**String Plagiarism Ckecker**

**Using KMP Algorithm**

**Aim of the Project 🡪**

This project was aimed at designing a system that will be able to detect the possible act of plagiarism By referencing a source file as our dataset and if the program finds a match with the input currently passing by then it detects plagiarism and notes it. We used one existing classical string matching algorithm KMP.

**Installation and Flow of the project** 🡪

Installation for Vs-Code 🡪

1. Create a JAVA project in Code
2. Copy entire pool of documents in a folder named "DataSet\_SourceFiles", and include the path of this directory in MClass.java on line no 34, example:---> final File folder = new File("j:\DataSet\_SourceFiles");
3. Include the test file on the line no 59 of the MClass.java, example:---> File inputFile = new File( "input.txt");

**Flow of the project** 🡪

1. The source code has many java classes which are:--> MClass.java, KMP.java a) MClass.java : Main java file. b) KMP.java : Has the KMP algorithm implementation.
2. Please run the MClass.java file, it will take the input file and test for plagiarism with respect to the files in the document pool.
3. The result of the plagiarism is shown on the console, at the same time the results are also written back on the result files:--> kmp.txt
4. The results written on the files are dependent on the threshold that we choose for the similarity for plagiarism, we have used the similarity threshold to be 60( /100 for double type) on line no 93 of the MClass.java, please change that according to your choice.

**Source Code of the project** **🡪**

**KMP.java🡪**

package stringPlagarism.aglorthim;

public class KMP {

/\*\*

\* Pre processes the pattern array based on proper prefixes and proper

\* suffixes at every position of the array

\*

\* @param ptrn

\* word that is to be searched in the search string

\* @return partial match table which indicates

\*/

public int searchTimeCount= 0;

public int preprocessingTimeCount= 0, timeCount=0;

public int[] preProcessPattern(char[] ptrn) {

int i = 0, j = -1;

int ptrnLen = ptrn.length;

int[] b = new int[ptrnLen + 1];

b[i] = j;

while (i < ptrnLen) {

while (j >= 0 && ptrn[i] != ptrn[j]) {

j = b[j];

preprocessingTimeCount++;

}

i++;

j++;

b[i] = j;

}

// print pettern, partial match table and index

/\*System.out.println("printing pattern, partial match table, and its index");

System.out.print(" ");\*/

for (char c : ptrn) {

//System.out.print(c + " ");

}

// System.out.println(" ");

for (int tmp : b) {

//System.out.print(tmp + " ");

}

// System.out.print("\n ");

for (int l = 0; l < ptrn.length; l++) {

//System.out.print(l + " ");

}

// System.out.println();

return b;

}

/\*\*

\* Based on the pre processed array, search for the pattern in the text

\*

\* @param text

\* text over which search happens

\* @param ptrn

\* pattern that is to be searched

\*/

public int searchSubString(String text, String ptrn) {

int i = 0, j = 0;

// pattern and text lengths

String patern = ptrn;

String txt = text;

char[] patrn = patern.toCharArray();

char[] t = txt.toCharArray();

int ptrnLen = patrn.length;

int txtLen = t.length;

int matchCounter;

int maxSubstringLength=0;

// initialize new array and preprocess the pattern

int[] b = preProcessPattern(patrn);

while (i < txtLen) {

while (j >= 0 && t[i] != patrn[j]) {

/\* System.out.println("Mismatch happened, between text char "

+ t[i] + " and pattern char " + patrn[j]

+ ", \nhence jumping the value of " + "j from " + j

+ " to " + b[j] + " at text index i at " + i

+ " based on partial match table");\*/

j = b[j];

searchTimeCount++;

}

i++;

j++;

maxSubstringLength = Math.max(maxSubstringLength, j);

//System.out.println("The value of J is "+ j);

// a match is found

if (j == ptrnLen) {

/\*System.out.println("FOUND SUBSTRING AT i " + i + " and index:"

+ (i - ptrnLen));

System.out.println("Setting j from " + j + " to " + b[j]);\*/

j = b[j];

}

}

//System.out.println("The Length of Max Substring Length is " + maxSubstringLength);

return maxSubstringLength;

}

public int getTimeCount(){

if(searchTimeCount>preprocessingTimeCount)

{ timeCount = searchTimeCount;

}

else

{

timeCount = preprocessingTimeCount;

}

return timeCount;

}

// only for test purposes

public static void main(String[] args) {

KMP stm = new KMP();

// pattern

String patern,text;

text ="Sachin";

patern = "Sach";

stm.searchSubString(text, patern);

}

}

**Mclass.java** 🡪

package stringPlagarism.aglorthim;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

public class MClass {

/\*\*

\* @param args

\* @throws IOException

\*/

public static void main(String[] args) throws IOException {

String ptrnLine, textLine,inpLine,sFilePath,srcLine;

String input;

int srcLineIndex=1, inpLineIndex=1;

KMP kmpComponent;

int inputLen,srcLen,patterntextLength;

double kmpSimRatio = 0;

int fullTextLength=0, fullPatternLength=0;

int mainTimeCount= 0,kmpComponentCount=0;

//Source folder for the genuine files (pool)

final File folder = new File("J:\\StringSet");

File fileKmp = new File("kmp.txt"); // kmp result to be written in kmp.txt

int coun = 0;

// old result files deleted for the new result files

fileKmp.delete();

FileWriter outKmpFile = new FileWriter("kmp.txt", true);

// reading the files from the pool for the plagiarism

for (final File fileEntry : folder.listFiles()) {

sFilePath = fileEntry.getPath();

srcLineIndex=1;

File sourceFile = new File(sFilePath);

File inputFile = new File( "input.txt"); // the source file to be examined for plagiarism

@SuppressWarnings("resource")

BufferedReader sReader = new BufferedReader( new FileReader(sourceFile));

while((srcLine = sReader.readLine())!=null){

BufferedReader reader = new BufferedReader( new FileReader(inputFile));

inpLineIndex=1;

fullTextLength = fullTextLength+srcLine.length();

while((inpLine = reader.readLine())!=null){

inputLen = inpLine.length();

srcLen = srcLine.length();

if(inputLen>0 && srcLen>0) // msking sure that the strings to be compared have non-zero length

{

if(srcLen>inputLen)

{ textLine = srcLine;

ptrnLine = inpLine;

}

else

{ textLine = inpLine;

ptrnLine = srcLine;

}

patterntextLength = ptrnLine.length();

//fullTextLength = fullTextLength+textLine.length();

if(coun<1)

{

fullPatternLength = fullPatternLength+ ptrnLine.length();

}

//Running KMP component

kmpComponent = new KMP();

if(patterntextLength!=0)

{ kmpSimRatio= (kmpComponent.searchSubString(textLine, ptrnLine)/(double)(patterntextLength));

}

System.out.println("KMP Result");

System.out.println("Similarity ratio = "+kmpSimRatio\*100.000+" Line Number of the input file= "+inpLineIndex+

" Line Number of the source file "+fileEntry.getName()+ " = "+srcLineIndex);

System.out.println("------------------------------------------------------------------------------------------------------------------------------------------");

PrintWriter outPKmpFile = new PrintWriter(outKmpFile);

if(kmpSimRatio>0.60)

{ outPKmpFile.append("Line "+inpLineIndex + " of the input file has plagarised " +kmpSimRatio\*100.000+

"% from line "+srcLineIndex +" of the source file "+fileEntry.getName()+"\n");

}

//Computing KMP Time Count

kmpComponentCount = kmpComponentCount+kmpComponent.getTimeCount();

System.out.println("The amount of time KMP took for a text lenghth "+textLine.length()+" and pattern length " +ptrnLine.length()+" is "+

kmpComponentCount);

mainTimeCount = kmpComponentCount+mainTimeCount;

}

inpLineIndex++;

}

}

coun++;

srcLineIndex++;

}

mainTimeCount = kmpComponentCount;

System.out.println("The amount of time KMP took for a text lenghth "+fullTextLength+" and pattern length " +fullPatternLength+" is "+

kmpComponentCount);

System.out.println("The amount of total time took for full Text length "+fullTextLength+" and full pattern length " +fullPatternLength+" is "+

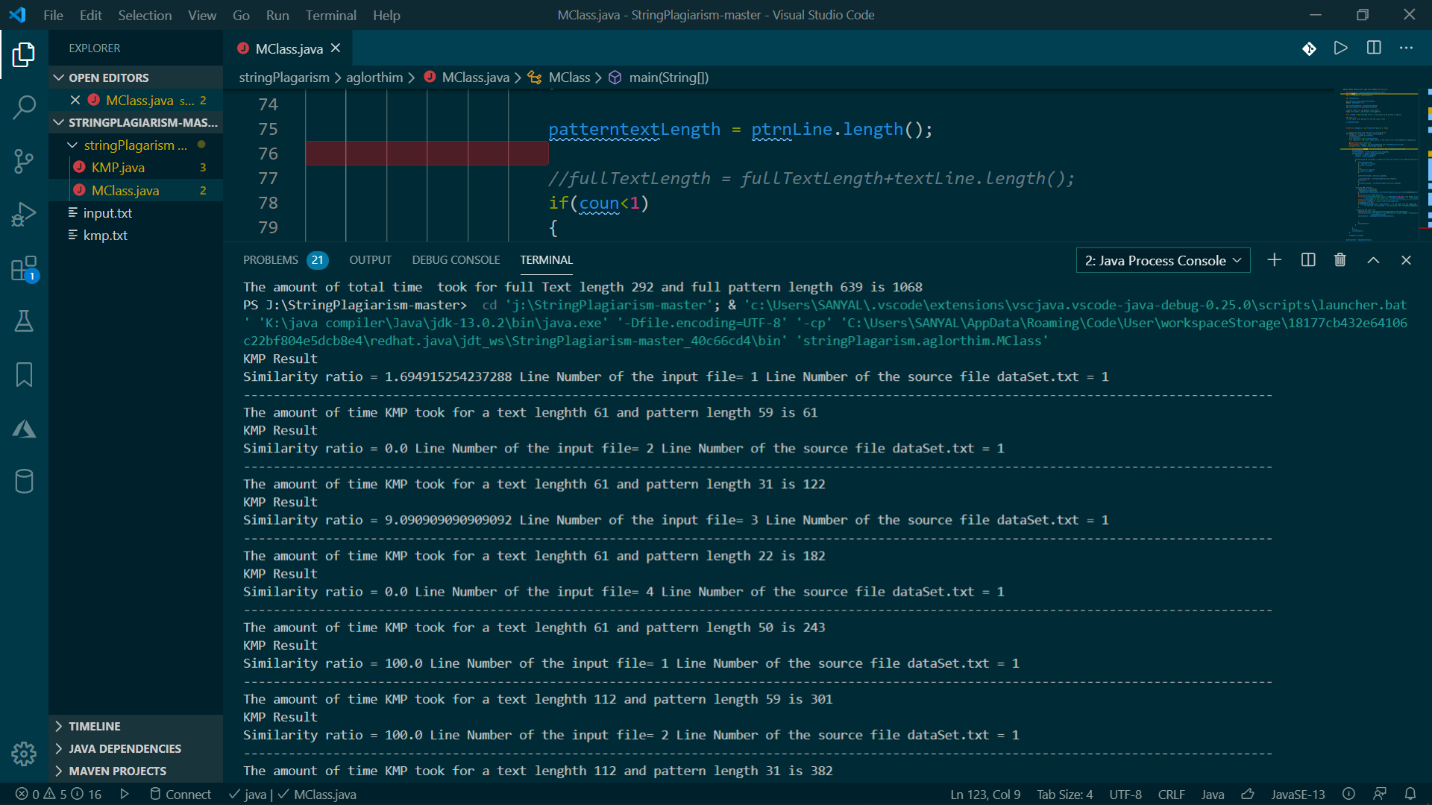
mainTimeCount);

outKmpFile.close();

}

}

**Output** 🡪 SysOut mode :-



File Append mode :-

# 

|  |
| --- |
| **Project Member**   * 1. Subhajit Sanyal(116)   2. Ranojit Datta(135)   3. Suman Chakraborty(122)   4. Jeet Hazra(132)   5. Sourav Hazra(110) |