

Московский государственный технический университет
имени Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Ю.Е. Гапанюк

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по дисциплине
«Архитектура автоматизированных систем
обработки информации и управления»
Профиль: «Архитектура программных
систем»**

**Домашнее задание 1
Реализация алгоритма поиска с опечатками**

Москва – 2026

Оглавление

1	УСЛОВИЯ ДОМАШНЕГО ЗАДАНИЯ №1.....	3
2	РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА С ОПЕЧАТКАМИ	3
2.1	Расстояние ДАМЕРАУ-ЛЕВЕНШТЕЙНА.....	3
2.2	Вычисление расстояния ДАМЕРАУ-ЛЕВЕНШТЕЙНА.....	5
2.3	ПРИМЕР ВЫЧИСЛЕНИЯ РАССТОЯНИЯ ДАМЕРАУ-ЛЕВЕНШТЕЙНА.....	8
2.4	Алгоритм ВАГНЕРА-ФИШЕРА ВЫЧИСЛЕНИЯ РАССТОЯНИЯ ДАМЕРАУ-ЛЕВЕНШТЕЙНА	9
3	КОНТРОЛЬНЫЕ ВОПРОСЫ	12
4	ИСТОЧНИКИ	12

1 Условия домашнего задания №1

- Домашнее задание реализуется в виде консольного приложения на языке программирования C#. Рекомендуется использовать операторы верхнего уровня.
- Пользователь вводит две строки, после чего программа выводит значение расстояния Левенштейна между этими строками.
- Указанные выше действия реализуются в виде вечного цикла. Для выхода из цикла в качестве первой строки вводится значение «exit».
- Для сдачи домашнего задания необходимо продемонстрировать работоспособность программы, ответить на вопросы и предоставить отчет, содержащий текст программы и результаты ее выполнения.

2 Реализация алгоритма поиска с опечатками

Практически любая крупная информационная система содержит такие данные как фамилии, имена, отчества, географические названия, которые могут быть набраны с опечатками при вводе данных или при вводе ключевого слова для поиска.

Устранение опечаток при поиске представляет собой нетривиальную задачу. Для ее решения предлагались различные методы, например разбиение слов на два или три символа с их комбинаторным перебором. Но все они оказались мало эффективны.

2.1 Расстояние Дамерау-Левенштейна

Эффективный способ решения данной задачи предложил отечественный исследователь – Владимир Иосифович Левенштейн.

Вычисление расстояния Левенштейна (редакционного расстояния) основано на понятии редакционного предписания. Редакционным предписанием называется

последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Обычно действия обозначаются так:

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

Пример редакционного предписания представлен на рис. 25:

Действие	M	M	I	R	M	M	R	D
Исходное слово	П	Р		И	М	Е	Р	Ы
Результат	П	Р	Е	Д	М	Е	Т	

Рис. 1. Пример редакционного предписания.

Для преобразования слова «ПРИМЕРЫ» в слово «ПРЕДМЕТ» необходимо выполнить 4 действия (добавление, удаление и две замены).

Расстоянием Левенштейна называется минимальное количество действий, необходимых для преобразования одного слова в другое. В этом примере расстояние Левенштейна равно 4.

Преобразовать одно слово в другое можно различными способами, количество действий также может быть разным. Поэтому при вычислении расстояния и выбирается минимальное количество действий.

Если поиск производится в тексте, который набирается на клавиатуре, то вместо расстояния Левенштейна используют усовершенствованное расстояние Дамерау-Левенштейна. Фредерик Дамерау – один из американских пионеров-исследователей в области обработки текстов на естественном языке. Исследования Дамерау показали, что наиболее частой ошибкой при наборе слова является перестановка двух соседних букв – транспозиция, которая обозначается как T (transposition). Данное действие принято называть поправкой Дамерау к расстоянию Левенштейна.

В случае одной транспозиции расстояние Левенштейна равняется двум (две замены символа). При использовании поправки Дамерау транспозиция считается как единичное расстояние (рис. 26).

Действие (расстояние Дамерау-Левенштейна)	M	M	T		M	M	M
Действие (расстояние Левенштейна)	M	M	R	R	M	M	M
Исходное слово	P	R	I	M	E	P	Y
Результат	P	R	M	I	E	P	Y

Рис. 2. Транспозиция при вычислении расстояний Левенштейна и Дамерау-Левенштейна.

При использовании расстояния Дамерау-Левенштейна за единичное расстояние принимаются следующие действия:

- I (insert) — добавление символа.
- D (delete) — удаление символа.
- R (replace) — замена символа.
- T (transposition) — перестановка двух соседних символов.

2.2 Вычисление расстояния Дамерау-Левенштейна

Пусть заданы строка S_1 длиной M символов и строка S_2 длиной N символов.

Предполагается, что строка S_1 длиннее или равна по длине строке S_2 ($M \geq N$).

Для вычисления расстояния Левенштейна необходимо рекуррентно вычислить значения элементов матрицы $D[i, j]$ размером $M \times N$. Значение в правом нижнем элементе $D[M, N]$ и будет являться расстоянием Левенштейна.

Расстояние Левенштейна $ed(S_1, S_2)$ вычисляется на основе формулы 1:

$$ed(S_1, S_2) = D[M, N], \text{ где} \quad (1)$$

$$D[i, j] = \begin{cases} 0 & ; i = 0, j = 0 \\ i & ; i > 0, j = 0 \\ j & ; i = 0, j > 0 \\ \min (& \\ D[i - 1, j] + 1, (\text{утверждение 1}) & ; i > 0, j > 0 \\ D[i, j - 1] + 1, (\text{утверждение 2}) & (\text{случай 4}) \\ D[i - 1, j - 1] + m(S_1[i], S_2[j]) (\text{утверждение 3}) & \end{cases}$$

Выражение $m(\text{символ1}, \text{символ2})$ равняется нулю, если символы совпадают и единице, если символы не совпадают (проверяется равенство i -го символа из первой строки и j -го символа из второй строки).

Рассмотрим формулу более подробно в соответствии с [4], где $D(i, j)$ – расстояние между префиксами строк: первыми i символами строки S_1 и первыми j символами строки S_2 .

Редакционное расстояние между двумя пустыми строками равно нулю (случай 1).

Для получения пустой строки из строки длиной i , нужно совершить i операций удаления (случай 2).

Чтобы получить строку длиной j из пустой строки, нужно провести j операций вставки (случай 3).

В случае 4 обе строки непусты. В оптимальной последовательности операций, операции можно произвольно менять местами. Рассмотрим две последовательные операции:

- Две замены одного и того же символа – неоптимально (если мы заменили x на y , потом y на z , выгоднее было сразу заменить x на z).
- Две замены разных символов можно менять местами.
- Два удаления или два добавления можно менять местами.

- Добавление символа с его последующим удалением – неоптимально (можно отменить оба действия).
- Удаление и добавление разных символов можно менять местами.
- Добавление символа с его последующей заменой – неоптимально (излишняя замена).
- Добавление символа и замена другого символа меняются местами.
- Замена символа с его последующим удалением – не оптимально (излишняя замена).
- Удаление символа и замена другого символа меняются местами.

Пускай S_1 заканчивается на символ «а», S_2 заканчивается на символ «б».

Тогда выполняется одно из следующих утверждений (соответствующих утверждениям 1-3 в формуле 1):

1. Символ «а», на который кончается S_1 , в какой-то момент был удален. Сделаем это удаление первой операцией. Тогда мы удалили символ «а», после чего превратили первые $i-1$ символов S_1 в S_2 (на это потребовалось $D[i-1, j]$ операций), значит, всего потребовалось $D[i-1, j] + 1$ операций.
2. Символ «б», на который кончается S_2 , в какой-то момент был добавлен. Сделаем это добавление последней операцией. Мы превратили S_1 в первые $j-1$ символов S_2 , после чего добавили «б». Аналогично предыдущему случаю, потребовалось $D[i, j-1] + 1$ операций.
3. Оба предыдущих утверждения неверны. Если мы добавляли символы справа от «а», то чтобы сделать последним символом «б», мы должны были или в какой-то момент добавить его (но тогда утверждение 2 было бы верно), либо заменить на него один из этих добавленных символов (что тоже невозможно, потому что добавление символа с его последующей заменой не оптимально). Значит, символов справа от финального «а» мы не добавляли. Самого финального «а» мы не удаляли, поскольку утверждение 1 неверно. Значит, единственный способ

изменения последнего символа – его замена. Заменять его 2 или больше раз неоптимально. Следовательно, возможны два варианта:

3.1. Если «а» совпадает с «б», мы последний символ не меняли. Поскольку мы его также не стирали и не приписывали ничего справа от него, он не влиял на наши действия, и, значит, мы выполнили $D[i - 1, j - 1]$ операций, $m(S_1[i], S_2[j]) = 0$.

3.2. Если «а» не совпадает с «б», мы последний символ меняли один раз. Сделаем эту замену первой. В дальнейшем, аналогично предыдущему случаю, мы должны выполнить $D[i - 1, j - 1]$ операций, значит, всего потребуется $D[i - 1, j - 1] + 1$ операций, $m(S_1[i], S_2[j]) = 1$.

Поскольку расстояние Левенштейна определяет минимальное количество действий, то берется минимум из утверждений 1-3.

Если вычисляется расстояние Дамерау-Левенштейна (транспозиция считается единичным расстоянием), то на основе вычисленных элементов матрицы $D[i, j]$ вычисляются элементы матрицы $DT[i, j]$ (формула 2).

$$DT[i, j] = \begin{cases} \min (& ; \text{Если выполняются} \\ & D[i, j] (\text{расстояние Левенштейна}) & \text{условия } i > 1, j > 1, \\ & D[i - 2, j - 2] + m(S_1[i], S_2[j]) & S_1[i] = S_2[j - 1], \\) & (\text{транспозиция}) & S_1[i - 1] = S_2[j] \\ D[i, j] & ; \text{Если условия} \\ & \text{не выполняются} \end{cases} \quad (2)$$

Значение в правом нижнем элементе $DT[M, N]$ и будет являться расстоянием Дамерау-Левенштейна.

2.3 Пример вычисления расстояния Дамерау-Левенштейна

Допустим, что при вводе в справочник информационной системы оператор допустил ряд ошибок при вводе фамилии ИВАНОВ:

1. Добавление (I) буквы Н в середине слова – ИВАННОВ.
2. Удаление (D) буквы И в начале слова – ВАННОВ.
3. Замена (R) буквы В на Б – БАННОВ.

4. Перестановка (Т) букв В и О – БАННВО.

Так как каждое действие увеличивает расстояние на 1, то итоговое расстояние Дамерау-Левенштейна равно 4.

Пример вычисление расстояния с помощью матрицы $DT[i,j]$ приведен на рис. 27.

	И	В	А	Н	О	В
Б	0	1	2	3	4	5
А	1	1	2	3	4	5
Н	2	2	2	2	3	4
Н	3	3	3	3	2	3
В	4	4	4	4	3	3
О	5	5	4	5	4	4
	6	6	5	5	5	4

Рис. 3. Пример вычисления расстояния Дамерау-Левенштейна.

В правой нижней ячейке матрицы находится вычисленное расстояние – число 4.

Таким образом, использование расстояния Дамерау-Левенштейна позволяет находить слово даже при большом количестве опечаток.

2.4 Алгоритм Вагнера-Фишера вычисления расстояния Дамерау-Левенштейна

Существует несколько алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна.

Алгоритм Вагнера-Фишера – это наиболее простой алгоритм, который вычисляет расстояние Дамерау-Левенштейна на основе определения (формула 2).

Недостатками алгоритма являются невысокая скорость работы и большие затраты памяти.

Временная сложность алгоритма $O(M \times N)$, где M и N – длины строк. То есть время выполнения алгоритма прямо пропорционально количеству ячеек в матрице DT .

Рассмотрим реализацию алгоритма на языке C# в соответствии с фрагментами примера 16:

```

/// <summary>
/// Вычисление расстояния Дамерау-Левенштейна
/// </summary>
public static int Distance(string str1Param, string str2Param)
{
    if ((str1Param == null) || (str2Param == null)) return -1;

    int str1Len = str1Param.Length;
    int str2Len = str2Param.Length;

    //Если хотя бы одна строка пустая,
    //возвращается длина другой строки
    if ((str1Len == 0) && (str2Len == 0)) return 0;
    if (str1Len == 0) return str2Len;
    if (str2Len == 0) return str1Len;

    //Приведение строк к верхнему регистру
    string str1 = str1Param.ToUpper();
    string str2 = str2Param.ToUpper();

    //Объявление матрицы
    int[,] matrix = new int[str1Len + 1, str2Len + 1];

    //Инициализация нулевой строки и нулевого столбца матрицы
    for (int i = 0; i <= str1Len; i++) matrix[i, 0] = i;
    for (int j = 0; j <= str2Len; j++) matrix[0, j] = j;

    //Вычисление расстояния Дамерау-Левенштейна
    for (int i = 1; i <= str1Len; i++)
    {
        for (int j = 1; j <= str2Len; j++)
        {
            //Эквивалентность символов, переменная symbEqual
            //соответствует m(s1[i],s2[j])
            int symbEqual =
                (str1.Substring(i - 1, 1) ==
                 str2.Substring(j - 1, 1)) ? 0 : 1;

            int ins = matrix[i, j - 1] + 1; //Добавление
            int del = matrix[i - 1, j] + 1; //Удаление
            int subst = matrix[i - 1, j - 1] + symbEqual; //Замена

            //Элемент матрицы вычисляется
            //как минимальный из трех случаев
            matrix[i, j] = Math.Min(Math.Min(ins, del), subst);

            //Дополнение Дамерау по перестановке соседних символов
            if ((i > 1) && (j > 1) &&

```

```

        (str1.Substring(i - 1, 1) == str2.Substring(j - 2, 1)) &&
        (str1.Substring(i - 2, 1) == str2.Substring(j - 1, 1)))
    {
        matrix[i, j] = Math.Min(matrix[i, j],
                                matrix[i - 2, j - 2] + symbEqual);
    }
}
//Возвращается нижний правый элемент матрицы
return matrix[str1Len, str2Len];
}

```

Функция Distance принимает на вход две строки, вычисляет значения матрицы DT и возвращает значение нижнего правого элемента матрицы.

Вспомогательная функция WriteDistance выводит в консоль результаты вычисления расстояния Дамерау-Левенштейна:

```

/// <summary>
/// Вывод расстояния Дамерау-Левенштейна в консоль
/// </summary>
public static void WriteDistance(string str1Param, string str2Param)
{
    int d = Distance(str1Param, str2Param);
    Console.WriteLine("'" + str1Param + "','" +
                      str2Param + "' -> " + d.ToString());
}

```

Пример вычисления расстояния:

```

Console.WriteLine("Добавление одного символа в начало, середину и конец
строки");
EditDistance.WriteDistance("пример", "1пример");
EditDistance.WriteDistance("пример", "при1мер");
EditDistance.WriteDistance("пример", "пример1");

Console.WriteLine("Добавление двух символов в начало, середину и конец
строки");
EditDistance.WriteDistance("пример", "12пример");
EditDistance.WriteDistance("пример", "при12мер");
EditDistance.WriteDistance("пример", "пример12");

Console.WriteLine("Добавление трех символов");
EditDistance.WriteDistance("пример", "1при2мер3");

Console.WriteLine("Транспозиция");
EditDistance.WriteDistance("прИМер", "прМИер");

Console.WriteLine("Рассмотренный ранее пример");
EditDistance.WriteDistance("ИВАНОВ", "БАННВО");

```

Результаты вывода в консоль:

Добавление одного символа в начало, середину и конец строки

'пример', '1пример' -> 1

'пример', 'при1мер' -> 1

'пример', 'пример1' -> 1

Добавление двух символов в начало, середину и конец строки

'пример', '12пример' -> 2

'пример', 'при12мер' -> 2

'пример', 'пример12' -> 2

Добавление трех символов

'пример', '1при2мер3' -> 3

Транспозиция

'приМер', 'прМИер' -> 1

Рассмотренный ранее пример

'ИВАНОВ', 'БАННВО' -> 4

3 Контрольные вопросы

1. Что такое расстояние Левенштейна?
2. Что такое расстояние Дамерау-Левенштейна?
3. Объясните алгоритм Вагнера-Фишера для вычисления расстояния Дамерау-Левенштейна.

4 Источники

1. Нейгел К., Ивьеен Б., Глинн Д., Уотсон К. C# 5.0 и платформа .NET 4.5 для профессионалов. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2014. – 1440 с.
2. Шилдт Г. C# 4.0: полное руководство. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2013. – 1056 с.
3. Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов. – СПб.: Питер, 2015. – 432 с.
4. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР, 1965. 163.4:845-848.