Министерство науки и высшего образования   
Российской Федерации

Ивановский государственный университет

Факультет математики и компьютерных наук

«Рекомендовать к защите»

Заведующий кафедрой прикладной   
математики и компьютерных наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Соколов Е. В.

протокол заседания кафедры № \_\_\_\_

от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Выпускная квалификационная работа   
(магистерская диссертация)

Алгоритмы нахождения расстояния между строками

|  |  |
| --- | --- |
| Направление подготовки: | 01.04.01 Математика |
| Направленность образовательной программы: | Фундаментальная математика |
| Выпускную квалификационную работу выполнил: | студент 2 курса очной формы обучения  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Исай Валерия Андреевна |
| Руководитель выпускной квалификационной работы: | доцент кафедры прикладной математики и компью­терных наук, кандидат физико-математических наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хашин Сергей Иванович |

Иваново, 2020

Оглавление

[Введение 3](#_Toc41482620)

[Алгоритмы нечеткого поиска 4](#_Toc41482621)

[Расстояние Левенштейна 5](#_Toc41482622)

[Редакционное предписание 5](#_Toc41482623)

[Описание работы программы 10](#_Toc41482624)

[Результаты работы программы 20](#_Toc41482625)

[Заключение 27](#_Toc41482626)

[Список литературы 33](#_Toc41482627)

# Введение

Целью данной работы является изучение алгоритмов нахождения расстояния между строками.

Актуальность работы состоит в том, что имеется несколько различных методов нечёткого сравнения строк. Их сравнительная скорость работы более-менее ясна, но качество сравнения на больших объемах реальных данных не совсем понятно.

Практическая задача выглядит так.

Есть файл со списком трансформаторных подстанций по стране (двадцатилетней давности). И имеется другой файл со списком линий электропередач, каждая из которых идет от одной подстанции до другой. В названиях есть ошибки и разночтения, поэтому строки приходится сравнивать не точно, а приближённо. Целью настоящей работы является написать программу, которая:

1. для каждой строки из первого файла находит пару строк из второго файла, наиболее подходящих;
2. расхождения (если есть) при сопоставлении этих строк;
3. время, за которое будет находиться расстояние Левенштейна между строками длины 10, 30,100, 300, 1000, 3000, 10000 и 30000 символов.

Таким образом, постановка задачи и цели исследования следующие:

* Реализация алгоритмов.
* Сравнение скорости работы на больших объемах данных.
* Сравнение качества результатов, полученных разными алгоритмами.

Для решения основной задачи было необходимо найти и изучить соответствующую литературу. Была выбрана среда разработки Visual Studio, язык С++, программа сделана под Windows. Далее, нужно было определить основные этапы работы, а так же сформулировать выводы по проделанной работе.

# Алгоритмы нечеткого поиска

Нечеткий поиск - это поиск информации, при котором выполняется сопоставление информации заданному образцу поиска или близкому к этому образцу значению.

Алгоритмы нечеткого поиска (также известного как поиск по сходству или fuzzy string search) являются основой систем проверки орфографии и полноценных поисковых систем, например, Google или Yandex. Такие алгоритмы используются для функций по типу «Возможно, вы имели в виду …» в тех же поисковых системах.  
Расстояние Левенштейна используется для сравнения двух слов или двух строк, чтобы определить их схожесть. Существует схожая задача - в заданной строке искать вхождение слов, словосочетаний и формул, похожих на образец.

Нечеткий поиск является весьма полезной функцией любой поисковой системы. Вместе с тем, его эффективная реализация намного сложнее, чем реализация простого поиска по точному совпадению.

Задачу нечеткого поиска можно сформулировать следующим образом:  
«По заданному слову найти в тексте или словаре размера n все слова, совпадающие с этим словом (или начинающиеся с этого слова) с учетом k возможных различий».  
Например, при запросе «Машина» с учетом двух возможных ошибок, найти слова «Машинка», «Махина», «Малина», «Калина» и так далее.  
Алгоритмы нечеткого поиска характеризуются метрикой*.*

Метрика – это функция расстояния между двумя словами, которая позволяет оценить степень их сходства в данном контексте.

Строгое математическое определение метрики включает в себя необходимость соответствия условию неравенства треугольника (где *X* — множество слов, *p* — метрика):  
***p(x,y) p(x,z)+p(z,y), x,y,z X.***  
В большинстве случаев под метрикой подразумевается более общее понятие, не требующее выполнения такого условия, это понятие можно также назвать расстоянием.  
Наиболее известными метриками являются расстояние Левенштейна, Дамерау-Левенштейна и Хемминга. При этом расстояние Хемминга является метрикой лишь только на множестве слов одинаковой длины, что довольно сильно ограничивает область его применения. В данной работе рассматривается и используется расстояние Левенштейна.[5]

# Расстояние Левенштейна

Расстояние Левенштейна (*редакционное расстояние*, *дистанция редактирования*) — метрика, измеряющая разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (а именно вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены.   
Данная задача впервые была поставлена советским математиком Владимиром Левенштейном в 1965 году при изучении последовательностей{\displaystyle 0-1} и впоследствии более общую задачу для произвольного алфавита связали с его именем. Так же большой вклад в изучение данного вопроса внёс Дэн Гасфилд.  
Наиболее популярный алгоритм расчета - метод Вагнера-Фишера.[1,2,3,4]

## Редакционное предписание

*Редакционным предписанием* называется последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Обычно действия обозначаются так: **D** (англ. *delete*) — удалить, **I** (англ. insert) — вставить, **R** (англ. *replace*) — заменить, **M** (англ. *match*) — совпадение.

Например, для 2 строк «ДВИЖЕНИЕ» и «РЕШЕНИЕ» можно построить следующую таблицу преобразований:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **D** | **R** | **R** | **R** | **M** | **M** | **M** | **M** |
| ***Д*** | ***В*** | ***И*** | ***Ж*** | ***Е*** | ***Н*** | ***И*** | ***Е*** |
|  | ***Р*** | ***Е*** | ***Ш*** | ***Е*** | ***Н*** | ***И*** | ***Е*** |

Найти только расстояние Левенштейна — более простая задача, чем найти ещё и редакционное предписание. [1,2,3,4]

**Формула**

Здесь и далее считается, что элементы строк нумеруются с первого, как принято в математике, а не с нулевого, как принято во многих языках программирования.

Пусть {\displaystyle S\_{1}} SSSsssS1 и S2 {\displaystyle S\_{2}} — две строки (длиной {\displaystyle M}M и {\displaystyle N}N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) {\displaystyle {\rm {d}}(S\_{1},S\_{2})}d(S1, S2) можно подсчитать по следующей рекуррентной формуле:

d(S1, S2) – D(M,N), где

D(*i,j)* =

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

Здесь шаг по {\displaystyle i}i символизирует удаление (D) из первой строки, по {\displaystyle j}j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Очевидно, справедливы следующие утверждения:

d(S1,S2) max ( |S1|,|S2|)

d(S1,S2) ( |S1|- |S2|)

d(S1,S2) = 0 S1 = S2

**Пример работы алгоритма**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | а | р | е | с | т | а | н | т |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| д | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| а | 2 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 7 |
| г | 3 | 2 | 2 | 3 | 4 | 5 | 6 | 6 | 7 |
| е | 4 | 3 | 3 | 2 | 3 | 4 | 5 | 6 | 7 |
| с | 5 | 4 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |
| т | 6 | 5 | 5 | 4 | 3 | 2 | 3 | 4 | 5 |
| а | 7 | 6 | 6 | 5 | 4 | 3 | 2 | 3 | 4 |
| н | 8 | 7 | 7 | 6 | 5 | 4 | 3 | 2 | 3 |

[6]

**Доказательство**

Рассмотрим формулу более подробно. Очевидно, что редакционное расстояние между двумя пустыми строками равно нулю. Так же очевидно то, что чтобы получить пустую строку из строки длиной {\displaystyle i}i, нужно совершить {\displaystyle i}i операций удаления, а чтобы получить строку длиной {\displaystyle j}j из пустой, нужно произвести {\displaystyle j}j операций вставки.

Осталось рассмотреть нетривиальный случай, когда обе строки непусты.

Для начала заметим, что в оптимальной последовательности операций их можно произвольно менять местами. В самом деле, 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 стирания или две вставки можно менять местами
* Вставка символа с его последующим стиранием — неоптимально (можно их обе отменить)
* Стирание и вставку разных символов можно менять местами
* Вставка символа с его последующей заменой — неоптимально (излишняя замена)
* Вставка символа и замена другого символа меняются местами
* Замена символа с его последующим стиранием — неоптимально (излишняя замена)
* Стирание символа и замена другого символа меняются местами

Пусть S1{\displaystyle S\_{1}} кончается на символ «a», {\displaystyle S\_{2}} S2 кончается на символ «b». Есть три варианта:

1. Символ «а», на который кончается {\displaystyle S\_{1}} S1, в какой-то момент был стёрт. Сделаем это стирание первой операцией. Тогда мы стёрли символ «a», после чего превратили первые {\displaystyle i-1}i-1 символов {\displaystyle S\_{1}} S1 в {\displaystyle S\_{2}} S2 (на что потребовалось {\displaystyle D(i-1,\ j)}D(i-1, j) операций), значит, всего потребовалось {\displaystyle D(i-1,\ j)+1} D(i-1, j) +1 операций
2. Символ «b», на который кончается S2 {\displaystyle S\_{2}}, в какой-то момент был добавлен. Сделаем это добавление последней операцией. Мы превратили {\displaystyle S\_{1}} в S1 первые {\displaystyle j-1}j-1 символов {\displaystyle S\_{2}} S2, после чего добавили «b». Аналогично предыдущему случаю, потребовалось {\displaystyle D(i,\ j-1)+1} D(i, j-1) +1  операций.
3. Оба предыдущих утверждения неверны. Если мы добавляли символы справа от финального «a», то, чтобы сделать последним символом «b», 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 «a» мы не добавляли. Самого финального «a» мы не стирали, поскольку утверждение 1 неверно. Значит, единственный способ изменения последнего символа — его замена. Заменять его 2 или больше раз неоптимально. Значит:
   1. Если {\displaystyle a=b}a=b, 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, было выполнено {\displaystyle D(i-1,\ j-1)} D(i-1, j-1) операций.
   2. Если {\displaystyle a\neq b}a≠b, мы последний символ меняли один раз. Сделаем эту замену первой. В дальнейшем, аналогично предыдущему случаю, мы должны выполнить {\displaystyle D(i-1,\ j-1)} {\displaystyle D(i-1,\ j-1)} D(i-1, j-1)  операций, значит, всего потребуется {\displaystyle D(i-1,\ j-1)+1} выполнить {\displaystyle D(i-1,\ j-1)} D(i-1, j-1)+1 операций.[1,2,3,4]

# Описание работы программы

Изначально подключаем библиотеки. Определяем макрос, и затем мы объявляем функции.

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <windows.h>

#include <time.h>

#define min(f1,f2) f1<=f2?f1:f2

int LevenshteinDistance(const std::string& strWord, const std::string& strDictionaryWord, double\* time = NULL);

int LCommonSString(const char\* s1, int N1, const char\* s2, int N2, int& p1, int& p2);

int LCommonSS(const char\* s1, int N1, const char\* s2, int N2);

void CheckSpeedRunning();

Далее уже начинается выполнение программы в **main**.

Первые две строки в функции main - это выставление русского языка в консоли.  
Для того чтобы продемонстрировать работу двух функций, необходимо открыть два файла на чтение, а именно **ps.txt** и **vl.txt**. **ps.txt** - файл со списком трансформаторных подстанций по стране, а **vl.txt-** файл со списком линий электропередач (далее - ЛЭП), каждая из которых идет от одной подстанции до другой. Если хотя бы один из них не открылся, мы завершаем программу.

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

std::ifstream StationText("ps.txt");//открываем файл с названием станций

std::ifstream VLText("vl.txt");//открываем файл с названием линий

std::vector<std::string> arrStation, arrVL;

if (!StationText.is\_open())//проверяем, открылся ли файл

{

std::cout << "Не найден файл ps.txt\n";

system("PAUSE");

return 0;

}

if (!VLText.is\_open())//проверяем открылся ли файл

{

std::cout << "Не найден файл vl.txt\n";

system("PAUSE");

return 0;

}

std::string temp;

while (!StationText.eof())//считываем название станций

{

getline(StationText, temp);

arrStation.push\_back(temp);

}

StationText.close();

while (!VLText.eof())//считываем название линий

{

getline(VLText, temp);

arrVL.push\_back(temp);

}

StationText.close();

На данном этапе мы создаем результирующий файл **Result.txt** для найденных станций в линиях, а так же файл различий **Difference.txt** найденных названий станций в линиях между функциями **LCommonSS** и **LevenshteinDistance**.

std::ofstream outResult("Result.txt");

std::ofstream outDifference("Difference.txt");

if (!outResult.is\_open())//проверяем

{

std::cout << "Не удалось создать выходной файл\n";

system("PAUSE");

return 0;

}

Создаем цикл по названиям всех ЛЭП.

for (int i = 0; i < arrVL.size(); i++)

{

int LevenshteinMin1 = MAXINT, LevenshteinMin2 = MAXINT, numberMin1 = -1, numberMin2 = -1, TempLen;

int LCommonSSNumber = -1, LCommonSSMin = -1;

for (int j = 0; j < arrStation.size(); j++)//цикл по названиям всех станций

{

Находим расстояние Левенштейна и, если оно меньше наименьшего предыдущего расстояния Левенштейна, то запоминаем его, и номер, для какой станции это расстояние верно.

TempLen = LevenshteinDistance(arrVL[i], arrStation[j]);

if (TempLen < LevenshteinMin1)

{

LevenshteinMin1 = TempLen;

numberMin1 = j;

}

Далее находим наибольшую общую подстроку и, если она больше наибольшей предыдущей подстроки, то запоминаем её, и номер, для какой станции эта подстрока верна.

TempLen = LCommonSS(arrVL[i].c\_str(), arrVL[i].length(), arrStation[j].c\_str(), arrStation[j].length());

if (TempLen> LCommonSSMin)

{

LCommonSSMin = TempLen;

LCommonSSNumber = j;

}

}

Если номер с наименьшим расстоянием Левенштейна и наибольшей общей подстрокой различны, то записываем названия станций ЛЭП и эти строки в файл **Difference.txt**.

if (LCommonSSNumber != numberMin1)

{

outDifference << i << ")" << arrVL[i] << std::endl;

outDifference << "LCommonSS: " << arrStation[LCommonSSNumber] << std::endl;

outDifference << "LevenshteinDistance: " << arrStation[numberMin1] << std::endl;

}

Далее «вырезаем» из названия ЛЭП наибольшую подстроку между её названием и названием найденной с помощью расстояния Левенштейна для неё подстанции.

int nP1 = 0, nP2 = 0;

int nMaxLenSS = LCommonSString(arrVL[i].c\_str(), arrVL[i].length(), arrStation[numberMin1].c\_str(), arrStation[numberMin1].length(), nP1, nP2);

std::string strTemp(arrVL[i]);

std::string strDebug(arrStation[numberMin1]);

if (nP1 == -1)

continue;

if (nMaxLenSS + nP1 >= strTemp.length())

strTemp.erase(nP1);

else

strTemp.erase(nP1, nMaxLenSS - 1);

Из строки, с первым «вырезанным» названием, находим вторую подходящую подстанцию за счет расстояния Левенштейна тем же способом.

for (int j = 0; j < arrStation.size(); j++)

{

TempLen = LevenshteinDistance(strTemp, arrStation[j]);

if (TempLen < LevenshteinMin2)

{

LevenshteinMin2 = TempLen;

numberMin2 = j;

}

}

Выводим в результирующий файл, а именно **Result.txt**, название ЛЭП и двух подходящих станций.

outResult << i + 1 << ")" << arrVL[i] << std::endl;

if (numberMin1 != -1)

outResult << arrStation[numberMin1] << std::endl;

if (numberMin2 != -1)

outResult << arrStation[numberMin2] << std::endl;

}

outResult.close();

outDifference.close();

CheckSpeedRunning();

system("PAUSE");

return 0;

}

Функция **CheckSpeedRunning** проводит временное сравнение функций **LCommonSS** и **LevenshteinDistance** по заранее заданным файлам и записывает результат в файл. То есть, функция находит во вложенной папке **Text** название 18 файлов и сравнивает их между собой, записывая затраченное время.  
Изначально создаем файл со сравнением по времени. Прописываем названия испорченных файлов и их оригиналов. Затем производим цикл по всем парам файлов, в котором прописываем относительный путь как к оригиналу, так и к испорченному файлу. Если не находим какой-либо файл, то берем следующую пару.

Далее считываем весь файл с первоначальным(оригинальным) текстом в одну строку, затем весь файл с испорченным текстом также в одну строку и, с помощью переменной LevensteinTime, засекаем время выполнения без использования системных вызовов выделения памяти. Затем засекаем и рассчитываем время для выполнения функции LCommonSS. В конце записываем название файлов, время работы функций и закрываем файл.

void CheckSpeedRunning()

{

std::ofstream outTime("TimeCompare.txt");

outTime.setf(std::ios\_base::fixed, std::ios\_base::floatfield);

std::string CorruptionFileName[9] = { "10.txt","30.txt","100.txt","300.txt","1000.txt","3000.txt","10000.txt","30000.txt","100000.txt"};

std::string OriginalFinlName[9] = { "10\_orig.txt","30\_orig.txt","100\_orig.txt","300\_orig.txt","1000\_orig.txt","3001\_orig.txt","10000\_orig.txt","30000\_orig.txt","100000\_orig.txt"};

for (int i = 0; i < 9; i++)

{

int nLev = 1, nLComon = 1;

std::string CorruptionText;

std::string OriginalText;

std::string CurrentCorruptionFileNamel = "Text//"+ CorruptionFileName[i];

std::string CurrentOriginalFileNamel = "Text//" + OriginalFinlName[i];

std::ifstream CorruptionFile(CurrentCorruptionFileNamel);

std::ifstream OriginalFile(CurrentOriginalFileNamel);

if (!OriginalFile.is\_open() || !CorruptionFile.is\_open())

continue;

getline(OriginalFile, OriginalText, '\0');

getline(CorruptionFile, CorruptionText, '\0');

double LevenshteinTime, LCommonSSTime, time;

do

{

LevenshteinTime = 0;

for (int j = 0; j < nLev; j++)

{

LevenshteinDistance(OriginalText, CorruptionText, &time);

LevenshteinTime += time;

}

nLev \*= 10;

} while (LevenshteinTime==0);

LevenshteinTime /= nLev;

do

{

LCommonSSTime = 0;

for (int j = 0; j < nLComon; j++)

{

double TimeStart, TimeEnd;

TimeStart = clock();

LCommonSS(OriginalText.c\_str(), OriginalText.length(), CorruptionText.c\_str(), CorruptionText.length());

TimeEnd = clock();//время выхода из функции LCommonSS

LCommonSSTime += (double)(TimeEnd - TimeStart) / CLOCKS\_PER\_SEC;

}

nLComon \*= 10;

} while (LCommonSSTime == 0);

LCommonSSTime /= nLComon;

outTime << "Сравнение файлов "<< OriginalFinlName[i] <<" и "<< CorruptionFileName[i] <<std::endl;

outTime << "Время выполнения LevenshteinDistance: "; outTime<< LevenshteinTime << " c\n";

outTime << "Время выполнения LCommonSS: " << LCommonSSTime << " c\n";

}

outTime.close();//закрываем файл

}

Принцип работы алгоритма **LevenshteinDistance** заключается в том, что находим минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. В данном алгоритме передаются две строки, а возвращается расстояние Левенштейна. **strWord** –первая строка, **strDictionaryWord**-вторая строка, **time**- параметр, который, возвращает при необходимости затраченное время на работу без системных вызовов на выделение памяти, если нет необходимости, то этот параметр пропускается.

int LevenshteinDistance(const std::string& strWord, const std::string& strDictionaryWord,double\* time)

{

if (strWord.empty() || strDictionaryWord.empty())

return strWord.length() > strDictionaryWord.length() ? strWord.length() : strDictionaryWord.length();

int\*\* pMatrixDistance;

int nString(strWord.length() + 1);

int nColumn(strDictionaryWord.length() + 1);

int nFirst, nSecond, nThird;

int nCmp;

double TimeStart, TimeEnd;

pMatrixDistance = new int\* [nString];

for (int i = 0; i < nString; i++)

{

pMatrixDistance[i] = new int[nColumn];

}

for (int i = 0; i < nString; i++)

{

pMatrixDistance[i][0] = i;

}

for (int i = 1; i < nColumn; i++)

{

pMatrixDistance[0][i] = i;

}

if (time!=NULL)

TimeStart = clock();

for (int i = 1; i < nString; ++i)

{

for (int j = 1; j < nColumn; ++j)

{

nFirst = pMatrixDistance[i - 1][j] + 1;

nSecond = pMatrixDistance[i][j - 1] + 1;

if (strWord[i - 1] == strDictionaryWord[j - 1])

nCmp = 0;

else

nCmp = 1;

nThird = pMatrixDistance[i - 1][j - 1] + nCmp;

pMatrixDistance[i][j] = min(nFirst, nSecond);

pMatrixDistance[i][j] = min(pMatrixDistance[i][j], nThird);

}

}

if (time != NULL)

{

TimeEnd = clock();

\*time = (double)(TimeEnd - TimeStart) / CLOCKS\_PER\_SEC;

}

int nAnswer = pMatrixDistance[nString - 1][nColumn - 1];

for (int i = 0; i < nString; i++)

{

delete[] pMatrixDistance[i];

}

delete[] pMatrixDistance;

return nAnswer;

}

Принцип работы алгоритма **LCommonSString** заключается в том, что находим самую длинную общую подстроку у двух строк, а возвращает она её длину. **(s1,N1), (s2,N2)** - строки и их длины, **p1,p2** - позиции найденных подстрок в строках.

int LCommonSString(const char\* s1, int N1, const char\* s2, int N2, int& p1, int& p2)

{

p1 = p2 = -1;

if (N1 <= 0 || N2 <= 0) return 0;

int maxL = 0;

for (int q1 = 0; q1 < N1; q1++)

for (int q2 = 0; q2 < N2; q2++) {

if (s1[q1] != s2[q2]) continue;

int len = 1, r1 = q1 + 1, r2 = q2 + 1;

while (s1[r1] == s2[r2]&&r1<N1&&r2<N2){

len++;

if (++r1 == N1) break;

if (++r2 == N2) break;

}

if (len > maxL) {

maxL = len;

p1 = q1;

p2 = q2;

}

}

return maxL;

}

Принцип работы функции **LCommonSS** заключается в нахождении суммы всех найденных подстрок путем разбиения двух строк.

int LCommonSS(const char\* s1, int N1, const char\* s2, int N2)

{

int p1, p2;

int L = LCommonSString(s1, N1, s2, N2, p1, p2);// самая длинная общая подстрока

if (L == 0) return 0;

int LL = LCommonSS(s1, p1, s2, p2); // общая длина в левой части

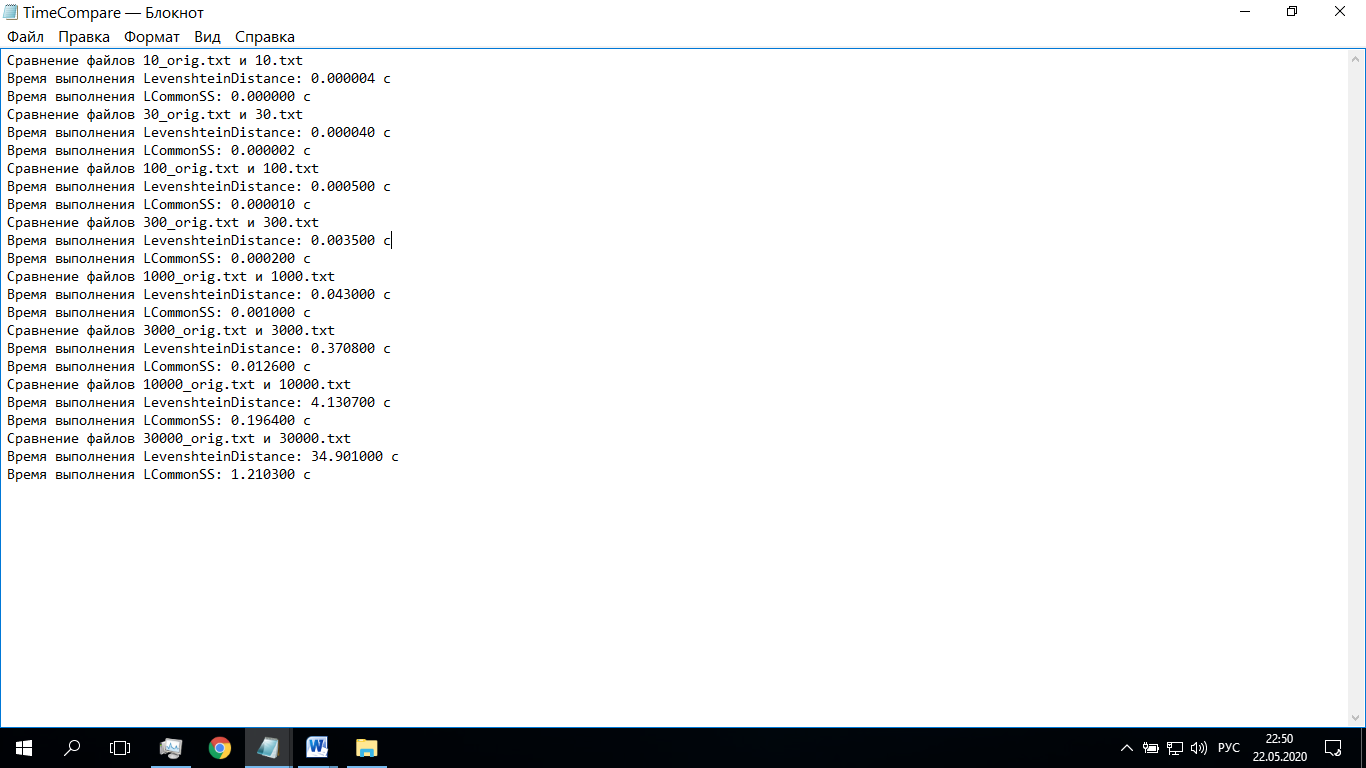
int LR = LCommonSS(s1 + p1 + L, N1 - p1 - L, s2 + p2 + L, N2 - p2 - L);// общая длина в правой части

return LL + L + LR;

}

# Сравнение времени работы алгоритмов

В результате проделанной работы было найдено время, за которое будет находиться расстояние Левенштейна между строками длины 10, 30,100, 300, 1000, 3000, 10000 и 30000 символов.



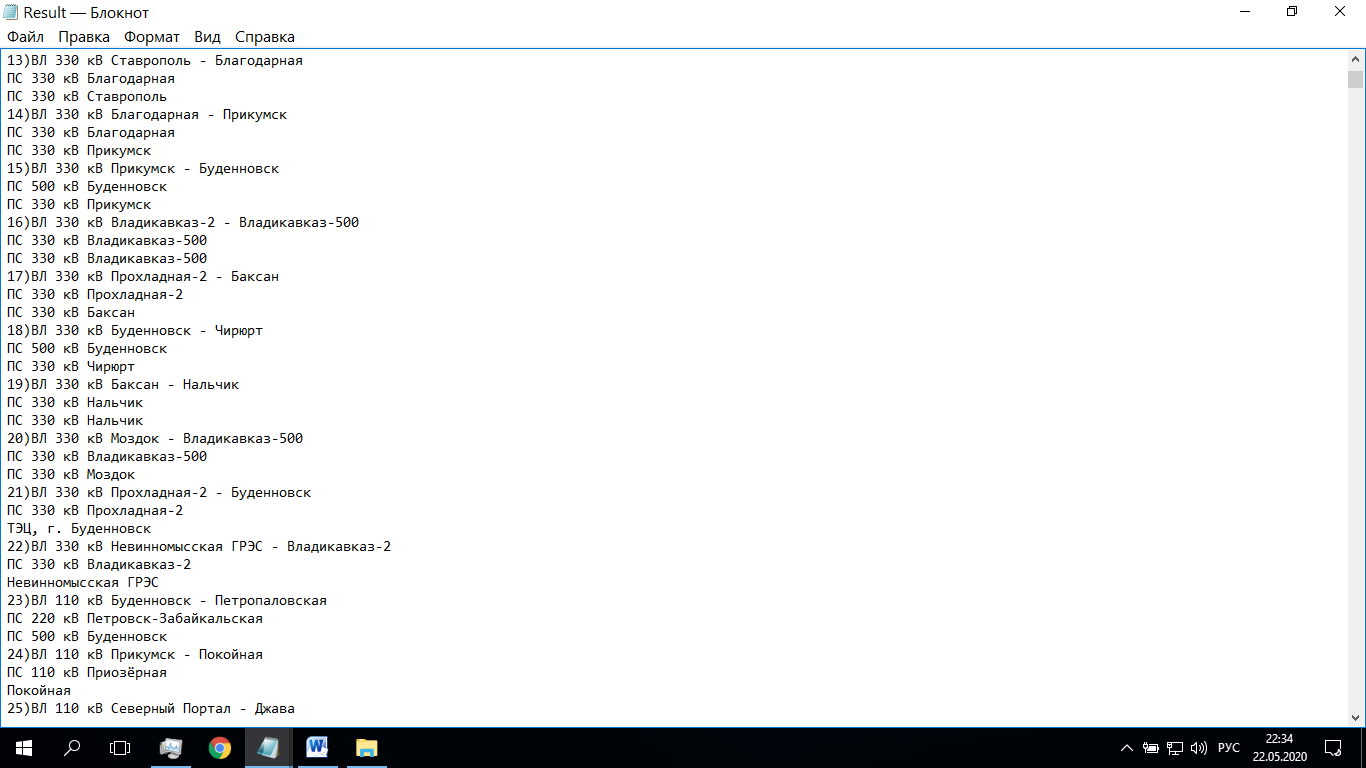
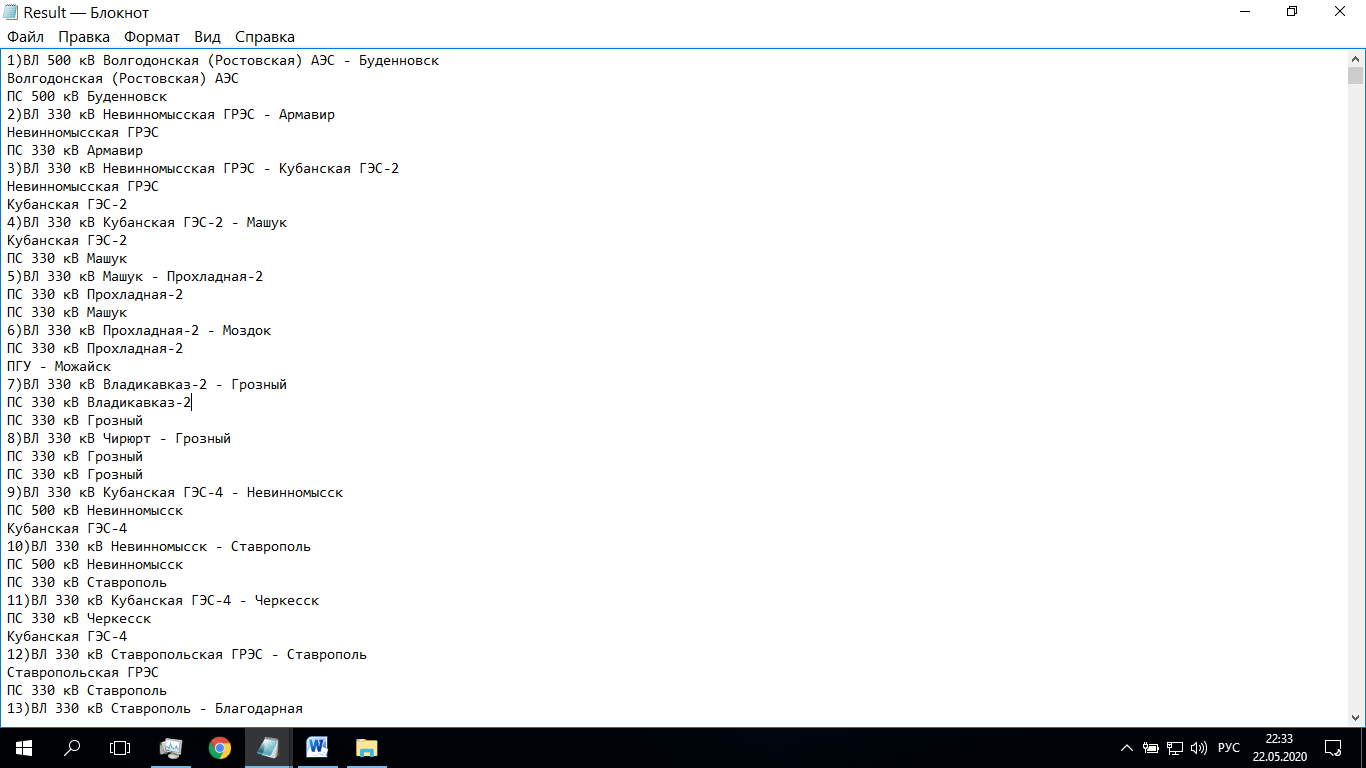
Ниже приведена следующая таблица (в секундах):

|  |  |  |
| --- | --- | --- |
| Длина | LevensteinDistance | LCommonSS |
| 10 | 0.000004c | <0.000004c |
| 30 | 0.00004с | 0.000002c |
| 100 | 0.0005с | 0.00001c |
| 300 | 0.0035с | 0.0002с |
| 1000 | 0.043с | 0.001с |
| 3000 | 0.3708с | 0.0126с |
| 10000 | 4.1307с | 0.1964с |
| 30000 | 34.091с | 1.2103с |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Исходя из этих данных был подобран тип функции, который бы отражал зависимость времени выполнения компьютерной функции от количества символов. Для этого:   * 1. Рассматривались разные шкалы, чтобы график был почти прямой линией. Определили, что это степенная функция вида:   y=C3\*xC4,  где х- длина строки, С3 и С4 –некоторые константы.   * 1. Для степенной шкалы были найдены коэффициенты этой прямой линии.   В таблице, приведенной ниже, производились рассчеты для аппроксимации функций LevensteinDistance и LCommonSS.  Здесь appr\_Lvs- аппроксимация функции LevensteinDistance, appr\_LCom- аппроксимация функции LCommonSS. | |  | | --- | |  | |  |  |  |  |  |  |
| Так же показано приближение графически. Lvs- график функции LevensteinDistance, Com – график функции LCommonSS.  Таким образом, делаем вывод, что:  Время(LevensteinDistance) ~= 6e-8\*Len2 секунд. Время(LCommonSS) ~= 1e-9\*Len2 секунд. Или в тактах процессора:  Время(LevensteinDistance) ~= 180\*Len2 тактов в секунду.  Время(LCommonSS) ~= 3\*Len2 тактов в секунду.  . |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

# Результаты работы программы

В результате работы программы мы получили следующие результаты.



1)ВЛ 500 кВ Волгодонская (Ростовская) АЭС – Буденновск

Волгодонская (Ростовская) АЭС

ПС 500 кВ Буденновск

2)ВЛ 330 кВ Невинномысская ГРЭС - Армавир

Невинномысская ГРЭС

ПС 330 кВ Армавир

3)ВЛ 330 кВ Невинномысская ГРЭС - Кубанская ГЭС-2

Невинномысская ГРЭС

Кубанская ГЭС-2

4)ВЛ 330 кВ Кубанская ГЭС-2 - Машук

Кубанская ГЭС-2

ПС 330 кВ Машук

5)ВЛ 330 кВ Машук - Прохладная-2

ПС 330 кВ Прохладная-2

ПС 330 кВ Машук

6)ВЛ 330 кВ Прохладная-2 - Моздок

ПС 330 кВ Прохладная-2

ПГУ - Можайск

7)ВЛ 330 кВ Владикавказ-2 - Грозный

ПС 330 кВ Владикавказ-2

ПС 330 кВ Грозный

8)ВЛ 330 кВ Чирюрт - Грозный

ПС 330 кВ Грозный

ПС 330 кВ Грозный

9)ВЛ 330 кВ Кубанская ГЭС-4 - Невинномысск

ПС 500 кВ Невинномысск

Кубанская ГЭС-4

10)ВЛ 330 кВ Невинномысск - Ставрополь

ПС 500 кВ Невинномысск

ПС 330 кВ Ставрополь

11)ВЛ 330 кВ Кубанская ГЭС-4 - Черкесск

ПС 330 кВ Черкесск

Кубанская ГЭС-4

12)ВЛ 330 кВ Ставропольская ГРЭС - Ставрополь

Ставропольская ГРЭС

ПС 330 кВ Ставрополь

13)ВЛ 330 кВ Ставрополь - Благодарная

ПС 330 кВ Благодарная

ПС 330 кВ Ставрополь

14)ВЛ 330 кВ Благодарная - Прикумск

ПС 330 кВ Благодарная

ПС 330 кВ Прикумск

15)ВЛ 330 кВ Прикумск - Буденновск

ПС 500 кВ Буденновск

ПС 330 кВ Прикумск

16)ВЛ 330 кВ Владикавказ-2 - Владикавказ-500

ПС 330 кВ Владикавказ-500

ПС 330 кВ Владикавказ-500

17)ВЛ 330 кВ Прохладная-2 - Баксан

ПС 330 кВ Прохладная-2

ПС 330 кВ Баксан

18)ВЛ 330 кВ Буденновск - Чирюрт

ПС 500 кВ Буденновск

ПС 330 кВ Чирюрт

19)ВЛ 330 кВ Баксан - Нальчик

ПС 330 кВ Нальчик

ПС 330 кВ Нальчик

20)ВЛ 330 кВ Моздок - Владикавказ-500

ПС 330 кВ Владикавказ-500

ПС 330 кВ Моздок

21)ВЛ 330 кВ Прохладная-2 - Буденновск

ПС 330 кВ Прохладная-2

ТЭЦ, г. Буденновск

22)ВЛ 330 кВ Невинномысская ГРЭС - Владикавказ-2

ПС 330 кВ Владикавказ-2

Невинномысская ГРЭС

23)ВЛ 110 кВ Буденновск - Петропаловская

ПС 220 кВ Петровск-Забайкальская

ПС 500 кВ Буденновск

24)ВЛ 110 кВ Прикумск - Покойная

ПС 110 кВ Приозёрная

Покойная

25)ВЛ 110 кВ Северный Портал - Джава

ПС 110 кВ Северный Портал

ПГУ - Можайск

26)ВЛ 6 кВ Грозный - Бердыкель

ПС 330 кВ Грозный

Бердыкель

27)ВЛ 330 кВ Баксан - Кисловодск

ПС 330 кВ Баксан

Кисловодск

28)ВЛ 330 кВ Кисловодск - Черкесск

ПС 330 кВ Черкесск

ПС 330 кВ Новгородская

29)ВЛ 330 кВ Невинномысская ГРЭС - Кубанская ГЭС-4

Невинномысская ГРЭС

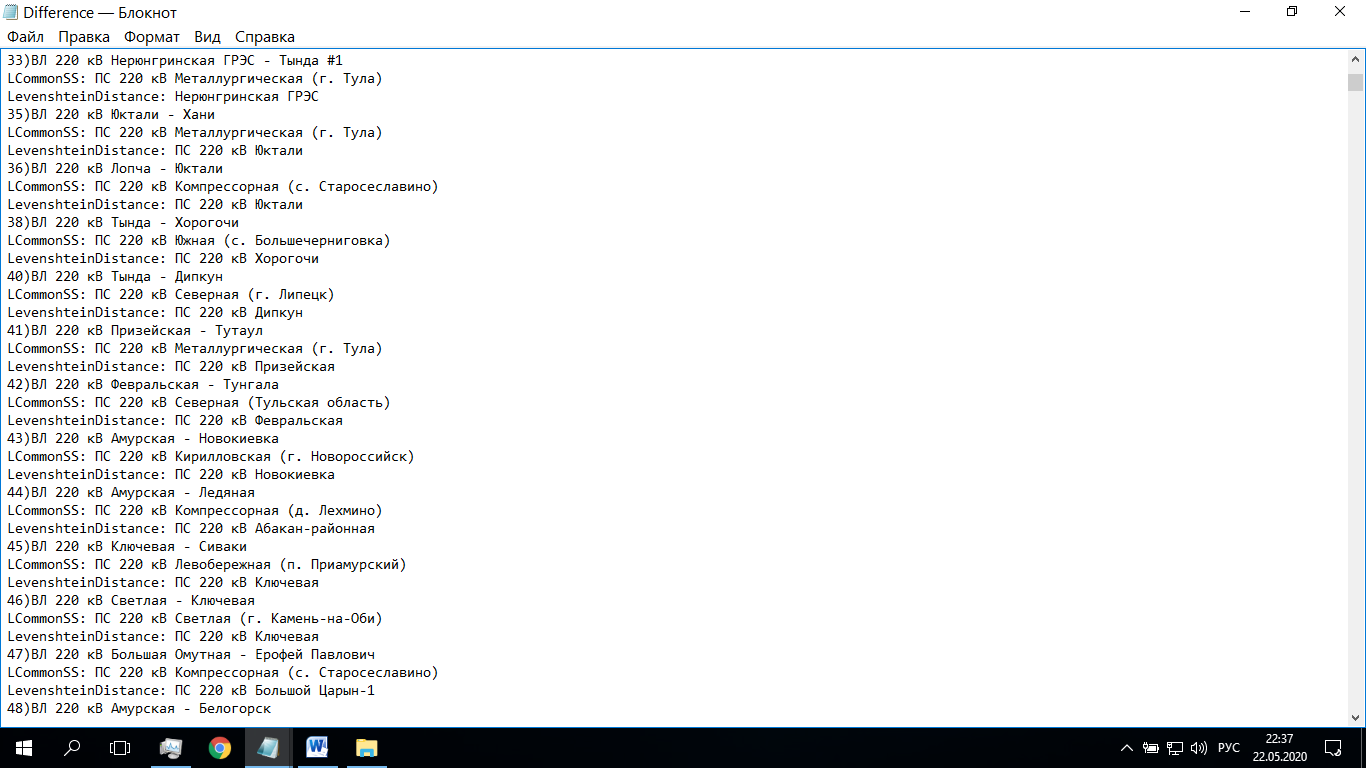
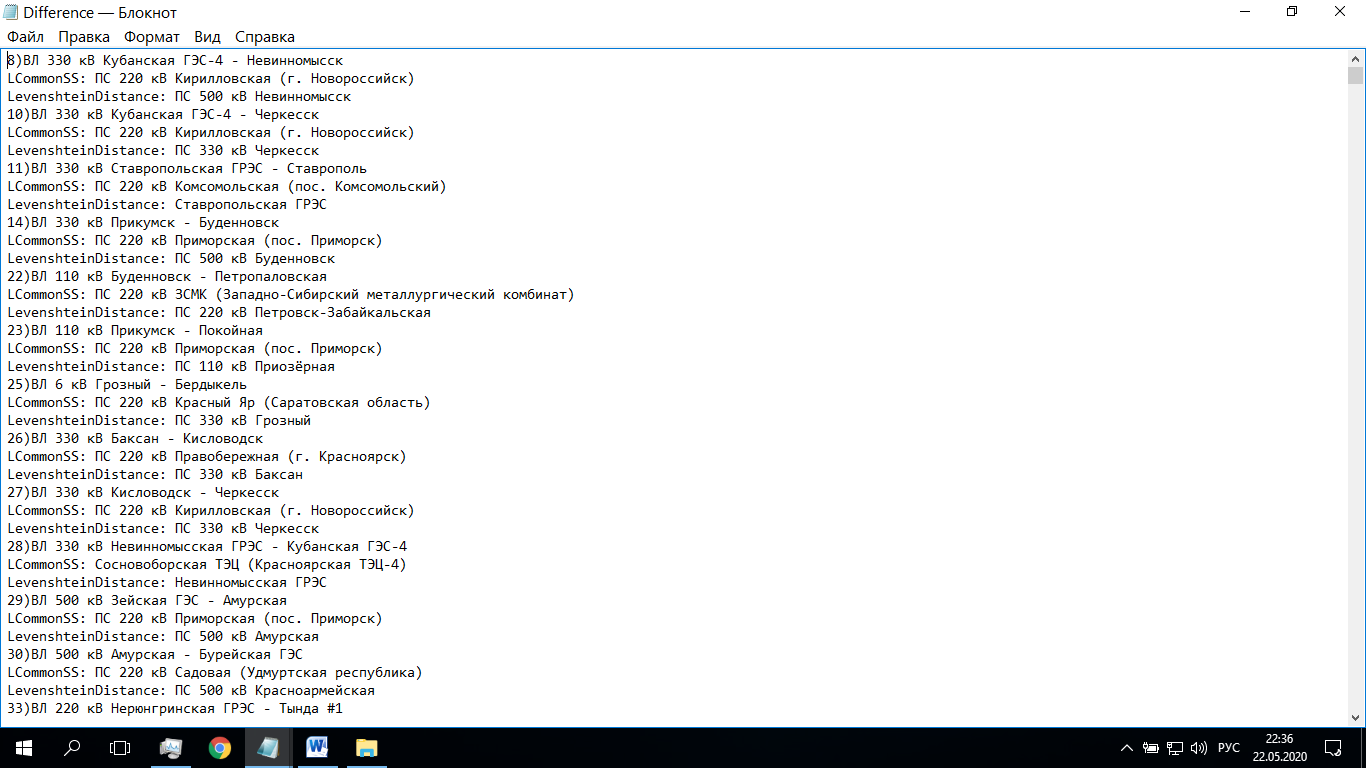
Кубанская ГЭС-4

30)ВЛ 500 кВ Зейская ГЭС - Амурская

ПС 500 кВ Амурская

ПС 500 кВ Томская

И это лишь 30 из 2003 результатов. Мы попытались найти максимально приближенно строки из одного файла, соответствующие строкам другого файла. Было выполнено 3 847 763 сравнений. По времени сопоставление заняло порядка 24 минут. Оба метода корректны, но алгоритм расстояния Левенштейна делает это почти в 93,9 раз медленнее, чем алгоритм длины наибольшей подстроки. Идеального решения у задачи нет.  
Так же мы попытались показать случаи, когда функции вывели два разных результата.



8)ВЛ 330 кВ Кубанская ГЭС-4 - Невинномысск

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 500 кВ Невинномысск

10)ВЛ 330 кВ Кубанская ГЭС-4 - Черкесск

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 330 кВ Черкесск

11)ВЛ 330 кВ Ставропольская ГРЭС - Ставрополь

LCommonSS: ПС 220 кВ Комсомольская (пос. Комсомольский)

LevenshteinDistance: Ставропольская ГРЭС

14)ВЛ 330 кВ Прикумск - Буденновск

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 500 кВ Буденновск

22)ВЛ 110 кВ Буденновск - Петропаловская

LCommonSS: ПС 220 кВ ЗСМК (Западно-Сибирский металлургический комбинат)

LevenshteinDistance: ПС 220 кВ Петровск-Забайкальская

23)ВЛ 110 кВ Прикумск - Покойная

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 110 кВ Приозёрная

25)ВЛ 6 кВ Грозный - Бердыкель

LCommonSS: ПС 220 кВ Красный Яр (Саратовская область)

LevenshteinDistance: ПС 330 кВ Грозный

26)ВЛ 330 кВ Баксан - Кисловодск

LCommonSS: ПС 220 кВ Правобережная (г. Красноярск)

LevenshteinDistance: ПС 330 кВ Баксан

27)ВЛ 330 кВ Кисловодск - Черкесск

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 330 кВ Черкесск

28)ВЛ 330 кВ Невинномысская ГРЭС - Кубанская ГЭС-4

LCommonSS: Сосновоборская ТЭЦ (Красноярская ТЭЦ-4)

LevenshteinDistance: Невинномысская ГРЭС

29)ВЛ 500 кВ Зейская ГЭС - Амурская

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 500 кВ Амурская

30)ВЛ 500 кВ Амурская - Бурейская ГЭС

LCommonSS: ПС 220 кВ Садовая (Удмуртская республика)

LevenshteinDistance: ПС 500 кВ Красноармейская

33)ВЛ 220 кВ Нерюнгринская ГРЭС - Тында #1

LCommonSS: ПС 220 кВ Металлургическая (г. Тула)

LevenshteinDistance: Нерюнгринская ГРЭС

35)ВЛ 220 кВ Юктали - Хани

LCommonSS: ПС 220 кВ Металлургическая (г. Тула)

LevenshteinDistance: ПС 220 кВ Юктали

36)ВЛ 220 кВ Лопча - Юктали

LCommonSS: ПС 220 кВ Компрессорная (с. Старосеславино)

LevenshteinDistance: ПС 220 кВ Юктали

38)ВЛ 220 кВ Тында - Хорогочи

LCommonSS: ПС 220 кВ Южная (с. Большечерниговка)

LevenshteinDistance: ПС 220 кВ Хорогочи

40)ВЛ 220 кВ Тында - Дипкун

LCommonSS: ПС 220 кВ Северная (г. Липецк)

LevenshteinDistance: ПС 220 кВ Дипкун

41)ВЛ 220 кВ Призейская - Тутаул

LCommonSS: ПС 220 кВ Металлургическая (г. Тула)

LevenshteinDistance: ПС 220 кВ Призейская

42)ВЛ 220 кВ Февральская - Тунгала

LCommonSS: ПС 220 кВ Северная (Тульская область)

LevenshteinDistance: ПС 220 кВ Февральская

43)ВЛ 220 кВ Амурская - Новокиевка

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 220 кВ Новокиевка

44)ВЛ 220 кВ Амурская – Ледяная

LCommonSS: ПС 220 кВ Компрессорная (д. Лехмино)

LevenshteinDistance: ПС 220 кВ Абакан-районная

45)ВЛ 220 кВ Ключевая - Сиваки

LCommonSS: ПС 220 кВ Левобережная (п. Приамурский)

LevenshteinDistance: ПС 220 кВ Ключевая

46)ВЛ 220 кВ Светлая - Ключевая

LCommonSS: ПС 220 кВ Светлая (г. Камень-на-Оби)

LevenshteinDistance: ПС 220 кВ Ключевая

47)ВЛ 220 кВ Большая Омутная - Ерофей Павлович

LCommonSS: ПС 220 кВ Компрессорная (с. Старосеславино)

LevenshteinDistance: ПС 220 кВ Большой Царын-1

48)ВЛ 220 кВ Амуpская - Белогорск

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 220 кВ Белогорск

49)ВЛ 220 кВ Белогорск - Завитая

LCommonSS: ПС 220 кВ Шерловогорская

LevenshteinDistance: ПС 220 кВ Белогорск

Это часть результатов. Более того, демонстрируется какая функция и что именно нашла для этой строки. По времени, поиск различий занял так же порядка 24 минут.

# Заключение

Исходя из совокупности вышеперечисленных факторов, можем сделать следующие выводы:

а) Касательно времени работы двух алгоритмов следует отметить, что функция **LevensteinDistance** делает это существенно медленнее, нежели функция **LCommonSS** между строками различной длины.

б) Касательно качества работы двух алгоритмов на рассмотренных  
примерах следует отметить, что в спорных случаях алгоритм **LevensteinDistance** дает заметно лучшие результаты, чем **LCommonSS**.

Таким образом, алгоритм **LevensteinDistance** по длительности выполнения гораздо медленнее второго алгоритма, но выдает заметно лучшие результаты. Можем это видеть на следующих примерах:

10)ВЛ 330 кВ Кубанская ГЭС-4 - Черкесск

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 330 кВ Черкесск

11)ВЛ 330 кВ Ставропольская ГРЭС - Ставрополь

LCommonSS: ПС 220 кВ Комсомольская (пос. Комсомольский)

LevenshteinDistance: Ставропольская ГРЭС

14)ВЛ 330 кВ Прикумск - Буденновск

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 500 кВ Буденновск

Разберем вышеупомянутые примеры более подробно.  
Первый пример:

ВЛ 330 кВ Кубанская ГЭС-4 - Черкесск

LCommonSS: ПС 220 кВ Кирилловская (г. Новороссийск)

LevenshteinDistance: ПС 330 кВ Черкесск  
Рассмотрим работу алгоритма LCommonSS.

Для начала, берется первый символ из первой строки и сравнивается с каждым символом из 2 строки. Если символы совпадают, то включается 3 цикл, который проверяет каждый последующий символ каждой строки друг с другом до тех пор, пока они не станут различными, или пока не кончится строка. Наилучший результат из 3 цикла сохраняется, и это получается количество символов наибольшей общей подстроки. Так же запоминается начало и конец общей подстроки в каждой строке.

Далее запускается эта же функция еще раз только для подстрок, которые получились слева и справа от наибольшей общей подстроки. Далее – рекурсия.

В конце возвращается сумма всех найденных подстрок.

ВЛ 330 кВ Кубанская ГЭС-4 – Черкесск  
ПС 220 кВ Кирилловская (г. Новороссийск)

В первый раз наибольшая общая подстрока «0\_кВ\_К»- **6 символов** (так как пробелы(здесь-нижнее подчеркивание) тоже считаются за символ).  
Берем подстроки слева и справа от нее. Слева:

ВЛ 33

ПС\_22  
Находим общую подстроку: «\_»- **1 символ.** Более не найдем уже каких-либо общих подстрок. Теперь начинаем поиск справа.  
 убанская\_ГЭС-4 – Черкесск  
 ирилловская\_(г. Новороссийск)  
Находим общую подстроку: **«ская\_» – 5 символов.** Снова ищем подстроки слева и справа.Слева:

убан

ирилловНаибольшей общей подстроки нет. Более не найдем. Ищем справа:  
ГЭС-4 – Черкесск  
(г. Новороссийск)  
Получаем: «**сс**»- **2 символа**. Далее, снова ищем подстроки слева и справа и так продолжается до тех пор, пока не находим слева «**\_**» и «**р**» и справа «**к**», то есть, **3символа**:

ГЭС-4 – Черке

(г. Новоро

к

ийск)

Считаем до тех пор, пока не закончатся наибольшие общие подстроки.   
И в конце складываем получившиеся результаты, то есть, совпадения: **17**.  
Теперь же рассмотрим алгоритм нахождения расстояния Левенштейна.  
ВЛ 330 кВ Кубанская ГЭС-4 - Черкесск  
ПС 330 кВ Черкесск  
Исходя из определения: это минимальное количество односимвольных операций (а именно вставки, удаления и замены), необходимых для превращения одной последовательности символов в другую.

Сопоставляем строки:

ВЛ 330 кВ ~~Кубанская ГЭС-4 –~~ Черкесск

ПС 330 кВ Черкесск  
Расстояние Левенштейна будет равно 20, так как минимизируется количество замен тем, что используется удаление «\_ ~~Кубанская ГЭС-4 –~~\_» и замена «ВЛ» на «ПС». Таким образом, замена + удаление =**20**.

Перейдем ко второму примеру:

ВЛ 330 кВ Ставропольская ГРЭС - Ставрополь

LCommonSS: ПС 220 кВ Комсомольская (пос. Комсомольский)

LevenshteinDistance: Ставропольская ГРЭС

Итак, согласно работе алгоритма LCommonSS:  
 ВЛ 330 кВ Ставропольская ГРЭС - Ставрополь

ПС 220 кВ Комсомольская (пос. Комсомольский)

Ищем наибольшую общую подстроку.«**ольская\_**» – **8 символов**. Начинаем поиск подстроки слева:

ВЛ 330 кВ Ставроп  
ПС 220 кВ Комсом

Получаем «0\_кВ**\_**» - наибольшая общая подстрока, **равная 5.** Затем, рассматриваем справа:

ГРЭС – Ставрополь

(пос. Комсомольский)  
**«оль»-** наибольшая общая подстрока, равная **3**. Снова считаем до тех пор, пока не закончатся наибольшие общие подстроки.

ВЛ 33Ставроп  
ПС 22Комсом  
Из данных двух строк можем сделать вывод, что в целом наибольшая общая подстрока будет состоять из «**\_**», «**С**» и «**о**», то есть, **3 символа**.

Возвращаемся к подстрокам:

ГРЭС – Ставрополь

(пос. Комсомольский)

ГРЭС – Ставроп

(пос. Комсом

В результате преобразований находим символы «**\_**» и «**о**», то есть, **2 символа.**

Более сравнений не предусматривается. Таким образом, наибольшая общая подстрока равна **21**.

Рассмотрим расстояние Левенштейна. Исходя из определения, имеем:

ВЛ\_330\_кВ\_Ставропольская ГРЭС\_–\_Ставрополь

Ставропольская ГРЭС

~~ВЛ\_330\_кВ\_~~Ставропольская ГРЭС~~\_-\_Ставрополь~~(удаляем все, кроме выделенного красным, минимальная цена будет именно такова).Расстояние Левенштейна = 23.

Рассмотрим последний, третий пример:

ВЛ 330 кВ Прикумск - Буденновск

LCommonSS: ПС 220 кВ Приморская (пос. Приморск)

LevenshteinDistance: ПС 500 кВ Буденновск

Согласно алгоритму LCommonSS:

ВЛ 330\_кВ\_Прикумск - Буденновск

ПС 220\_кВ\_Приморская (пос. Приморск)  
**«0\_кВ\_При»**- наибольшая общая подстрока, **8 символов.**

Слева – только «**\_**», **1 символ**:

ВЛ\_33

ПС\_22  
Справа:

кумск - Буденновск

морская (пос. Приморск)

«**ск**», **2 символа.** Справа остается только «)». Подстрока слева:

кумск\_–\_Буденнов

морская\_(пос. Примор  
Путем преобразований, находим и суммируем найденное количество наибольших общих подстрок, а именно **5 символов**: «**ск**», «**м**», «**\_**» и «**о**».  
Следовательно, наибольшая общая подстрока равна **16 символов**.

Найдем расстояние Левенштейна:

ВЛ\_330\_кВ\_Прикумск\_-\_Буденновск

ПС\_500\_кВ\_\_\_\_\_\_\_\_\_\_\_\_Буденновск  
Самый оптимальный вариант в данном случае – это удалить «\_Прикумск\_-» (11 символов) и выполнить замену «ВЛ» на «ПС», «33» на «50»(4 символа), Высчитываем редакционное расстояние и получаем: 2+2+11=**15**.

В качестве вывода следует отметить, что расстояние Левенштейна используется для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи), а так же для сравнения текстов. Расстояние Левенштейна широко используется в компьютерной лингвистике и теории информации. Так же активно применяется в биоинформатике, а именно для сравнения хромосом, генов и белков.   
Из недостатков, с точки зрения приложений, определение расстояния между словами или текстовыми полями по Левенштейну следует отметить, что при перестановке местами слов или частей слов получаются сравнительно большие расстояния. А так же расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными.

# Список литературы

1. https://ru.wikipedia.org/wiki/Расстояние\_Левенштейна (дата обращения: 20.05.2020).
2. Бобылева, О. В. Математические аспекты метода Вагнера — Фишера / О. В. Бобылева. — Текст : непосредственный // Молодой ученый. — 2014. — № 13 (72). — С. 1-4. — URL: https://moluch.ru/archive/72/12323/ (дата обращения: 20.05.2020).
3. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 163.4:845-848.
4. Гасфилд. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, 2003.
5. https://habr.com/ru/post/117063/(дата обращения: 11.05.2020).
6. https://habr.com/ru/post/114997/( дата обращения: 24.05.2020).