CSCE 1101
Section 1
Dr. Howaida Ismaeel

Jana Elmidany - 900226145
Rana Taher - 900221430
Haya Shalaby - 900222031
Final Project Report

Simple Plagiarism Detection Utility using String Matching

CSCE 1101-01, 02, 05 Spring 2023 Term Project Report
Department of Computer Science and Engineering, AUC

Keywords:
String-matching, Brute Force, Rabin Karp, Hamming Distance, QT, Plagiarism detection

---

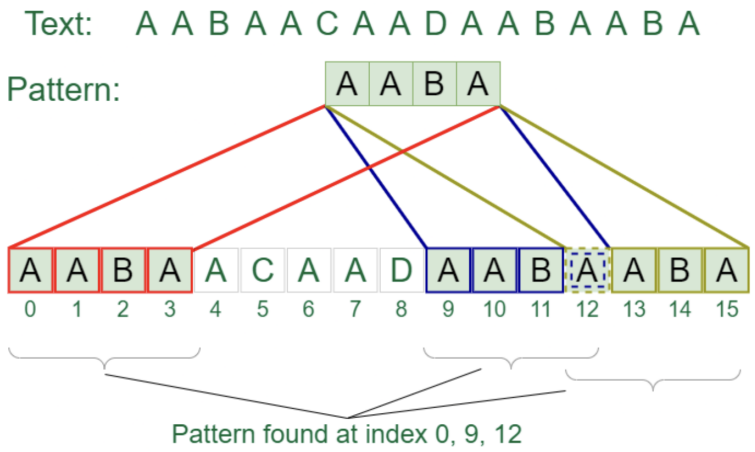
1. Introduction
This project uses string matching algorithms Brute Force (Hamming Distance) and Rabin Karp, and their sliding window approach to identify similarity and, in turn, plagiarism. This is done by comparing a potentially plagiarized file and original files from a database.

2. Topic Definition
This report aims to examine both string-matching algorithms and compare their effectiveness. This will be done by comparing their accuracy and runtime. It will also explain the code and identify any shortcomings and limitations found in the program.

3. Methodology
Two classes were created: Document class, and PlagiarismChecker class. A PlagiarismChecker object receives the file names of the potentially plagiarized file, and the database files the user wants to check against. Creating the object calls the Document constructor which opens all these files individually and stores their lines in individual vectors. The PlagiarismChecker constructor calls all the core functions of the program which use the aforementioned algorithms to detect areas of similarity, references, quotes, and plagiarism, and increments their assigned variables accordingly.

4. Specification of Algorithms to be used
The user is given a choice between the two algorithms which are Brute Force (Hamming Distance) and Rabin Karp. Brute Force uses a function of hamming distance that takes two strings as parameters: one is the line from the user document that is checked for plagiarism and the other is a line from the database checked against. This function will count the number of

differences between the user sentence and the database sentence. Then, it will compare the differences to a threshold that we set as half of the length of the shorter line, be it the line from the user document or the line from one of the documents in the comparison database. If the differences in the characters in the lines are less than or equal to the threshold then the sentence is plagiarized, otherwise, the sentence is not plagiarized. For Rabin Karp, the sliding window mechanism is used to create a pattern, and it looks for the hash value of the pattern made from the user line in the line obtained from one of the documents from the database. We implemented the sliding window mechanisms using the help of vectors. First, we separated each line from the user document into a vector of words. Then this vector is used to create a vector of all the possible patterns that a sentence can have. Afterward, in a loop, each pattern is tested against each line in the database document. If the pattern matches any of the hash values of any segment of the lines in the database then the whole line is plagiarized, otherwise it is not.



5. Data Specifications

Three different cases need to be tested. First, we input a document to check for plagiarism that is the exact same as one of the documents in the database we are using for comparison, to show all the lines in said document are plagiarized. The second case is comparing a document to a set of documents that are completely irrelevant to it, to show that the number of lines plagiarized is supposed to be zero. The third case is when the document we are using for testing consists of some sentences from some of the documents that are used for comparison. It is supposed to show that not all of the document is plagiarized but some lines from the document are plagiarized.

6. Experimental Results

We first started by using only two files: one user file and one comparison file. We tested them being the exact same, the user file containing quotes or references, or the user file having only some copied sentences from the comparison file. When we ensured that all of these cases worked properly, we started adding and testing with more comparison files. In the terminal, we got

accurate numbers; however, when we adapted the program to Qt, the output given was inaccurate.

7. Analysis and Critique

The code we produced had several issues. The first of these is that when we are checking for the quotation in the code, it will only count the quotation if there it has one sentence. If there are two or more, it will not count them. Additionally, in-text citations affect the program's ability to detect quotes. Furthermore, spaces can cause issues when we are comparing the sentence from user and database documents. If the spacing is different, it might not be able to detect plagiarism as the lines are not aligned. Another issue we faced in the code is that if the word references or works cited were mentioned before the actual references or works cited page then it might count all the sentences after as references instead of checking them for plagiarism.

8. Conclusions

Our program is able to detect and calculate the amount of similarity, which includes references and quotes, as well as the amount of plagiarism within the user's document. After analyzing our input and code continuously, we concluded that both algorithms harbor limitations that affect their accuracy. The hamming distance method used with the brute force algorithm depends on differences which could often be misleading since plagiarism could occur on a small scale that would not be detected by this algorithm. The Rabin Karp, on the other hand, is more accurate as it actually searches for a match using hash values rather than estimating the difference between files and depending on that to reach plagiarism and similarity statistics. Apart from algorithm limitations, our own code also had a considerable amount as previously discussed. Overall, there are still many more aspects a plagiarism detection program has the potential to cover that are even more advanced such as linguistic and logical plagiarism and similarity.

Acknowledgments

Special thanks to Ahmad Abdallah for his advice and help with debugging.

References

Rabin-Karp Algorithm for Pattern Searching. (n.d.). GeeksforGeeks. Retrieved May 14, 2023, from https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/

Opening and reading a file in Qt. (2017, June 23). Stack Overflow. https://stackoverflow.com/questions/44790470/opening-and-reading-a-file-in-qt

The Coding Train. (2019, December 18). Coding Challenge #156: Lissajous Curve Table [Video]. YouTube. https://www.youtube.com/watch?v=HS_2x8ozyAo

Appendix: Listing of all Implementation Codes

```
// Jana Elmidany, Haya Shalaby, Rana Taher
#ifndef _DOCUMENT_H
#define _DOCUMENT_H
#include <fstream>
#include <string>
#include <vector>
using namespace std;

class Document {
private:
  string fileName;
  vector<string> lines;
  int numOflines;

public:
  Document();
  Document(string); // constructor, opens file FN, stores lines in vector lines
  void setFileName(string);  // Sets file name
  void setLines(string);     // Sets lines
  void addLine(string);      // Adds lines
  int numOfLines();          // Returns number of lines
  string getFileName();      // getter for fileName
  vector<string> getLines(); // getter for vector lines
  //int getNumWords();         // returns no. of words in the document
  //int countWords(string);    // Counts number of words in a sentence
};

#endif //_DOCUMENT_H
```

```cpp
#include "Document.h"
#include <vector>
#include <iostream>
using namespace std;

Document::Document() {
  fileName = "";
  numOflines = 0;
}

Document::Document(string FN) {
  setFileName(FN);
  setLines(FN);
}

void Document::setFileName(string FN) { fileName = FN; }

void Document::setLines(string FN) {
  fstream file;
  file.open(FN);
  string temp;
  while (!file.eof()) {
    getline(file, temp, '.');
    lines.push_back(temp);
  }
  //cout << lines.size()-1 << endl;
}

void Document::addLine(string sentence) {
  lines.push_back(sentence);

}

string Document::getFileName() { return fileName; }

vector<string> Document::getLines() { return lines; }

int Document::numOfLines()
{
    double L=lines.size();
    return L;

}

```

```h
#ifndef _PLAGIARISMCHECKER_H
#define _PLAGIARISMCHECKER_H

#include <string>
#include <fstream>
#include <iostream>
#include <vector>
#include "Document.h"
using namespace std;

class PlagiarismChecker {
    private:
        int totalPlag;
        int totalRef;
        int totalSim;
        int totalQuotes;
        Document userDoc;
        vector<Document> database;
        vector<int> quoteIndex;
        Document highlightDoc;
        bool match ;
        char type;
    public:
        PlagiarismChecker();
        PlagiarismChecker(string userFile); //opens userFile and ogFile
        bool isQuote(string line); //checks if similar line is a quotation & increments totalQuotes if true
        bool isRef(string sent);  //looks for reference page & adds count of all words starting there to totalRef
        void calcPlag(); //calculates totalPlag by subtracting totalRef and totalQuotes from totalSim
        void compDoc(); //compares userDoc against database for similarity, produces highlighted document
        int getTotalPlag(); //getter for totalPlag
        int getTotalRef(); //getter for totalRef
        int getTotalSim(); //getter for totalSim
        int getTotalQuotes(); //getter for totalQuotes
        Document getHighlightDoc(); //getter for highlightDoc
        void createCorpus();//enters the files into the vector
        void compLines(vector <string> data , vector <string> user);
        vector <string> getPat(string sentence);// gets the pattern of the sentence
        bool bruteforce(string pat, string txt,long threshold ); //brute force & rabin karp
       // int countWords(string sentence);
        int hammingDistance(string str1, string str2);
        bool rabinKarpSearch(string pattern, string text, int q);
        vector <string> createPat(vector <string> pats);
        void set_type(char ty);

```

```cpp
#include <string>
#include <vector>
#include <ctype.h>
#include "PlagiarismChecker.h"
#include "Document.h"
using namespace std;

// d is the number of characters in the input alphabet //for stringMatching
#define d 256

PlagiarismChecker::PlagiarismChecker()
{
  userDoc=Document();

  totalPlag = 0;
  totalRef = 0;
  totalSim = 0;
  totalQuotes = 0;
    match=false;
}

PlagiarismChecker::PlagiarismChecker(string userFile)
{
  userDoc=Document(userFile);
  createCorpus();

}

void PlagiarismChecker::createCorpus()
{
  int size;
  string filename;
  Document corpus;

  cout<<"Enter the number of files you want to check from "<<endl;
  cin>>size;

  cout<<"Enter names of files"<<endl;

  for(int i=0; i<size ; i++)
    {
      cin>>filename;

      corpus=Document(filename+".txt");
      database.push_back(corpus);
    }
}

bool PlagiarismChecker::isQuote(string Line) {

    for(int i=0 ; i<Line.length() ; i++)
    {
        if(Line[i] == '"' )
        {
            for(int j=i ; j<Line.length() ; j++)
            {
                if(Line[j]=='"')
                    return true;
            }

        }
    }

  return false;
```

```
66
67  bool PlagiarismChecker::isRef(string sent)
68  {
69    if(sent.substr(0,10) == "References" || sent.substr(0,11) == "Works Cited" ) //or should it be sent == "References"
70    {
71       return true;
72    }
73    return false;
74  }
75
76  void PlagiarismChecker::calcPlag()
77  {
78    totalPlag = totalSim - (totalQuotes + totalRef);
79  }
80
81  void PlagiarismChecker::compDoc() // We will count the words for refrences and quotes here -rana
82  {
83      vector <string> dataLines,userLines;
84    userLines=userDoc.getLines();
85    for(int i=0 ; i<database.size() ; i++)
86    {
87      dataLines=database[i].getLines();
88      compLines(dataLines,userLines);
89    }
90      cout<<"The percentage of similarity is: "<<(static_cast<double>(totalQuotes+totalPlag+totalRef)/userLines.size())*1
91      cout<<"The percentage of plagiarism is: "<<(static_cast<double>(totalPlag)/userLines.size())*100<<"%"<<endl;
92      cout<<"The percentage of Quotes is: "<<(static_cast<double>(totalQuotes)/userLines.size())*100<<"%"<<endl;
93      cout<<"The percentage of References is: "<<(static_cast<double>(totalRef)/userLines.size())*100<<"%"<<endl;
94
95  }
96
97  vector <string> PlagiarismChecker::createPat(vector <string> patterns)
98  {
99      string temp;
100     int count;
101     vector <string> fullPats;
102
103     for(int s=0 ; s<fullPats.size()+1 ; s++)
104     {
105         temp="";
106         count=0;
107
108         while(count<5 && count<patterns.size() )
109         {
110             temp += ' '+patterns[count];
111             count++;
112         }
113         if(temp!=""){
114             fullPats.push_back(temp);}
115         else{
116             break;
117         }
118
119         patterns.erase(patterns.begin());
120         if(patterns.size()<5)
121             break;
122     }
123
124     return fullPats;
125 }
126
127 void PlagiarismChecker::set_type(char ty)
128 {
129     type=ty;
130 }
```

```
void PlagiarismChecker::compLines(vector <string> data , vector <string> user) // We will count the words for refrences
{
  long threshold;
    bool x;

    vector <string> patterns;// vector of words
    vector <string> fullPats;// vector of patterns

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


          x=isRef(user[j]);


          if(x==true && totalRef==0)
          {
            cout<<"The remaining is refreneces"<<endl;
              for(int in=j ; in<user.size() ; in++)
              {
                  totalRef ++;
                  //totalSim ++;
              }
              break;
          }
          else if(isQuote(user[j]) && totalQuotes==0)
            {
                totalQuotes++;
            }
          else
          {
              patterns=getPat(user[j]);
                if(patterns.size()<5)
                {
                    continue;
                }

                fullPats=createPat(patterns);

              bool rabin=false;
        if(type=='r')
        {
            for(int index=0 ; index<fullPats.size() ;index++ ){
                for(int k=0 ; k<data[k].size() ; k++)
                {

                    if(rabinKarpSearch(fullPats[index], data[k] , 97))
                    {
                        rabin=true;
                        totalPlag ++;
                        break;
                    }
                }

                if(rabin)
                    break;
                }
        }
        else{
```

```
            for(int k=0 ; k<data[k].size() ; k++)
            {

                if(data[k].length() > user[j].length())
                    threshold=user[j].length()/2;
                else
                    threshold=data[j].length()/2;


                if(bruteforce(user[j],data[k],threshold))
                {
                    totalPlag ++;
                    break;
                }
            }
        }

          }
      }
}



bool PlagiarismChecker::bruteforce(string usertxt, string datatxt, long threshold )
{
    int distance=hammingDistance( usertxt, datatxt);
    return (distance <= threshold);
}

 vector <string> PlagiarismChecker::getPat(string sentence)
  {
     vector <string> pattern;
     string temp;

     for(int i=0 ; i<sentence.length();)
     {
          temp="";
         if(sentence[i]==' ')
         {
             i++;
             continue;
         }
         for(int j=i ; j<sentence.length() ; j++)
         {
             if (sentence[j] ==' ')
             {
                 break;
             }
             temp += sentence[j];
         }

         pattern.push_back(temp);
         i += temp.length();
     }

     return pattern;

  }
```

```
int PlagiarismChecker::hammingDistance(string str1, string str2)
{
    int distance = 0;
    for (int i = 0; i < str1.length(); i++) {
        if (str1[i] != str2[i]) {
            distance++;
        }
    }
    return distance;
}

bool PlagiarismChecker::rabinKarpSearch(string pat, string txt , int q)
{
        long M = pat.length();
        long N = txt.length();
        int i, j;
        int p = 0; // hash value for pattern
        int t = 0; // hash value for txt
        int h = 1;

        // The value of h would be "pow(d, M-1)%q"
        for (i = 0; i < M - 1; i++)
            h = (h * d) % q;

        // Calculate the hash value of pattern and first
        // window of text
        for (i = 0; i < M; i++) {
            p = (d * p + pat[i]) % q;
            t = (d * t + txt[i]) % q;
        }

        // Slide the pattern over text one by one
        for (i = 0; i <= N - M; i++) {

            // Check the hash values of current window of text
            // and pattern. If the hash values match then only
            // check for characters one by one
            if (p == t) {
                /* Check for characters one by one */
                for (j = 0; j < M; j++) {
                    if (txt[i + j] != pat[j]) {
                        break;
                    }
                }

                // if p == t and pat[0...M-1] = txt[i, i+1,
                // ...i+M-1]

                if (j == M)
                {
                    cout << "Pattern found at index " << i
                         << endl;
                   return true;
                }
            }

            // Calculate hash value for next window of text:
            // Remove leading digit, add trailing digit
            if (i < N - M) {
                t = (d * (t - txt[i] * h) + txt[i + M]) % q;

                // We might get negative value of t, converting
                // it to positive
                if (t < 0)
                    t = (t + q);
            }
```

```
305
306             // Calculate hash value for next window of text:
307             // Remove leading digit, add trailing digit
308             if (i < N - M) {
309                 t = (d * (t - txt[i] * h) + txt[i + M]) % q;
310
311                 // We might get negative value of t, converting
312                 // it to positive
313                 if (t < 0)
314                     t = (t + q);
315             }
316         }
317     return false;
318 }
319
320
321 int PlagiarismChecker::getTotalPlag()
322 {
323   return totalPlag;
324 }
325
326 int PlagiarismChecker::getTotalRef()
327 {
328   return totalRef;
329 }
330
331 int PlagiarismChecker::getTotalSim()
332 {
333   return totalSim;
334 }
335
336 int PlagiarismChecker::getTotalQuotes()
337 {
338   return totalQuotes;
339 }
340
341
342
343
```