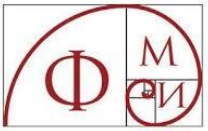
Софийски университет “Св. Климент Охридски”
Факултет по математика и информатика



Курсов проект по “Системи основани на знания” зимен семестър 2021/2022

Тема - Plagiarism system

Изработили:
Кенан Кемал Юсеин 71947,
Веселин Славов Тодоров 71923,
Лъчезар Атанасов Пещерлиев 71930
Специалност: Информационни системи
Курс: 3

Гр. София

Декември 2021

Съдържание:

1. Задание – кратко описание на решаваната задача

Поради онлайн обучението, много изпити и контролни се правят на компютър, което води до лесно преписване. По тази причина идеята на нашия проект е система за плагиатство (“plagiarism checker”). При нея можем да добавим множество файлове, като се сравнява съдържанието на файловете и намира процентът на съвпадение в текста, като използва алгоритъма на Левенщайн и някои оптимизации свързани с по-точните резултати, които той дава. Системата е имплементирана под формата на Уеб приложение, в която могат свободно да се зареждат файловете, и да изведе в кои файлове, какъв е процента на плагиатство спрямо тяхното съдържание сравнено с вече наличните в базата файлове.

2. Описание на използвания алгоритъм за решаване на задачата

Основния алгоритъм, който е използван е алгоритъма на Левенщайн, който открива броя промени(като замяна на символ с друг символ, размяна на позициите на два символа, изтриване на символ) на даден низ са нужни, за да го превърнем в друг низ (нарича се още Левенщайново разстояние). За да бъде, обаче алгоритъма оптимален трябва да премахнем всички препинателни знаци, които биха били отчетени като плагиатство. Също така, тъй като стандартно има разлика като малки и главни букви, трябва да игнорираме разликата между малки и главни букви. Алгоритмите, които ще приложим върху двата низа преди да използваме алгоритъма на Левенщайн са алгоритъм за премахване на препинателни знаци, и превръщане на главните букви в малки. След това на преработените низове прилагаме алгоритъм който премахва много често срещани думи като ‘be’, ‘when’, ‘was’ и др.

Псевдокод за алгоритъма за изчистване на препинателни знаци и игнорирането на малки и главни букви:

```
tokenize(string text):
    listOfWords = text.split("(\s+)|([.,!?:;'\"\'-])");
    for(word : listOfWords)
    {
        if(word.isEmpty())
        {
            listOfWords.remove(word);
            continue;
        }
        word = word.toLowerCase();
    }
    return listOfWords;
```

Псевдокод за алгоритъма за изчистване на често срещаните думи:

```
commonWords е листът с често срещани думи
stem(list<string> listOfWords):
    for(word : listOfWords)
    {
        if(commonWords.contains(word))
        {
            listOfWords.remove(word);
            continue;
        }
    }
    return listOfWords.joining(' ');
```

Алгоритъма на Левенщайн приема 2та низа и извършва следните стъпки:

1. Ако единия от двата низа е празен, то броя промени ще бъде дължината на втория низ, например ако имаме низовете “hello” и “” броя промени ще е 5.

2. Ако двата низа не са празни, то изчисленото разстоянието на всяка стъпка ще се държи в матрица n x m, където n е дължината на първия низ + 1, а m е дължината на втория низ + 1. Запълваме първия ред и първата колона на матрицата с последователни числа като започваме от 0.

```
for (int i = 0; i <= sourceWordCount; distance[i][0] = i++)
    ;
for (int j = 0; j <= targetWordCount; distance[0][j] = j++)
    ;
```

3. За останалите клетки в матрицата (с координати i и j) ще попълним 0, ако символа на позиция i - 1 в първия низ и символа на позиция j - 1 във втория низ, в противен случай ще попълним 1

```
for (int i = 1; i <= sourceWordCount; i++) {
    for (int j = 1; j <= targetWordCount; j++) {
        int cost = (target.charAt(j - 1) == source.charAt(i - 1) ? 0 : 1);
```

4. За всяка клетка освен от ред и колона 1 разстоянието може да бъде изчислено използвайки следния код

```
distance[i][j] = Math.min(Math.min(distance[i - 1][j] + 1, distance[i][j - 1] + 1), distance[i - 1][j - 1] + cost);
```

Псевдокод:

levenstein(fst, snd)
{
if(fst == “” or snd == “”)
{
return max(fst.size, snd.size);
}

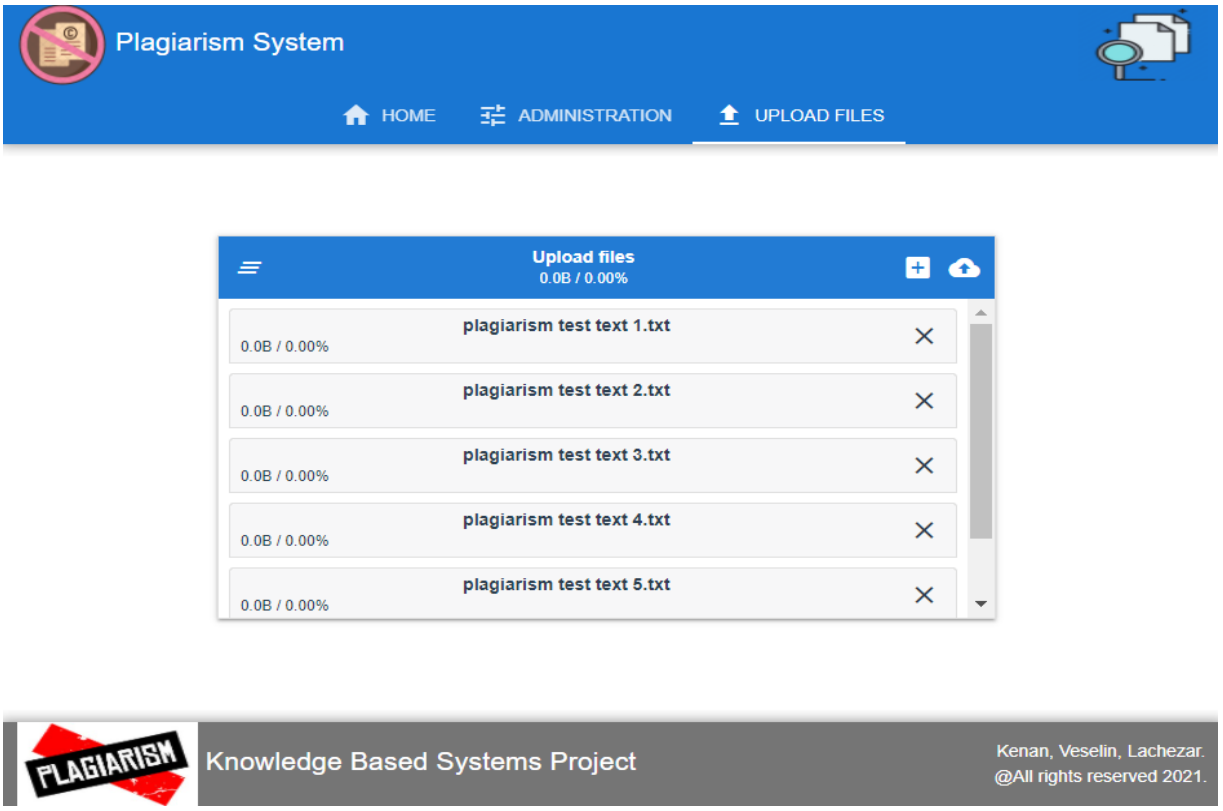
for(int i : fst.size)
{
for(int j: snd.size)
{
int cost = (fst[i] == snd[j]) ? 0 : 1;
distance[i][j] = min(
min(distance[i-1][j] + 1, distance[i][j-1]+1),
distance[i-1][j-1] + cost

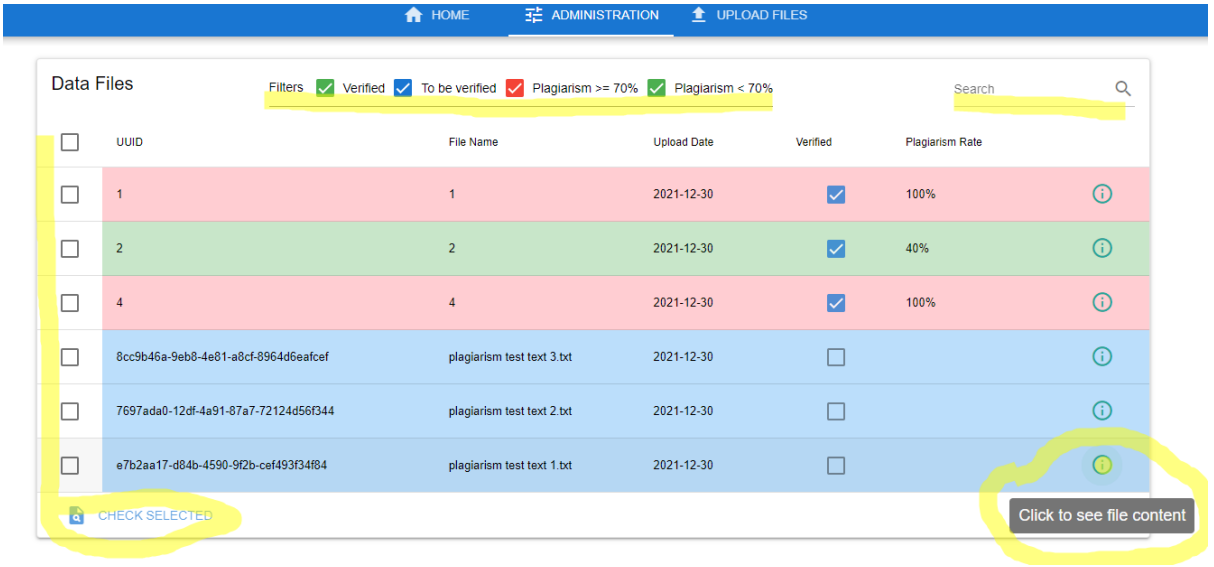
}}
return distance[fst.size-1][snd.size-1];
}

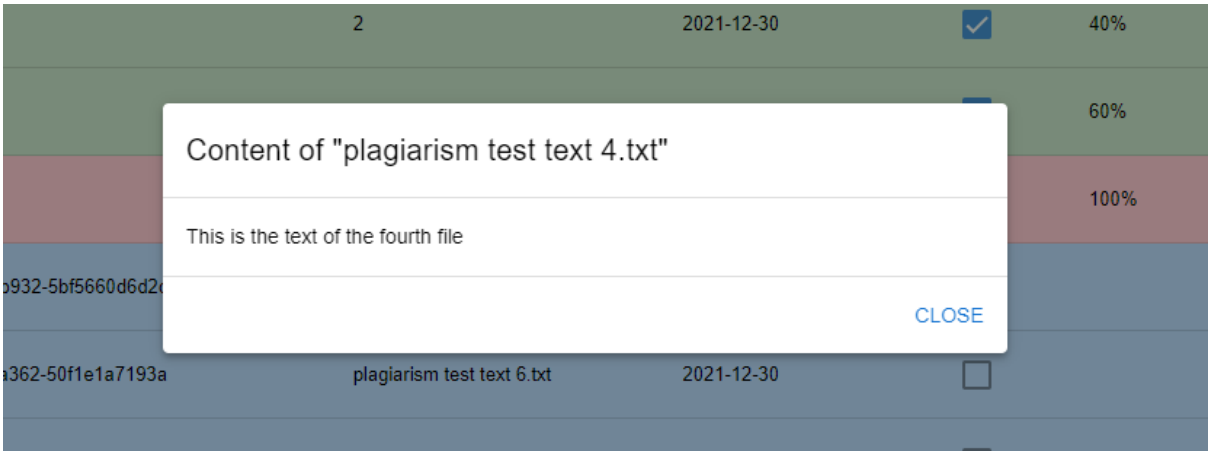
След като се намерим колко стъпки са нужни, за промяна на низа, процента плагиатство се изчислява като 1 - броя стъпки / дължината на по дългия низ.

3. Описание на данните – описание на структурата и произхода на използваните данни

Входните данни са 2 низа, а изходния резултат е цяло число от 0 до 100, като информацията идва от файлове (използва се разширение .txt, но лесно и бързо могат да бъдат добавени и други формати на файлове). В приложението има бутон за добавяне на файловете, като в база от данни се записва съдържанието на файловете и от там се извличат за прилагане на алгоритъма. Крайния резултат излиза в приложението. Списъка с често срещани думи е взет от следния адрес: [The 100 most common words in English – Espresso English]().



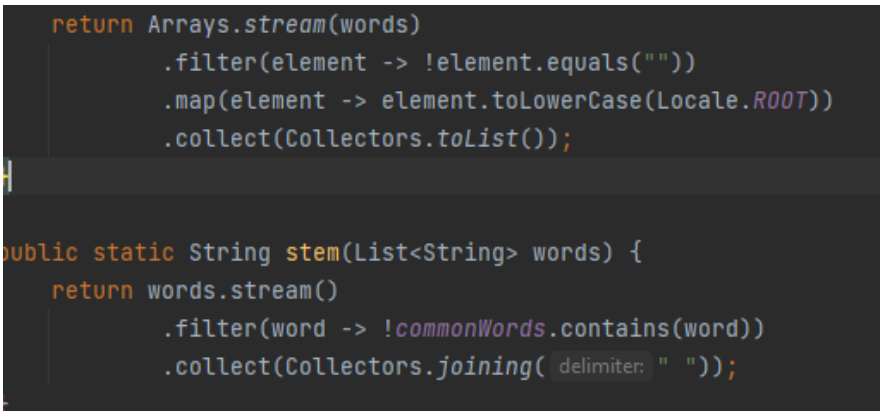

HOME
ADMINISTRATION
UPLOAD FILES
Data Files
Filters
Verified
To be verified
Plagiarism >= 70%
Plagiarism < 70%
Search
UUID
File Name
Upload Date
Verified
Plagiarism Rate
1
1
2021-12-30
100%
2
2
2021-12-30
40%
4
4
2021-12-30
100%
8cc9b46a-9eb8-4e81-a8cf-8964d6eafcef
plagiarism test text 3.txt
2021-12-30
7697ada0-12df-4a91-87a7-72124d56f344
plagiarism test text 2.txt
2021-12-30
e7b2aa17-d84b-4590-9f2b-cef493f34f84
plagiarism test text 1.txt
2021-12-30
CHECK SELECTED
Click to see file content


2
2021-12-30
40%
60%
100%
Content of "plagiarism test text 4.txt"
This is the text of the fourth file
CLOSE
plagiarism test text 6.txt
2021-12-30

4. Описание на особеностите на създадения програмен код

Проектът представлява УЕБ приложение изработено чрез следните технологии:
Backend - Oracle Database + Liquibase, Kotlin with Spring boot + JOOQ
REST api - spring rest
Frontend - Vue.js, Quasar, Vite, Axious, TS

Кодът за алгоритъма е написан на езика Java, и са използвани част от удобствата на езика. Като за начало при двата мутиращи низовете алгоритъма, се използват stream, поради удобството от декларативния подход който предлагатmap и filter функциите, които притежава.



Също така както се вижда в приложената снимка се използват и предоставения от езика метод за минаване от главна в малка буква toLower. За да бъдат удобни за използване първия алгоритъм връща списъка от думи в двата низа, които да могат лесно сравнявани със списъка с често срещани думи и да бъдат лесно премахнати, когато

това се налага. В крайна сметка обаче резултата от двата метода е символен низ

5. Примерни резултати от работата на създаденото приложение

В програмния код има създадени тестове за приложения алгоритъм:

Тест 1:
“He went to the kitchen”
“He went to the kitchen”
=> 100% - двата низа са равни тоест има пълно плагиатство

Тест 2:
“He went to the kitchen”
"The man was in the kitchen last night"
=> 48% - тук повторение има само на думата kitchen, тъй като думите to, the, in и was се премахват като често срещани думи. Тоест низовете които се сравняват са “He went kitchen”, “Man kitchen last night”.

Тест 3:
“hello”
“”

=> 0% няма плагиатство защото единия низ е празен.

Тест 4:
“to kitchen”
“kitchen”

=>70% плагиатство, думата to се премахва, но празното място остава, което означава че трябва да се премахне за да има еднаквост на низовете.

6. Оценка на ефективността на реализацията.

Алгоритъма на левенщайн използва два вложени цикъла съответно с n и m на брой итерации (дължините на двата низа), което дава сложност по време O(n*m) (или O(nm)). Освен това помощните алгоритми изискват обхождания на двата масива следователно двата алгоритъма имат сложност O(n + m), тоест общата оценка на алгоритъма по време е O(nm + 2(n+m)), което може да се сведе до O(nm). Сложността на алгоритъма по памет отново е O(nm), тъй като имаме матрица с n реда и m колони.

Възможни подобрения: Подобрение към алгоритъма би било начин по който да се хваща не само пряко

плагиатство, а и промяна на словоред, когато смисъла при размяна на думи не би променило смисъла на изречението. Също така алгоритъма използва изключително доста памет и при дълги низове би бил доста неефективен, което може да се оптимизира чрез промяна на матрицата в друга структура от данни като списък от списъци които да държат минималния брой стъпки във всеки момент.

7. Инструкции за компилация.

За да може да отворите и компилирате успешно проекта, трябва да имате инсталирана Java (версия 13+), Node.js, Vite, Oracle 11.2 Database, IntelliJ IDEA (може и друго, но в рамките на тези инструкции ще използваме IntelliJ).

1. Трябва да си създадете User в Оракъл базата, в който като се пусне проектът автоматично ще създаде необходимите таблици. За целта отворете Consola и напишете следните команди:
    - sqlplus / as sysdba
    - create user PLAGIARISM identified by plagiarism;
    - grant all privileges to PLAGIARISM;
2. Отворете проекта с IDEA-то и изчакайте да се индексира.
3. По подразбиране трябва да има 2 Run- конфигурации:
    - PlagiarismBackendApplication (backend)
    - Dev (frontend)

Пуснете ги и двата и в адрес localhost:3000 ще намерите приложението.

Ако няма Run конфигурации, намерете PlagiarismBackendApplication.kt (намира се в бекенд модула, котлин пакета.) и пуснете Spring main функцията. След това отворете терминал и чрез cd / влезте в папката на фронт енда, където трябва да напишете “npm run dev”, което ще стартира УЕБ приложението.

Използвана литература:

1. Levenshtein distance - Wikipedia - 22.12
2. NLP: Tokenization, Stemming, Lemmatization and Part of Speech Tagging | by Kerem Kargın | MLearning.ai | Medium - 23.12
3. Microsoft Word - 006-0018(2018).doc (iaras.org) - 18.12
4. https://spring.io/projects/spring-boot
5. https://quasar.dev/
6. https://vuejs.org/v2/guide/
7. https://www.jooq.org/doc/3.15/manual/
8. https://docs.liquibase.com/home.html
9. https://kotlinlang.org/docs/home.html