 1] + 1;
13             replace_skip_cond = (s1[i] == s2[j]);
14             replace_cost = matrix[i - 1][j - 1] +
                   (replace_skip_cond ? 0 : 1);
15             if (i >= 2 && j >= 2) [[likely]]
16             {
17                 swap_cond = (s1[i] == s2[j - 1] && s1[i - 1] ==
                       s2[j]);
18                 swap_cost = swap_cond ? matrix[i - 2][j - 2] + 1
                       : U_INF; // U_INF = -1
19                 who = min4(&insert_cost, &delete_cost,
                       &replace_cost, &swap_cost);
20             }
```

```
21                else
22                {
23                    who = min3(&insert_cost, &delete_cost,
                          &replace_cost);
24                }
25                matrix[i][j] = *who;
26            }
27        }
28
29        result = matrix[len1 - 1][len2 - 1];
30
31        return result;
32    }
```

Листинг 8 – Реализация итеративного алгоритма нахождения расстояния Дамерау — Левенштейна

```
1  size_t damerau_levenshtein_iterative_full_matrix(const wchar_t
       *str1, size_t len1, const wchar_t *str2, size_t len2)
2  {
3      if (len1 == 0) return len2;
4      if (len2 == 0) return len1;
5
6      ++len1;
7      ++len2;
8      const wchar_t *s1 = str1 - 1;
9      const wchar_t *s2 = str2 - 1;
10
11     size_t **matrix = create_matrix(len1, len2);
12     if (matrix == NULL) return -1;
13
14     size_t result = damlev_ifm_helper(matrix, s1, len1, s2, len2);
15
16     free_matrix(matrix, matrix[0]);
17
18     return result;
19 }
```

Листинг 9 – Часть реализации рекурсивного с кешированием алгоритма нахождения расстояния Дамерау — Левенштейна, участвующая в замерах времени выполнения реализаций исследуемых алгоритмов

```
size_t damlev_rwc_helper(size_t **matrix, const wchar_t *str1,
    size_t len1, const wchar_t *str2, size_t len2)
{
    if (len1 == 0) return len2;
    if (len2 == 0) return len1;

    size_t i = len1 - 1;
    size_t j = len2 - 1;

    size_t insert = (((j > 0) && (matrix[i][j - 1] != U_INF))
        ? matrix[i][j - 1]
        : damlev_rwc_helper(matrix, str1, len1, str2, len2 - 1))
        + 1;

    size_t del = (((i > 0) && (matrix[i - 1][j] != U_INF))
        ? matrix[i - 1][j]
        : damlev_rwc_helper(matrix, str1, len1 - 1, str2, len2))
        + 1;

    size_t replace = (((i > 0 && j > 0) && (matrix[i - 1][j - 1]
       != U_INF))
        ? matrix[i - 1][j - 1]
        : damlev_rwc_helper(matrix, str1, len1 - 1, str2, len2 -
            1))
        + (str1[i] == str2[j] ? 0 : 1);

    size_t swap = U_INF;
    if (i > 1 && j > 1 && matrix[i - 2][j - 2] != U_INF &&
       (str1[i] == str2[j - 1] && str1[i - 1] == str2[j]))
    {
        swap = matrix[i - 2][j - 2] + 1;
    }
    else if (i >= 1 && j >= 1 && (str1[i] == str2[j - 1] &&
       str1[i - 1] == str2[j]))
    {
        swap = damlev_rwc_helper(matrix, str1, len1 - 2, str2,
            len2 - 2) + 1;
    }

    size_t result = *min4(&insert, &del, &replace, &swap);

```

```
36      if (matrix[i][j] == U_INF) matrix[i][j] = result;
37
38      return result;
39  }
```

Листинг 10 – Реализация рекурсивного с кешированием алгоритма нахождения расстояния Дамерау — Левенштейна

```
1  size_t damerau_levenshtein_recursive_with_cache(const wchar_t
       *str1, size_t len1, const wchar_t *str2, size_t len2)
2  {
3      if (len1 == 0) return len2;
4      if (len2 == 0) return len1;
5
6      size_t **matrix = create_matrix(len1, len2);
7
8      if (matrix == NULL) return -1;
9
10     for (size_t i = 0; i < len1; i++)
11         for (size_t j = 0; j < len2; j++)
12             matrix[i][j] = U_INF;
13
14     size_t result = damlev_rwc_helper(matrix, str1, len1, str2,
           len2);
15
16     free_matrix(matrix, matrix[0]);
17
18     return result;
19 }
```

Листинг 11 – Реализация рекурсивного алгоритма нахождения расстояния Дамерау — Левенштейна

```
1  size_t damerau_levenshtein_recursive_no_cache(const wchar_t
       *str1, size_t len1, const wchar_t *str2, size_t len2)
2  {
3      if (len1 == 0) return len2;
4      if (len2 == 0) return len1;
5
6      size_t insert = damerau_levenshtein_recursive_no_cache(str1,
           len1, str2, len2 - 1) + 1;
7      size_t del = damerau_levenshtein_recursive_no_cache(str1,
           len1 - 1, str2, len2) + 1;
```

```
8       size_t replace = damerau_levenshtein_recursive_no_cache(str1,
            len1 - 1, str2, len2 - 1)
9           + (str1[len1 - 1] == str2[len2 - 1] ? 0 : 1);
10      size_t swap = (len1 >= 2 && len2 >= 2)
11          ? (
12                  (str1[len1 - 1] == str2[len2 - 2] && str1[len1 -
                        2] == str2[len2 - 1])
13                  ? damerau_levenshtein_recursive_no_cache(str1,
                        len1 - 2, str2, len2 - 2) + 1
14                  : U_INF
15              )
16          : U_INF;
17
18      return *min4(&insert, &del, &replace, &swap);
19  }
```

ПРИЛОЖЕНИЕ Б