МГТУ им. Баумана

Лабораторная работа №1

По курсу: "Анализ алгоритмов"

Расстояние Левенштейна

Работу выполнила: Серёгина Дарья, ИУ7-54Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2020

Оглавление

Введение

Расстояние Левенштейна - согласно [4] - минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для:

- исправления ошибок в слове;
- сравнения текстовых файлов (утилита diff);
- в биоинформатике для сравнения генов, хромосом и белков.

Целью данной лабораторной работы является изучение метода динамического программирования на примере алгоритмов Левенштейна и Дамерау-Левенштейна.

Задачами лабораторной работы являются:

- изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
- реализация рекурсивной и динамической вариации указанных алгоритмов;
- тестирование реализованных алгоритмов;
- проведение сравнительного анализа алгоритмов по затрачиваемым ресурсам (времени и памяти).

1 Аналитическая часть

Задача по нахождению расстояния Левенштейна заключается в поиске минимального количества операций вставки/удаления/замены для превращения одной строки в другую.

При нахождении расстояния Дамерау — Левенштейна добавляется операция транспозиции (перестановки соседних символов). Полное определение рассмотрено в [1].

Действия обозначаются так:

- D (англ. delete) — удалить;
- I (англ. insert) — вставить;
- R (replace) — заменить;
- M(match) - совпадение.

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда расстояние Левенштейна можно подсчитать по рекуррентной формуле (1.1), см [3]:

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ min(& \\ D(i,j-1) + 1, & \\ D(i-1,j) + 1, & j > 0, i > 0 \\ D(i-1,j-1) + m(S_1[i], S_2[j]) & \\ ), & \end{cases} \tag{1.1}$$

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

Расстояние Дамерау-Левенштейна вычисляется по рекуррентной формуле (1.2), см [2]:

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & i > 0, j = 0 \\ j, & i = 0, j > 0 \\ min \begin{cases} D(i,j-1)+1, \\ D(i-1,j)+1, \\ D(i-1,j-1)+m(S_1[i],S_2[i]), \\ D(i-2,j-2)+m(S_1[i],S_2[i]), \end{cases} & \text{, если } i,j > 0 \\ & \text{и } S_1[i] = S_2[j-1] \\ & \text{и } S_1[i-1] = S_2[j] \\ min \begin{cases} D(i,j-1)+1, \\ D(i-1,j)+1, \\ D(i-1,j-1)+m(S_1[i],S_2[i]), \end{cases} & \text{, иначе} \end{cases} \tag{1.2}$$

1.1 Вывод

В данном разделе были рассмотрены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна, который является модификаций первого, учитывающего возможность перестановки соседних символов.

2 | Конструкторская часть

Ввод:

- на вход подаются две строки;
- строки могут быть пустыми, содержать пробелы, а также любые печатные символы UTF-8;
- uppercase и lowercase буквы считаются разными.

Вывод:

- программа выводит посчитанные каждым из алгоритмов расстояния;
- для динамических реализаций алгоритмов выводятся заполненные матрицы;
- в режиме замера ресурсов программа выводит средние время и память, затраченные каждым алгоритмом.

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов Схемы рекурсивного алгоритма нахождения расстояния Левенштейна, матричного алгоритма нахождения расстояния Левенштейна, рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна и матричного алгоритма нахождения расстояния Дамерау-Левенштейна показаны на рисунках 2.1, 2.2, 2.3 и 2.4, соответственно.

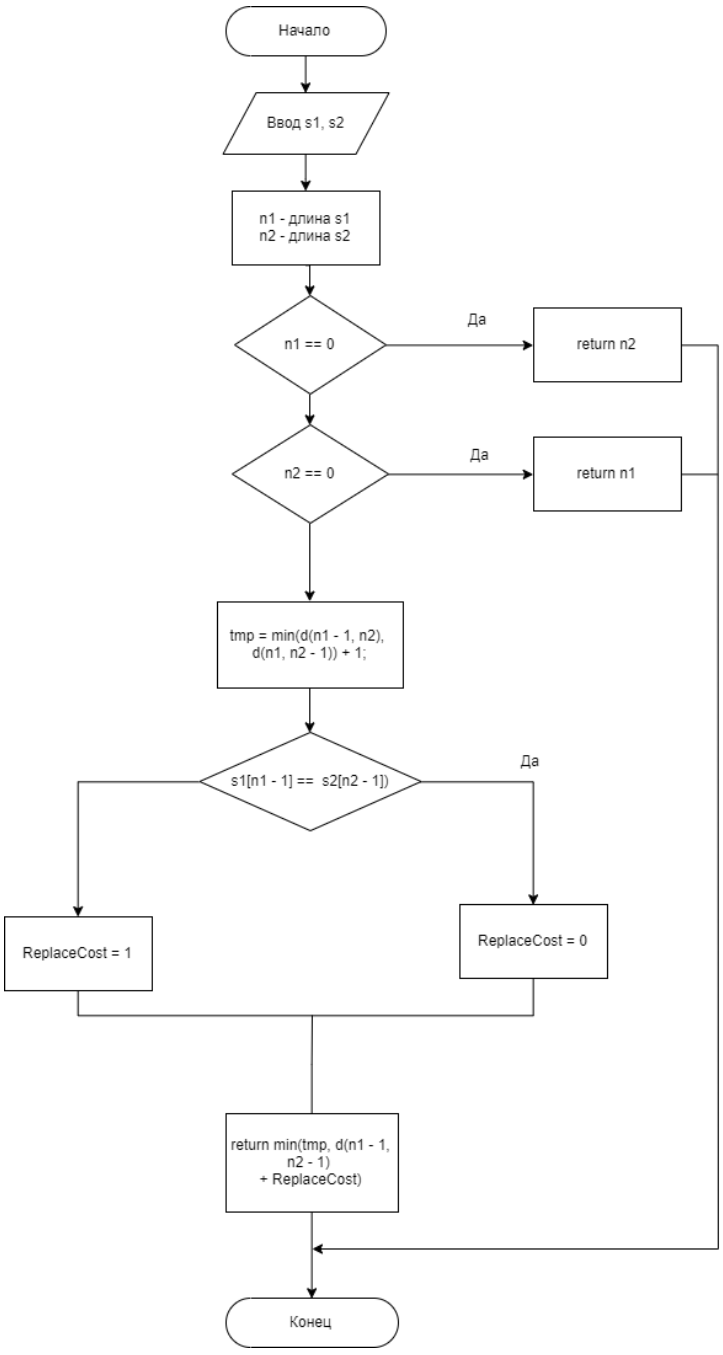

Рис. 2.1: Схема рекурсивного алгоритма нахождения расстояния Левенштейна

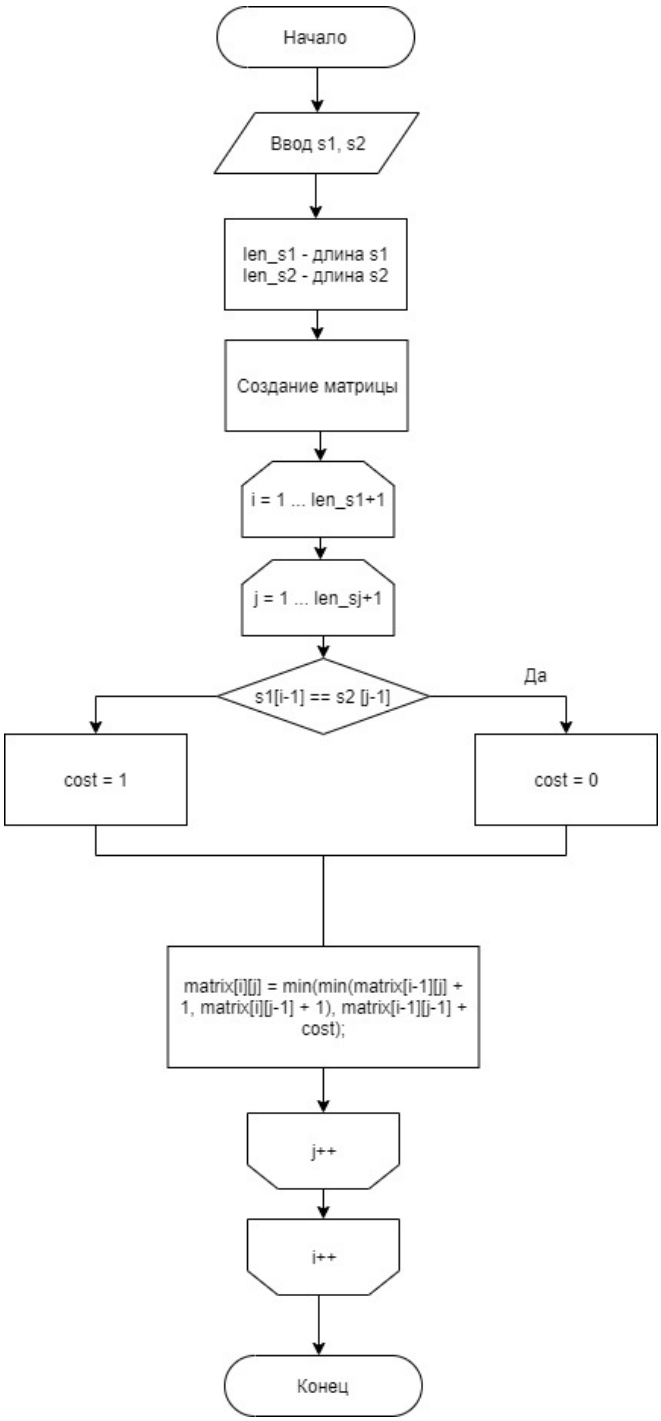

Рис. 2.2: Схема матричного алгоритма нахождения расстояния Левенштейна

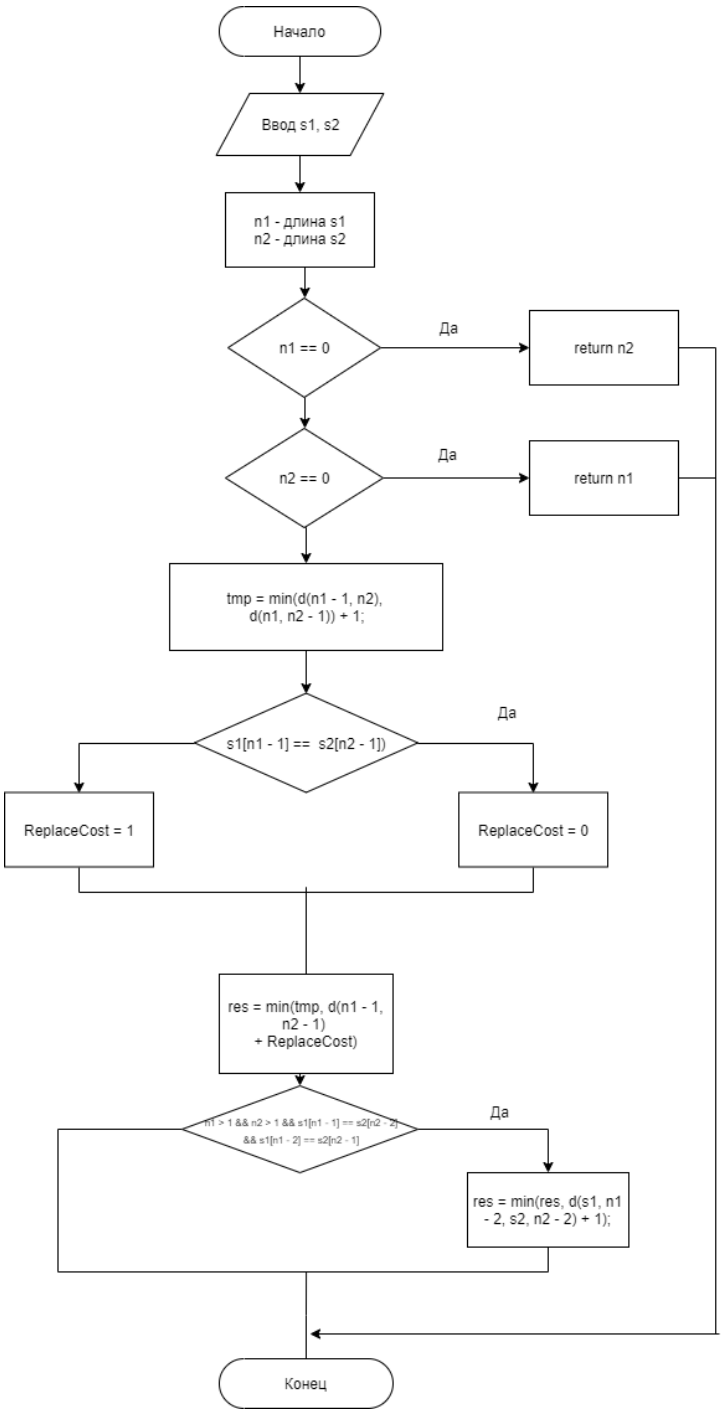

Рис. 2.3: Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна

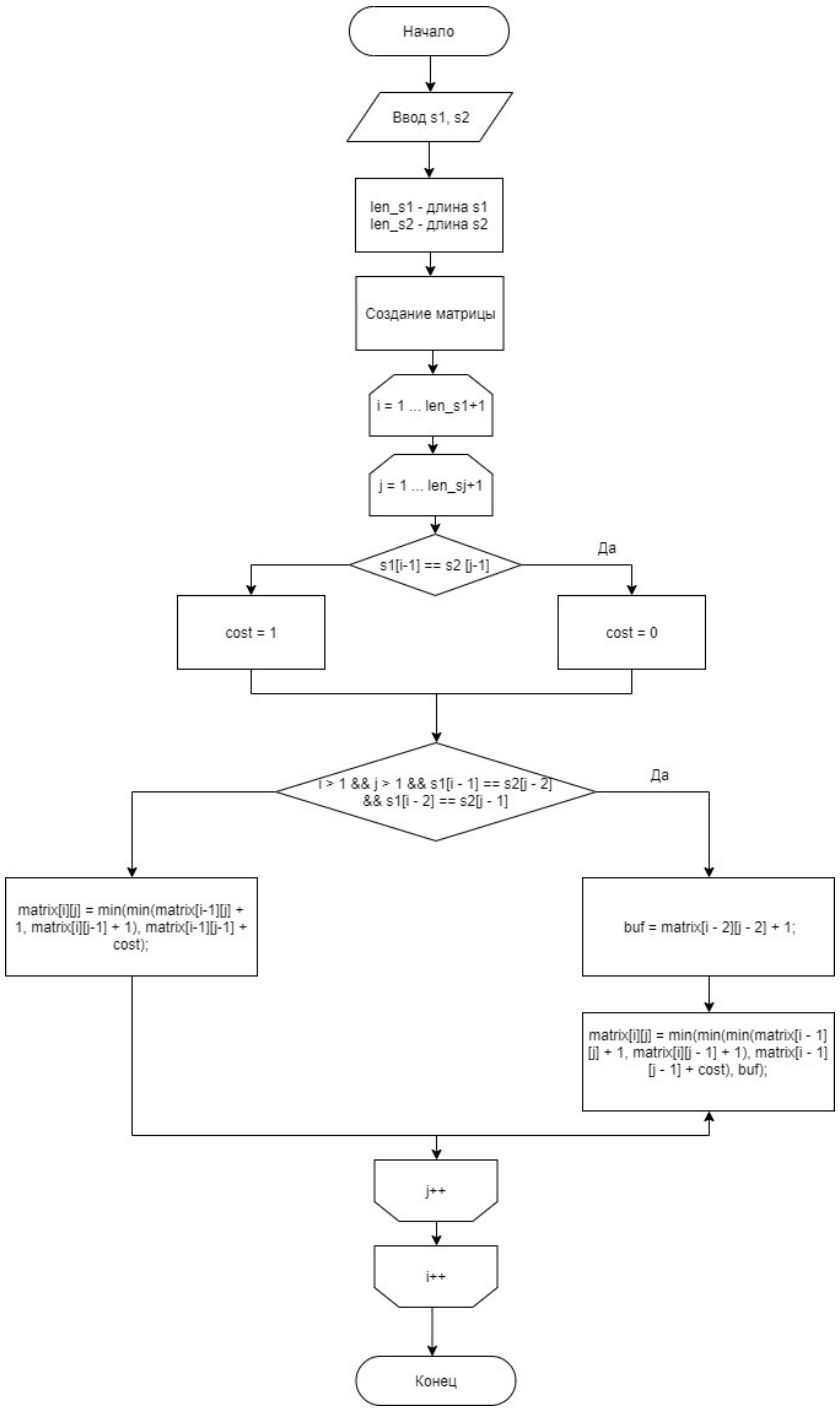

Рис. 2.4: Схема матричного алгоритма нахождения расстояния Дамерау-Левенштейна

3 | Технологическая часть

3.1 Выбор ЯП

Для реализации программы был выбран C++ из-за наличия опыта разработки на данном языке программирования. Среда разработки - Visual Studio Code.

3.2 Реализация алгоритма

Листинг 3.1: Функция нахождения расстояния Левенштейна рекурсивно

```
int levenstein_rec(string s1, string s2)
{
    int var = 1;
    int dist = 0;
    int s1_l, s2_l;

    s1_l = Llen(s1);
    s2_l = Llen(s2);


    if (s1.length() == 0 || s2.length() == 0) {
        dist = fabs(s1.length() - s2.length());
        return dist;
    }

    string s1_new = s1.substr(0, s1_l);
    string s2_new = s2.substr(0, s2_l);

```

```
19      if (s1[s1_l] == s2[s2_l]) {
20          var = 0;
21      }
22
23      return Mmin (levenstein_rec(s1_new, s2) + 1,
            levenstein_rec(s1, s2_new) + 1, levenstein_rec(
            s1_new, s2_new) + var);
24  }
```

Листинг 3.2: Функция нахождения расстояния Левенштейна матрично

```
1   int levenstein(string s1, string s2)
2   {
3       int len_s1 = s1.length() + 1;
4       int len_s2 = s2.length() + 1;
5
6       int arr[len_s1][len_s2];
7
8       for (int i = 0; i < len_s1; i++) {
9           for (int j = 0; j < len_s2; j++) {
10              if (i * j == 0) {
11                  arr[i][j] = i + j;
12              } else {
13                  arr[i][j] = 0;
14
15              }
16          }
17      }
18
19      cout << s1 << endl;
20      cout << s2 << endl;
21
22
23      for (int i = 1; i < len_s1; i++) {
24          for (int j = 1; j < len_s2; j++) {
25              int key = 1;
26              if (s1[i-1] == s2[j-1]) {
27                  key = 0;
28              }
29              arr[i][j] = Mmin(arr[i-1][j] + 1, arr[i][j-1] +
                    1,  arr[i-1][j-1] + key);
```

```
30          }
31      }
32
33      for (int i = 0; i < len_s1; i++) {
34          for (int j = 0; j < len_s2; j++) {
35              cout << arr[i][j] << " ";
36          }
37          cout << "\n";
38      }
39      cout << "\n";
40
41      return arr[len_s1 - 1][len_s2 - 1];
42  }
```

Листинг 3.3: Функция нахождения расстояния Дамерау-Левенштейна рекурсивно

```
1   int dameray_levenstein_rec(string s1, string s2)
2   {
3       int var = 1;
4       int dist = 0;
5       int s1_l, s2_l;
6
7       s1_l = Llen(s1);
8       s2_l = Llen(s2);
9
10      if (s1 == "" || s2 == "") {
11          dist = max(s1.length(), s2.length());
12          return dist;
13      }
14
15      string s1_new = s1.substr(0, s1.length() - 1);
16      string s2_new = s2.substr(0, s2.length() - 1);
17
18      if (s1[s1_l] == s2[s2_l]) {
19          var = 0;
20      }
21
22      dist = Mmin (dameray_levenstein_rec(s1_new, s2) + 1,
23      dameray_levenstein_rec(s1, s2_new) + 1,
24      dameray_levenstein_rec(s1_new, s2_new) + var);
```

```
25
26      if (s1.length() >= 2 && s2.length() >= 2 && s1[s1_l] ==
            s2[s2_l - 1] &&
27      s1[s1_l - 1] == s2[s2_l]) {
28          string s1_damer = s1.substr(0, s1.length() - 2);
29          string s2_damer = s2.substr(0, s2.length() - 2);
30          dist = Mmin(dist, dist, dameray_levenstein_rec(
                s1_damer, s2_damer) + 1);
31      }
32
33      return dist;
34  }
```

Листинг 3.4: Функция нахождения расстояния Дамерау-Левенштейна матрично

```
1  def levenshtein_damerau_matrix(str1, str2, output):
2      len_i = len(str1) + 1
3      len_j = len(str2) + 1
4      table = [[i + j for j in range(len_j)] for i in range(
           len_i)]
5
6      for i in range(1, len_i):
7          for j in range(1, len_j):
8              forfeit = 0 if (str1[i - 1] == str2[j - 1])
                   else 1
9              table[i][j] = min(table[i - 1][j] + 1,
10             table[i][j - 1] + 1,
11             table[i - 1][j - 1] + forfeit)
12             if (i > 1 and j > 1) and str1[i-1] == str2[j-2]
                   and str1[i-2] == str2[j-1]:
13                 table[i][j] = min(table[i][j], table[i-2][j
                       -2] + 1)
14     if (output):
15         OutputTable(table, str1, str2)
16     return(table[-1][-1]), memory
```

4 Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов. Результат замера показан в таблице 4.1.

Таблица 4.1: Время работы алгоритмов (в тиках)

len	Lev(R)	Lev(T)	DamLev(R)	DamLev(T)
1	1000	3818630	1100	4452931
2	2114	6861339	2420	6394623
3	6811	11611191	7507	11380591
4	30522	17771123	26990	16096365
5	115868	23581501	120615	20575176
6	561283	31673531	621520	30105181
7	6743323	41037416	7336250	41091310

Полученная зависимость времени работы алгоритмов от длины строк показана на рисунке 4.1.

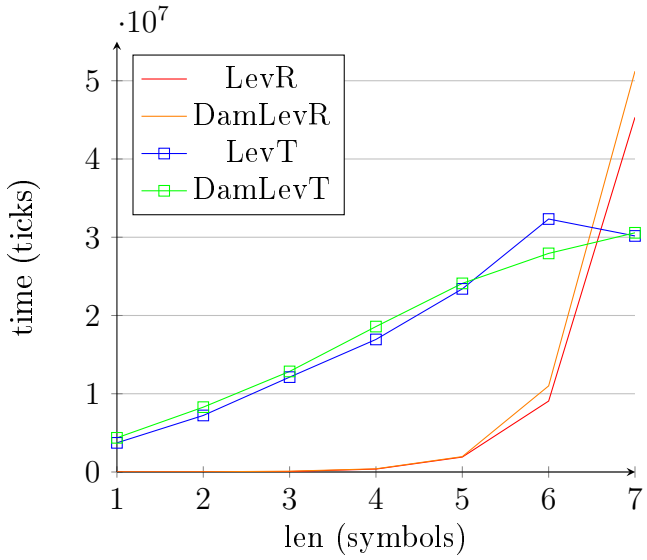

Рис. 4.1: Зависимость времени работы от длины строк

На основе проведённых измерений можно сделать вывод, что рекурсивные алгоритмы эффективней для коротких строк. Однако при увеличении длины, динамические алгоритмы выступают более эффективными, что обусловлено большим количеством повторных рассчетов в рекурсивных реализациях, в то время как в динамических реализациях ячейка матрицы расчитывается единожды. Также установлено, что алгоритм ДамерауЛевенштейна в среднем работает несколько дольше алгоритма Левенштейна, что объясняется наличием дополнительных проверок, однако алгоритмы сравнимы по временной эффективности.

4.2 Сравнительный анализ алгоритмов на основе замеров затрачиваемой памяти

Был проведен замер памяти, затрачиваемой алгоритмами. Результат замера показан в таблице 4.2.

Таблица 4.2: Количество памяти, затрачиваемой алгоритмом (в байтах)

len	Lev(R)	Lev(T)	DamLev(R)	DamLev(T)
1	34	34	38	26
2	180	56	204	40
3	1006	86	1158	62
4	5776	124	6416	92
5	76711	195	84015	151
6	197756	224	215732	176
7	1167334	286	1264610	230

Полученная зависимость памяти затрачиваемой алгоритмами от длины строк показана на рисунке 4.2.

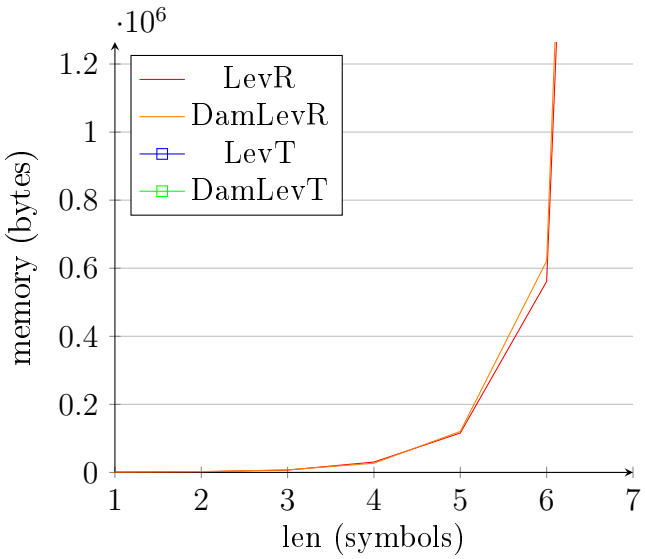

Рис. 4.2: Зависимость затрачиваемой памяти от длины строк

На основе проведённых измерений можно сделать вывод, что рекурсивные алгоритмы сравнимы по количеству затрачиваемой памяти с динамическими при малых длинах входных строк. Однако при росте длины строк количество памяти, затрачиваемой рекурсивными алгоритмами резко возрастает из-за локальных переменных, создаваемых при каждом вызове алгоритма, в то время как память динамических алгоритмов

изменяется слабо - только из-за увеличения хранимой матрицы.

4.3 Тестовые данные

Проведем тестирование программы. В столбцах "Ожидаемый результат"и "Полученный результат"4 числа соответсвуют рекурсивному алгоритму нахождения расстояния Левенштейна, матричному алгоритму нахождению расстоянию Левенштейна, рекурсивному алгоритму расстояния Дамерау-Левенштейна, матричному алгоритму нахождения расстояние Дамерау-Левенштейна.

Таблица 4.3: Таблица тестовых данных

№	Первое слово	Второе слово	Ожидаемый результат	Полученный результат
1			0 0 0 0	0 0 0 0
2	kot	skat	2 2 2 2	2 2 2 2
3	kate	ktae	2 2 1 1	2 2 1 1
4	abacaba	aabcaab	4 4 2 2	4 4 2 2
5	sobaka	sboku	3 3 3 3	3 3 3 3
6	qwerty	queue	4 4 4 4	4 4 4 4
7	apple	aplpe	2 2 1 1	2 2 1 1
8		cat	3 3 3 3	3 3 3 3
9	parallels		9 9 9 9	9 9 9 9
10	bmstu	utsmb	4 4 4 4	4 4 4 4

Заключение

Были изучены методы динамического и рекурсивного программирования на примере алгоритмов Левенштейна и Дамерау-Левенштейна. Получены практические навыки реализации указанных алгоритмов в матричной и динамической реализации.

Экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработаного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк.

В результате исследований можно прийти к вводу, что матричная реализация данных алгоритмов заметно выигрывает по времени при росте длины строк, следовательно более применима в реальных проектах.

Список использованных источников

1. Задача о расстоянии Дамерау-Левенштейна [Электронный ресурс]. – Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Задача-о-расстоянии-Дамерау-Левенштейна. – Дата доступа: 21.09.2020.

2. Вычисление редакционного расстояния [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/117063/. – Дата доступа: 21.09.2020.

3. Вычисление расстояния Левенштейна [Электронный ресурс]. – Режим доступа: https://foxford.ru/wiki/informatika/vychislenie-rasstoyaniya-levenshteyna. – Дата доступа: 21.09.2020.

4. Расстояние Левенштейна [Электронный ресурс]. – Режим доступа: https://vc.ru/newtechaudit/129654-rasstoyanie-levenshteyna-dlya-poiska-opechatok-v-dannyh-klienta. – Дата доступа: 21.09.2020.