



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

«Информатика и системы управления»

КАФЕДРА

«Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

По лабораторной работе №1

По курсу: «Анализ алгоритмов»

Тема: «Расстояния Левенштейна и Дамерау-Левенштейна»

Студент:

Пронин А. С.

Группа:

ИУ7-52Б

Преподаватель:

Волкова Л. Л.

Оценка:

Москва

2021

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Расстояние Левенштейна	4
1.2 Расстояние Дамерау-Левенштейна	5
1.3 Применение	6
2 Конструкторский раздел	7
2.1 Разработка алгоритмов	7
3 Технологический раздел	16
3.1 Выбор инструментов	16
3.2 Реализация алгоритмов	16
4 Исследовательский раздел	21
4.1 Тесты	21
4.2 Сравнительный анализ алгоритмов	21
4.3 Вывод	21
Заключение	22

Введение

Цель работы – получить навык динамического программирования.

Задачи работы:

- изучить расстояния Левенштейна;
- изучить расстояния Дамерау-Левенштейна;
- разработать алгоритм вычисления расстояния Левенштейна обычным способом (матричным);
- разработать алгоритм вычисления расстояния Левенштейна рекурсивным способом;
- разработать алгоритм вычисления расстояния Левенштейна рекурсивным способом с кэшированием;
- разработать алгоритм вычисления расстояния Дамерау-Левенштейна обычным способом (матричным);
- реализовать алгоритмы;
- провести сравнительный анализ процессорного времени выполнения реализации алгоритмов;
- провести анализ пикового значения затрачиваемой памяти в программе.

1 Аналитический раздел

1.1 Расстояние Левенштейна

Расстояние Левенштейна (базовый вид редакторского расстояния)— это минимальное количество редакций необходимое для превращения одной строки в другую.

Редакторские операции бывают:

- I (insert) - вставка
- D (delete) - удаление
- R (replace) – замена

У этих трёх операций штраф = 1.

Еще одна операция:

- M (Match) – совпадение

Эта операция не имеет штрафа (он равен нулю).

Пусть s_1 и s_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) $d(s_1, s_2)$ можно подсчитать по следующей рекуррентной

формуле:

$$|x| = \begin{cases} 0, \text{ если } i = 0, j = 0; \\ i, \text{ если } i > 0, j = 0; \\ j, \text{ если } i = 0, j > 0; \\ \min \begin{pmatrix} D(s1[1 \dots i], s2[1 \dots j - 1]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j - 1]) + \begin{bmatrix} 0, s1[i] == s2[j] \\ 1, \text{ иначе} \end{bmatrix} \end{pmatrix} \end{cases}$$

1.2 Расстояние Дамерау-Левенштейна

Вводится дополнительная операция: перестановка или транспозиция двух букв со штрафом 1. Если индексы позволяют и если две соседние буквы $s1[i] = s2[j - 1] \wedge s1[i - 1] = s2[j]$, то в минимум включается перестановка.

Пусть $s1$ и $s2$ — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Дамерау-Левенштейна) $d(s1, s2)$ можно подсчитать по следующей рекуррентной

формуле:

$$|x| = \begin{cases} 0, \text{ если } i = 0, j = 0; \\ i, \text{ если } i > 0, j = 0; \\ j, \text{ если } i = 0, j > 0; \\ \min \left(\begin{array}{l} D(s1[1 \dots i], s2[1 \dots j - 1]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j - 1]) + \begin{bmatrix} 0, s1[i] == s2[j] \\ 1, \text{ иначе} \end{bmatrix} \\ D(s1[0 \dots i - 2], s2[0 \dots j - 2]) + 1 \end{array} \right), \\ \text{если } i > 1, j > 1, s1[i - 1] = s2[j - 2], s1[i - 2] = s2[j - 1] \\ \min \left(\begin{array}{l} D(s1[1 \dots i], s2[1 \dots j - 1]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j]) + 1 \\ D(s1[1 \dots i - 1], s2[1 \dots j - 1]) + \begin{bmatrix} 0, s1[i] == s2[j] \\ 1, \text{ иначе} \end{bmatrix} \end{array} \right), \text{ иначе} \end{cases}$$

1.3 Применение

Расстояние Левенштейна и его обобщения активно применяются:

- при автозамене
- в поисковых строках
- "возможно вы имели ввиду"
- В биоинформатике (кодируем молекулы буквами)

2 Конструкторский раздел

2.1 Разработка алгоритмов

В разделе представлены схемы следующих алгоритмов вычисления расстояния:

- Левенштейна обычным способом (матричным) (рисунки 2.1-2.2)
- Левенштейна рекурсивным способом (рисунки 2.3-2.4)
- Левенштейна рекурсивным способом с кэшированием (рисунки 2.5-2.6)
- Дамерау-Левенштейна обычным способом (матричным) (рисунки 2.7-2.8)

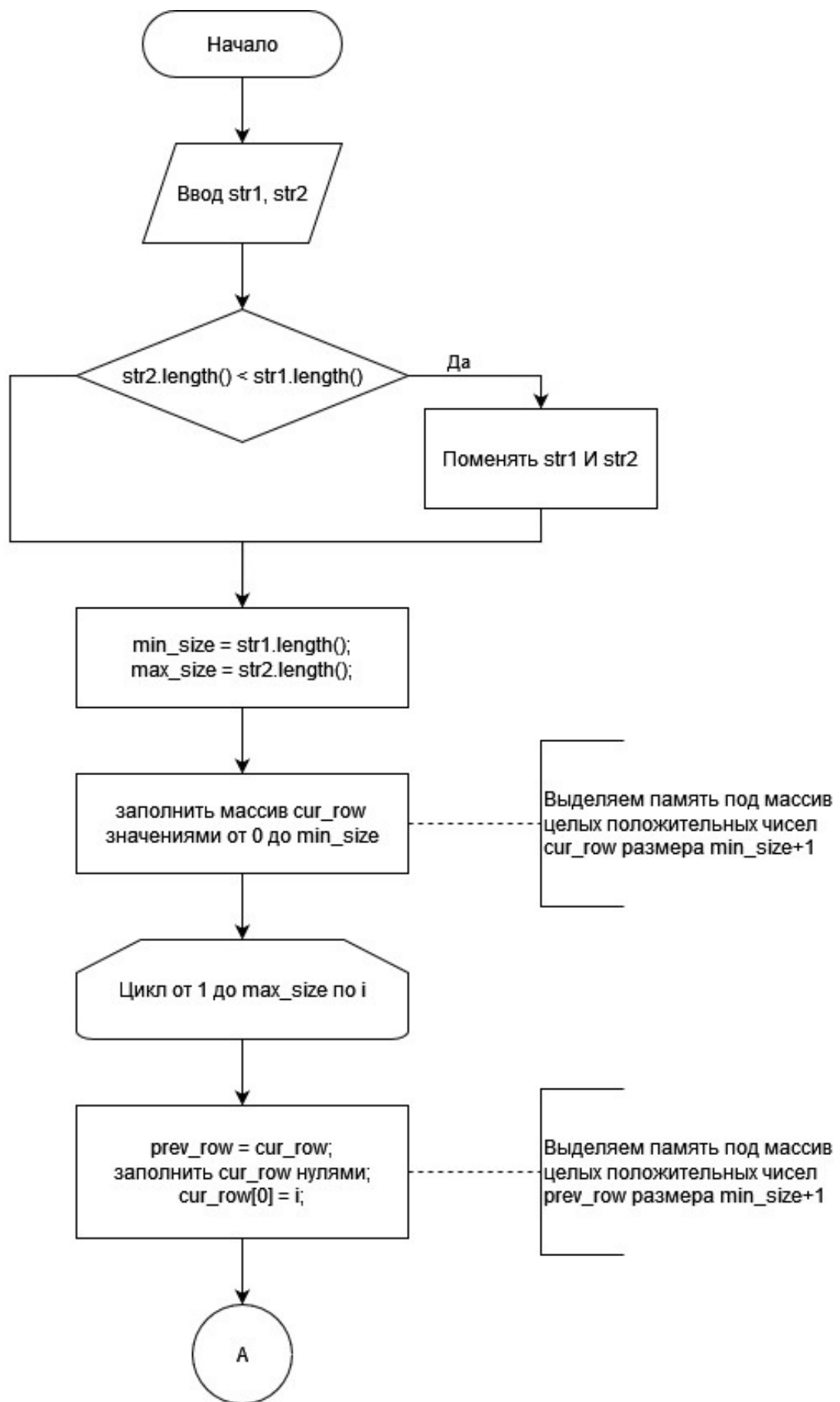


Рис. 2.1: Схема алгоритма Левенштейна часть 1

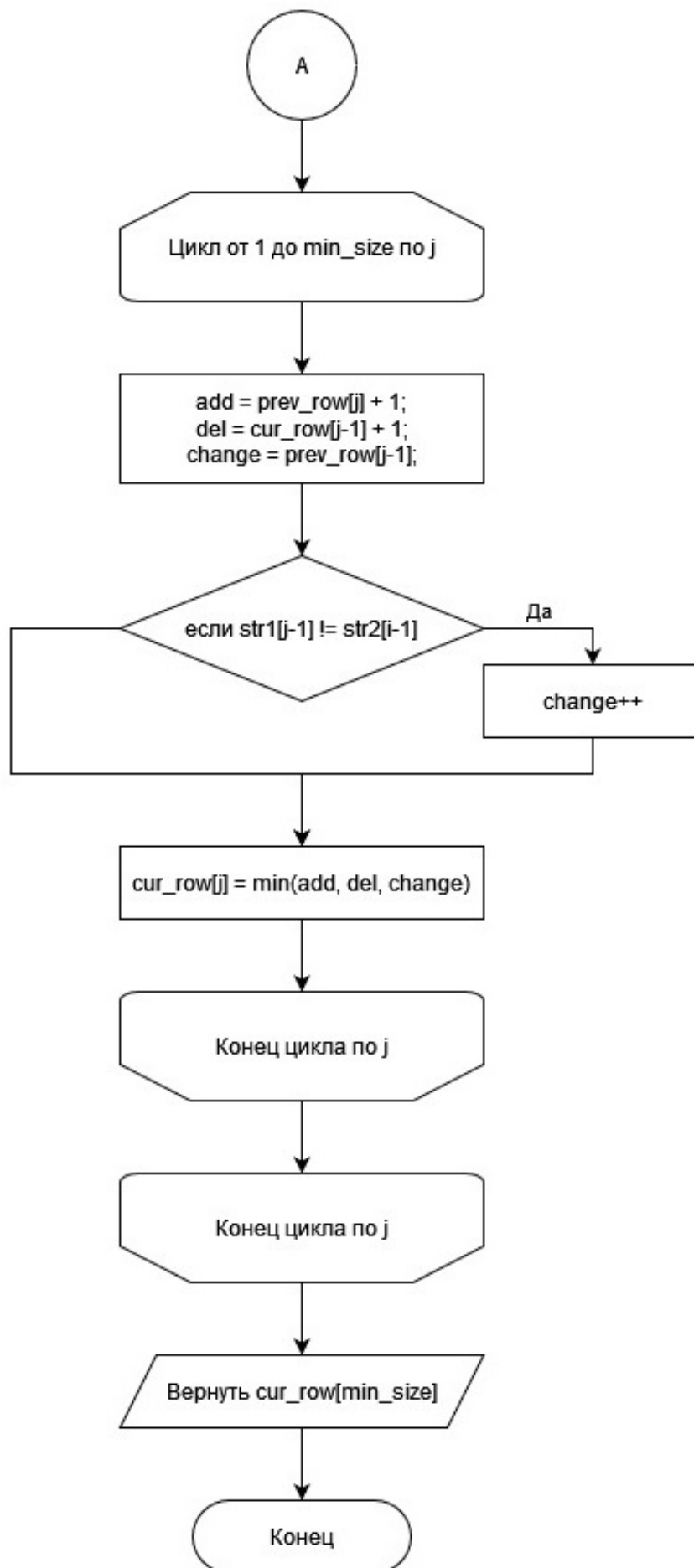


Рис. 2.2: Схема алгоритма Левенштейна часть 2

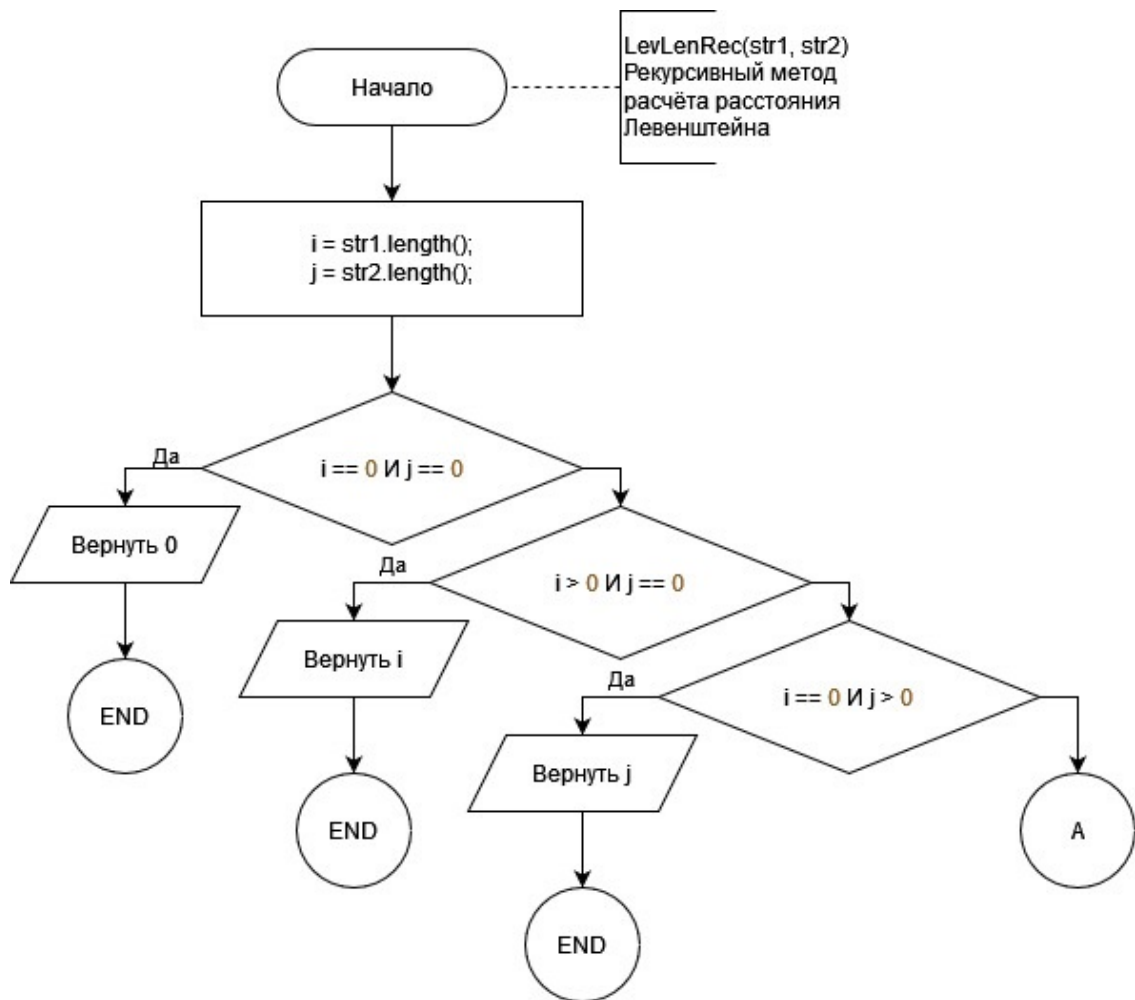


Рис. 2.3: Схема алгоритма Левенштейна с рекурсией часть 1

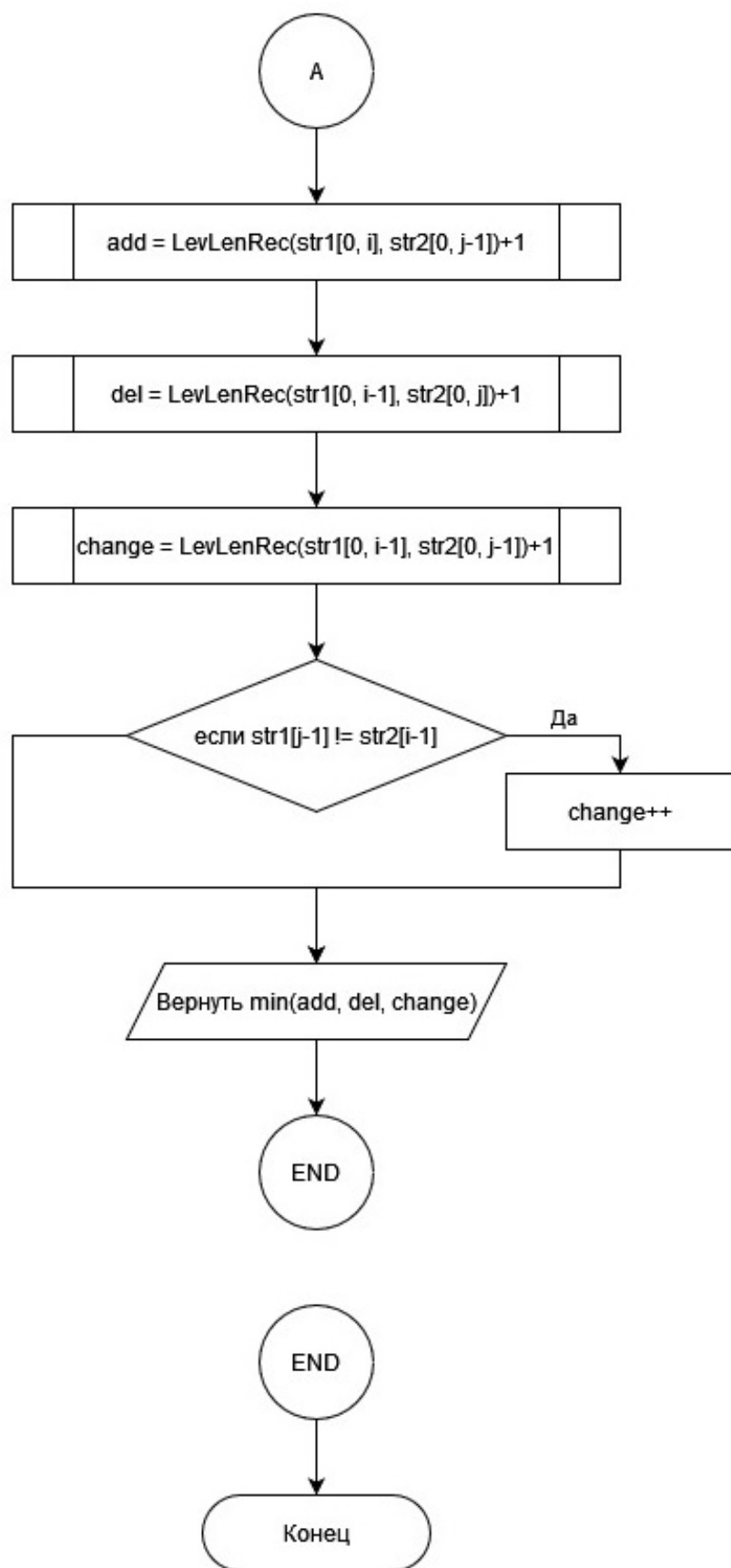


Рис. 2.4: Схема алгоритма Левенштейна с рекурсией часть 2

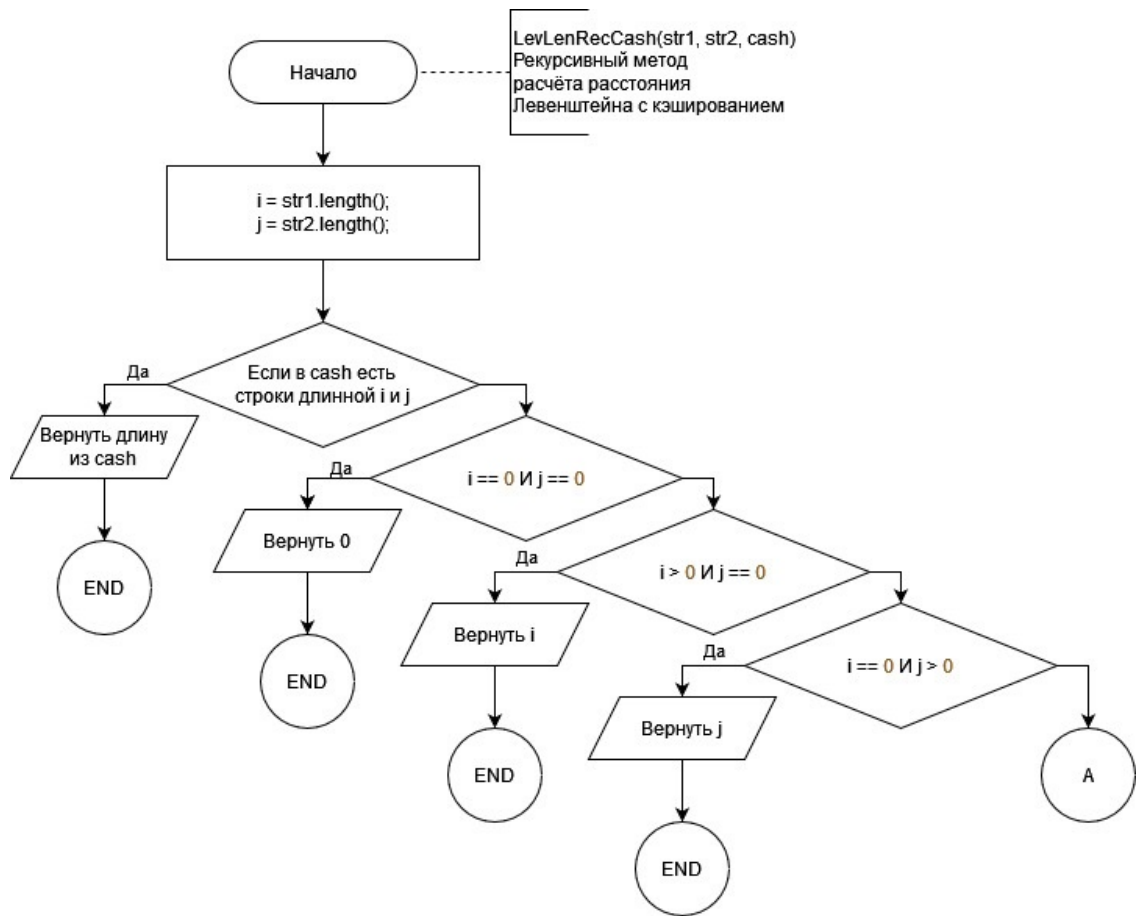


Рис. 2.5: Схема алгоритма Левенштейна с рекурсией и кэшированием часть 1

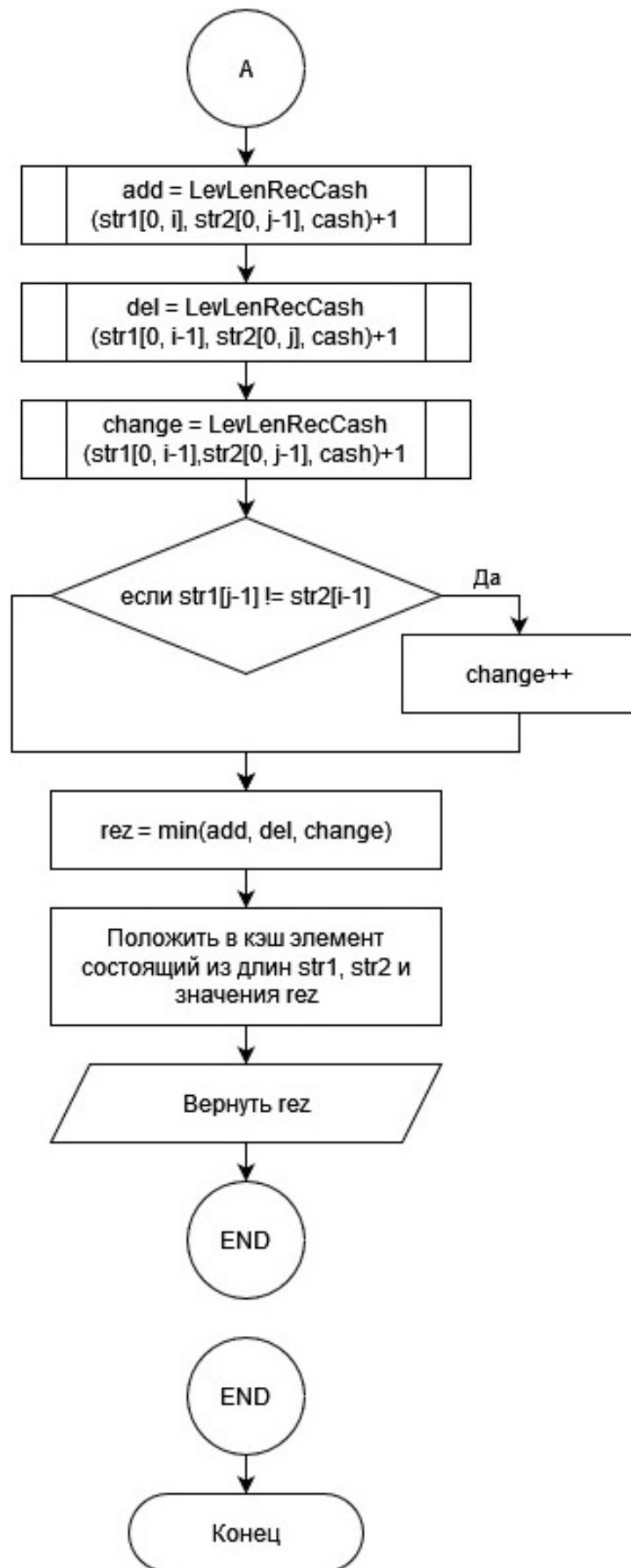


Рис. 2.6: Схема алгоритма Левенштейна с рекурсией и кэшированием часть 2

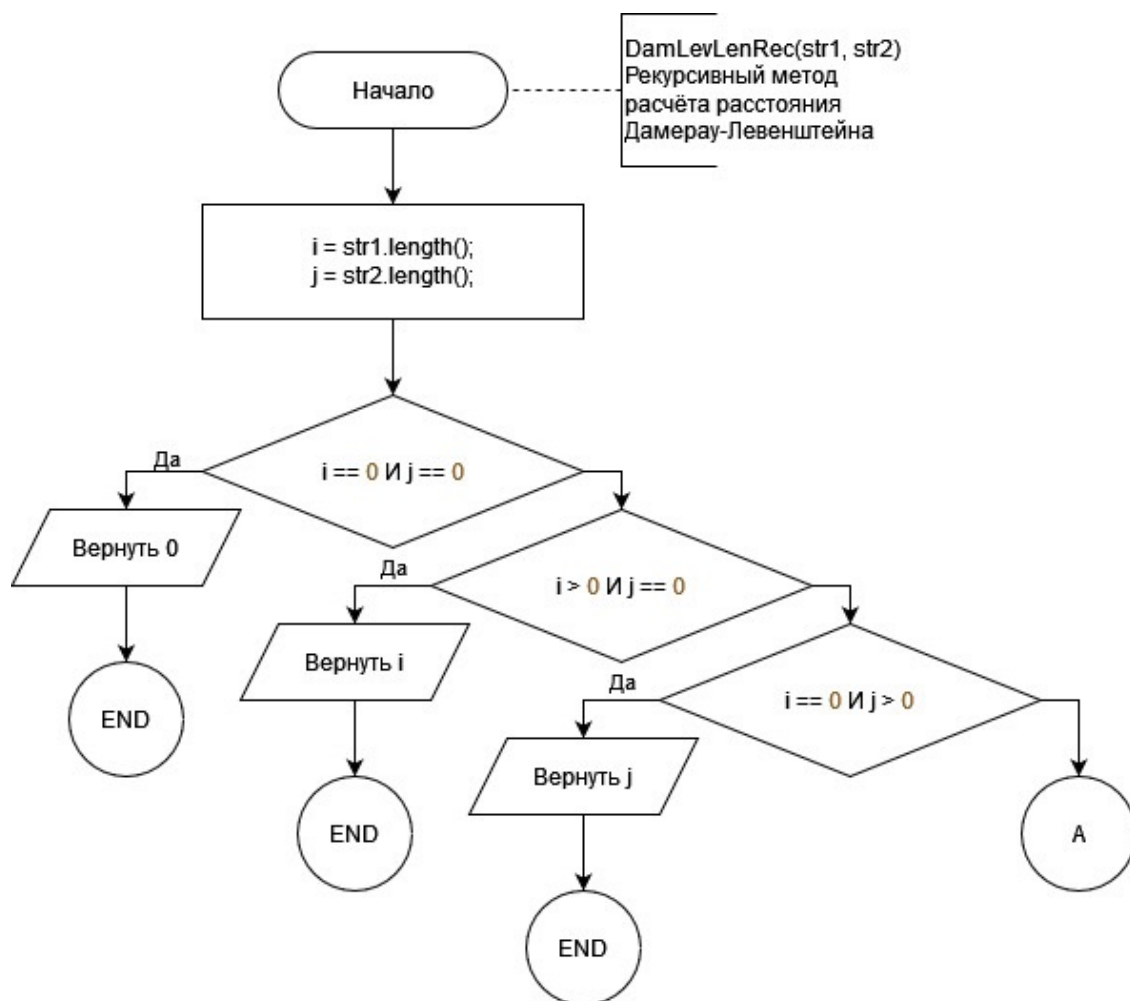


Рис. 2.7: Схема алгоритма Дамерау-Левенштейна с рекурсией часть 1

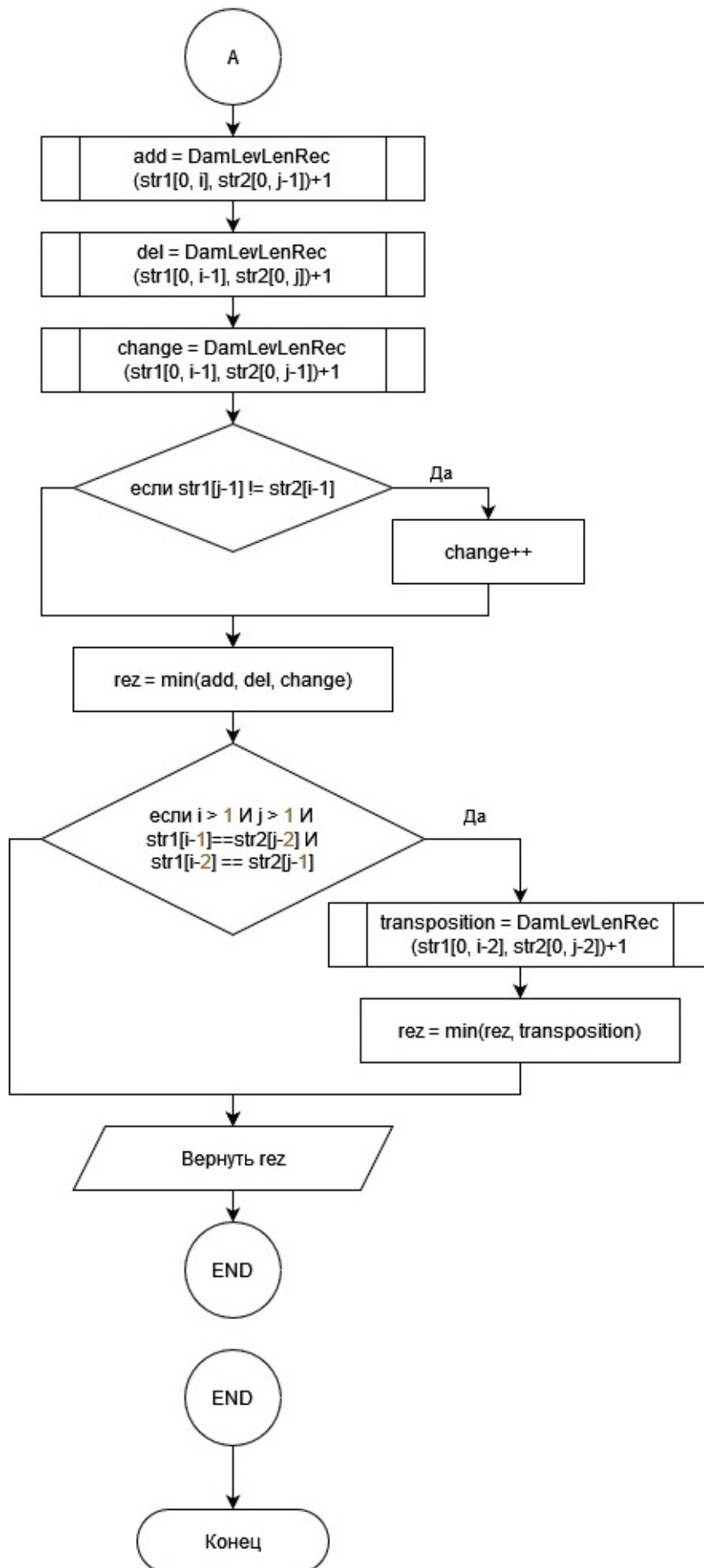


Рис. 2.8: Схема алгоритма Дамерау-Левенштейна с рекурсией часть 2

3 Технологический раздел

3.1 Выбор инструментов

Язык программирования (с++), среда разработки (QtCreator), библиотеки (windows.h?), valgrind

(как осуществляются замеры скорости и памяти?)

3.2 Реализация алгоритмов

На листингах , представлены реализации алгоритмов...

Листинг 3.1: Реализация алгоритма Левенштейна обычным способом

```
1 size_t LevLen(string str1, string str2)
2 {
3     if (str2.length() < str1.length())
4     {
5         return LevLen(str2, str1);
6     }
7
8     const size_t min_size = str1.length();
9     const size_t max_size = str2.length();
10
11     vector<size_t> cur_row(min_size+1);
12     for (size_t i = 1; i <= min_size; i++)
13     {
14         cur_row[i] = i;
15     }
16
17     for (size_t i = 1; i <= max_size; i++)
18     {
19         vector<size_t> prev_row = cur_row;
20         cur_row = vector<size_t>(min_size+1);
21         cur_row[0] = i;
22         for (size_t j = 1; j <= min_size; j++)
23         {
```



```

24         size_t add = prev_row[j] + 1;
25         size_t del = cur_row[j-1] + 1;
26         size_t change = prev_row[j-1];
27         if (str1[j-1] != str2[i-1])
28             change++;
29         cur_row[j] = min(add, min(del, change));
30     }
31 }
32
33 return cur_row[min_size];
34 }

```

Листинг 3.2: Реализация алгоритма Левенштейна рекурсивным способом

```

1 size_t LevLenRec(string str1, string str2)
2 {
3     size_t i = str1.length();
4     size_t j = str2.length();
5     if (i == 0 && j == 0)
6     {
7         return 0;
8     }
9     else if (i > 0 && j == 0)
10    {
11        return i;
12    }
13    else if (i == 0 && j > 0)
14    {
15        return j;
16    }
17    else
18    {
19        size_t add = LevLenRec(str1.substr(0, i),
20                               str2.substr(0, j-1)) + 1;
21        size_t del = LevLenRec(str1.substr(0, i-1),
22                               str2.substr(0, j)) + 1;
23        size_t change = LevLenRec(str1.substr(0, i-1),
24                                   str2.substr(0, j-1));
25        if (str1[i-1] != str2[j-1])
26            change++;
27        return min(add, min(del, change));
28    }
29 }

```

Листинг 3.3: Реализация алгоритма Левенштейна с рекурсией и кэшированием

```

1 bool checkCash(string str1, string str2, const cash_t &cash, size_t&
    rez)

```

```

2 {
3     for (auto &el : cash)
4     {
5         if (el.first.first == str1 && el.first.second == str2)
6         {
7             rez = el.second;
8             return true;
9         }
10    }
11    return false;
12 }
13
14 size_t LevLenRecCash(string str1, string str2, cash_t &cash)
15 {
16     size_t len = INT_MAX;
17     if (checkCash(str1, str2, cash, len))
18     {
19         return len;
20     }
21     //cout << str1 << " " << str2 << endl;
22     size_t i = str1.length();
23     size_t j = str2.length();
24     if (i == 0 && j == 0)
25     {
26         return 0;
27     }
28     else if (i > 0 && j == 0)
29     {
30         return i;
31     }
32     else if (i == 0 && j > 0)
33     {
34         return j;
35     }
36     else
37     {
38         size_t add = LevLenRecCash(str1.substr(0, i),
39                                     str2.substr(0, j-1), cash) + 1;
40         size_t del = LevLenRecCash(str1.substr(0, i-1),
41                                     str2.substr(0, j), cash) + 1;
42         size_t change = LevLenRecCash(str1.substr(0, i-1),
43                                       str2.substr(0, j-1), cash);
44         if (str1[i-1] != str2[j-1])
45             change++;
46
47         size_t rez = min(add, min(del, change));
48         cash.push_back(make_pair(make_pair(str1, str2), rez));
49         return rez;
50     }
51 }

```

```

47     }
48 }
49
50 size_t LevLenRecCash(string str1, string str2)
51 {
52     cash_t cash = cash_t();
53     return LevLenRecCash(str1, str2, cash);
54 }

```

Листинг 3.4: Реализация алгоритма Дамеру-Левенштейна рекурсивным способом

```

1 size_t DamLevLenRec(string str1, string str2)
2 {
3     size_t i = str1.length();
4     size_t j = str2.length();
5     if (i == 0 && j == 0)
6     {
7         return 0;
8     }
9     else if (i > 0 && j == 0)
10    {
11        return i;
12    }
13    else if (i == 0 && j > 0)
14    {
15        return j;
16    }
17    else
18    {
19        size_t add = DamLevLenRec(str1.substr(0, i),
20                                str2.substr(0, j-1)) + 1;
21        size_t del = DamLevLenRec(str1.substr(0, i-1),
22                                str2.substr(0, j)) + 1;
23        size_t change = DamLevLenRec(str1.substr(0, i-1),
24                                    str2.substr(0, j-1));
25        if (str1[i-1] != str2[j-1])
26            change++;
27        size_t rez = min(add, min(del, change));
28        if (i > 1 && j > 1 && str1[i-1]==str2[j-2] && str1[i-2]
29            == str2[j-1])
30        {
31            size_t transposition =
32                DamLevLenRec(str1.substr(0, i-2),
33                            str2.substr(0, j-2)) + 1;
34            rez = min(rez, transposition);
35        }
36        return rez;
37    }
38 }

```

$$\left. \begin{array}{l} 31 \\ 32 \end{array} \right\}$$

4 Исследовательский раздел

4.1 Тесты

Подготовлены тесты... тесты пройдены...

4.2 Сравнительный анализ алгоритмов

Графики скорости

Сравнение по памяти

4.3 Вывод

Берём свой готовый продукт и анализируем, приводим графики в виде рисунков, делаем выводы из экспериментов по отдельности и общий, итоговый (глобальный) вывод

Заключение

цель достигнута, задачи достигнуты: список задач.

Цель – получить навык динамического программирования

Задачи (План) – изучить расстояния Левенштайна, Д-Л. Разработать алгоритм поиска расстояния для (далее 4 алгоритма). Реализовать алгоритмы. Провести сравнительный анализ процессорного времени выполнения реализации алгоритма. Провести анализ пикового значения затрачиваемой памяти в программе

В конце (личный анализ реализаций алгоритмов (сравнивать в одинаковых расстояниях))

Слово “Лучше” лучше не использовать (сослаться на источник или обосновать)