

Pattern Matching for Detecting Plagiarism on Artworks

Azalea Fisitania 13511028¹

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

¹13511028@std.stei.itb.ac.id

Abstract—Plagiarism has been a serious concern, not only in Indonesia but everywhere, especially developed countries and developing countries. Plagiarism happens not only on literatures, papers, or books but also on artworks such as drawings, paintings, photographs. This paper presents a way for plagiarism detection on artworks using pattern matching. One of pattern matching technique, Knuth-Morris-Pratt algorithm (KMP), will be used in the program implementation. At the end of the paper, it will be concluded that pattern matching can be used to detect plagiarism with some simplification and constraints due to limitation of program.

Index Terms— artworks, Knuth-Morris-Pratt algorithm, pattern matching, plagiarism, plagiarism detection.

I. INTRODUCTION

Plagiarism, *n.* means the appropriation or imitation of another's ideas and manner of expressing them, as in art, literature, etc., to be passed off as one's own. Plagiarize, *v.* means to commit plagiarism. Plagiarist, *n.* is one who plagiarizes ^[1]. Term "plagiarize" is defined as to take (ideas, documents, code, images, etc.) from another and pass the off as one's own *without proper citation* ^[2].

Plagiarism happens in a lot of forms. On school, plagiarism is simply copying whether only some parts from another student's work or even the whole work. It is inconvenience; plagiarists will gain undeserved advantage such as higher scores for their assignments with less effort. It actually is unfortunate for plagiarists because their brains won't be stimulated to think and less developed than those who do assignments on their own. Still, it can be a demotivating factor for those who do assignments because it's unfair for them. On college, it happens on student assignments, papers, etc.

Images plagiarism also happens on artworks such as drawings, paintings, photographs, etc. Images plagiarism increases as image manipulation technology, ease of distribution, and publication via Internet advances. On this Internet Age, it's very easy to share through upload and download images on social media (e.g. Facebook, Twitter, Multiply) or blog posts (e.g. Tumblr, Wordpress, Blogspot). Licenses and watermarks have been applied on works published into the Internet. Creative Commons (CC) is a non-profit organization that provides free license one works. The license differs considering whether the author gives permission on adaptations and/or

commercial uses of the work or not. For people who respect the original artist, it's useful. For those who don't, it remains as formality. On the other side, websites for publishing artworks e.g. deviantART, iStockPhoto, and ClipartOf provide automatic watermarks on the whole image. It takes a lot of time to remove all the watermarks. Safer, but it detracts the beauty of the artwork itself so a lot of artists make their own watermarks manually. They usually are small and easy to be erased using image applications e.g. Adobe Photoshop thus easier to be plagiarized. Not to mention careless artists who don't aware on plagiarism and don't apply any watermarks.

It's hard to prevent plagiarism, but still can be detected after the occurrence; expecting that the plagiarism can be reported to the original artist and that the plagiarist can be forced to erase the image after it is confirmed by the original author. Detecting image plagiarism simply is done by matching two images: the original one and the plagiarized-suspected one. In the term of algorithm strategy, it's called pattern matching. In this paper, there will be discussed pattern matching for detecting image plagiarism.

Actually there are already some Internet-apps/plugins and stand-alone programs that can detect plagiarism on documents using the same concept: pattern matching. Nowadays, known an Internet plugin called TinEye plugin that able to search images in Internet. Currently, users of TinEye said that it still does exact match, so it isn't optimal in performing plagiarism detection.

II. PROBLEM DEFINITION

There is already term for matching images, called image matching. It involves image processing, which is a complicated mechanism. Thus, the image type here is limited into ASCII art which is the most universal computer art form in the world. ASCII art is any sort of pictures or diagrams drawn with the printable characters in the ASCII character set ^[3]. Because ASCII takes form as multi-line containing text using ASCII characters, it can be approached by pattern matching.

There is no explicit classification in ASCII art, but can be distinguished by its complexity as the two ASCII art images on Fig. 1 and Fig. 2 below:

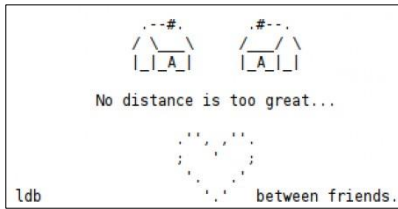


Image Source | <http://asciitartist.com/>
Fig. 1 A Simple ASCII Art

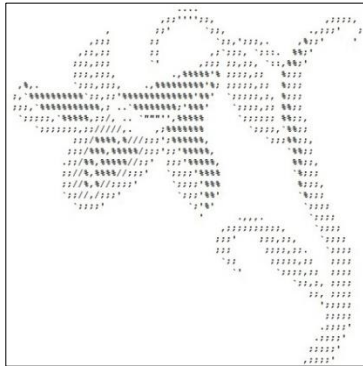


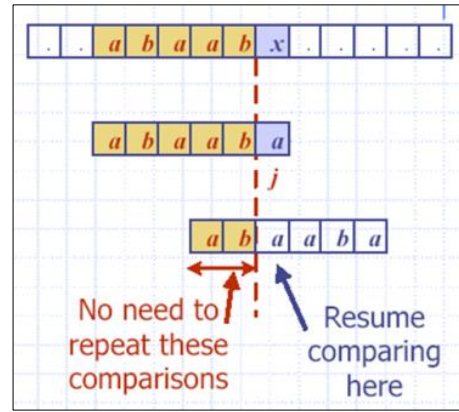
Image Source | <http://www.collectorsquest.com/>
Fig. 2 A More Complicated ASCII Art

Fig. 1 is simpler since it has fewer lines and uses fewer ASCII character types. Fig. 2 is more complicated as it takes a lot of lines and uses a lot of ASCII character types. The difference will take effect on pattern matching performance.

III. PATTERN MATCHING ALGORITHM

Given a text string T with n characters long and a pattern string P with m characters long (assuming $m < n$). Pattern matching is finding the pattern inside the text. There are a lot of pattern matching algorithm, such as Brute Force, Knuth-Morris-Pratt, and Boyer-Moore. The algorithm used in this paper is Knuth-Morris-Pratt (KMP).

KMP algorithm is discovered by Donald E. Knuth (born January 10, 1938), a computer scientist and Professor Emeritus at Stanford University. The technique is to look for the pattern in the text in a left-to-right order. Just like the brute force algorithm, but the shifting in KMP is more intelligently than the brute force algorithm. If a mismatch occurs between the text T and pattern P at P[j], the most shifting can be done to the pattern to avoid wasteful comparisons is the largest prefix of P[1 .. j-1] that is a suffix of P[1 .. j-1] [4].



Source | Dr. Andrew Davison, Pattern Matching Slide
Fig. 3 Illustration of KMP Shifting Technique

KMP preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself. As seen on Fig. 3, given text T "...abaabx..." and pattern P "abaaba". String "ab" is a suffix of matched string between pattern and text which is also a prefix of the original pattern. Suppose:

j = mismatch position in P[]

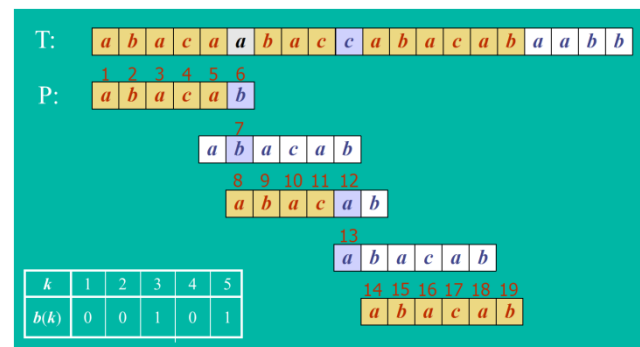
k = position before the mismatch ($k = j-1$)

The *border function*, or also called *failure function*, $b(k)$ is defined as the *size* of the largest prefix of P[1..k] that is also a suffix of P[1..k] [4]. If a mismatch occurs at P[j] (i.e. $P[j] \neq T[i]$), then

$k = j-1$;

$j = b(k) + 1$; //obtain the new j [4]

To clarify the KMP concept, given an example such there are string T "abacaabaccabacaabb" and pattern P "abacab", find if there is any match with P in T. The illustration of KMP algorithm execution is shown below in Fig. 4:



Source | Dr. Andrew Davison, Pattern Matching Slide
Fig. 4 Illustration of Pattern Matching using KMP

The mismatch occurs on P[j] where $j = 6$. The p' is now "abaca". Suffixes from p' are {"a", "ca", "aca", "baca"} while prefixes from P are {"a", "ab", "aba", "abac", "abaca"}. Thus, the longest suffix on p' which is also the longest prefix on P is "a" then shift the pattern as much as:

$\text{length}(p') - \text{length}("a") = 5 - 1 = 4$.

The matching and shifting is performed until exact match is found.

KMP has some advantages, such as:

- KMP runs in optimal time: $O(m+n)$, which is very fast
- KMP never needs to move backwards in the input text. This makes KMP good for processing very large files that are read in from external devices or through a network stream.

Besides the advantages, KMP also has disadvantages, such as:

- KMP doesn't work so well as the size of the alphabet increases because more possible mismatches. KMP is only faster when the mismatches occur later, not from early.
- Basic KMP try to re-match the letter in the text that caused the mismatch. For this problem, there is already KMP-extension that skips the mismatch letter. In other words, the shifting is incremented by 1. ^[4]

VI. PROBLEM SOLVING ANALYSIS WITH KMP ALGORITHM

As said on the introduction, content of ASCII art image is only ASCII characters, so simple pattern matching can be done on ASCII arts. Below is the pseudo code for reading ASCII art file that will be processed by the KMP algorithm in the program:

```
// Data Member
Vector<String> textASCII;
Vector<String> patternASCII;

// Read ASCII from file
fillASCIIart(String filename) → Vector<String>
{
    create fileReader to read lines;
    Vector<String> containerASCII;
    String line;
    while (next_line isn't null) {
        line ← next_line;
        add line to containerASCII;
    }
    → containerASCII;
}
```

To perform KMP algorithm, some basic methods to do string process are needed; get suffix of a string, get prefix of a string, and also get the longest suffix of a string that is also the longest prefix of another string. The pseudo code for string process is written below:

```
// String Process
getSuffixes(String _p) → Vector<String>
{
    Vector<String> listSuff;
    for (int i=length(_p)-1; i>=1; i--) {
        String suffix ← "";
        for (int j=i; j<length(_p); j++) {
```

```
            suffix ← suffix + p'[j];
        }
        add suffix to listSuff;
    }
    → listSuff;
}

getPrefixes(String P) → Vector<String>
{
    Vector<String> listPref;
    for (int i=1; i<=length(P)-1; i++) {
        String suffix ← "";
        for (int j=0; j<i; j++) {
            prefix = prefix + P[j];
        }
        add prefix to listPref;
    }
    → listPref;
}

getLongestPrefSuff(String P, String _p) → String
{
    Vector<String> prefixes ← getPrefixes(P);
    Vector<String> suffixes ← getSufixes(_p);
    Vector<String> prefsuff;
    for (int i=0; i<size(suffixes); i++) {
        if(suffixes[i] == prefixes[i]) {
            add suffixes[i] to prefsuff;
        }
    }
    String longest ← "";
    for (int i=0; i<size(prefsuff); i++) {
        if (length(prefsuff[i])>length(longest)) {
            longest = prefsuff[i];
        }
    }
    → longest;
}
```

Then KMP algorithm can be performed. As a note, the pure KMP algorithm is implemented in `isMatchKMP`. Meanwhile, `isAllMatchKMP` is specific to the problem discussed in this paper; ASCII art image. Below is pseudo-code for evaluating the two ASCII art images:

```
// Knuth-Morris-Pratt Algorithm
isMatchKMP(String T, String P) → boolean
{
    int N ← length(T);
    int _n ← length(P);
    int PosisiP0diT ← 0;
    int i ← PosisiP0diT;
    int j ← 0;
    while (not end of T or P) {
        String _p ← "";
        while (not end AND P[j] == T[i]) {
            _p = _p + P[j];
            i++;
            j++;
        }
        if (not end AND P[j] != T[i]) { //mismatch
            if (length(_p)==0) {
                PosisiP0diT++;
                i ← PosisiP0diT;
                j ← 0;
            } else {
                String l ← getLongestPrefSuff(P, _p);
```

```

        int geser ← length(_p) - length(l);
        shift PosisiP0diT geser times;
        i ← PosisiP0diT;
        j ← 0;
    }
}
}
if(end of T but not end of P) {
    → false;
} else {
    → true;
}
}

isAllMatchKMP (Vector<String> patternASCII,
Vector<String> textASCII) → Boolean
{
    for (int i=0; i<size(textASCII); i++) {
        if(get empty line on patternASCII[i] or
textASCII[i]) {
            → false;
        } else {
            if (!isMatchKMP(textASCII[i],
patternASCII[i])) {
                → false;
            }
        }
    }
    → true;
}
}

```

```

// String Process
public Vector<String> getSuffices(String _p);

public Vector<String> getPrefixes(String P);

public String getLongestPrefSuff(String P,
String _p);

// Knuth-Morris-Pratt Algorithm
public boolean isMatchKMP(String T, String P);
public boolean isAllMatchKMP (Vector<String>
patternASCII, Vector<String> textASCII);
}

```

V. IMPLEMENTATION ON PROGRAM

A. Program Specification

The inputs of the program are two files consisting ASCII art image (any type of file extension is allowed, as long as it contains ASCII art) which are the original image as the pattern P) and the suspected one (as the text T). The output of the program is whether the suspected image is plagiarized one or not. Pattern matching on this program is exact matching; not applicable for plagiarized image which applies resizing, rotating, or cropping the original image.

The program processes both of the file by scanning each line and put them into array of array of string. Suppose line of text i where $i = 1, 2, 3...$ For each line i (array of string), do pattern matching between line i in the original image and line i in the suspected image. The loop ends after the last line i has evaluated by pattern matching.

Below are headers of the source code to provide an overview of the program execution plot:

```

public class PatternMatching2 {

    // Data Member
    public Vector<String> textASCII;
    public Vector<String> patternASCII;

    // Read ASCII from file
    public Vector<String> fillASCIIart (String
filename);

    public void printASCIIart (Vector<String>
containerASCII);
}

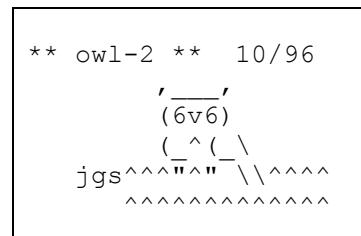
```

B. Program Testing

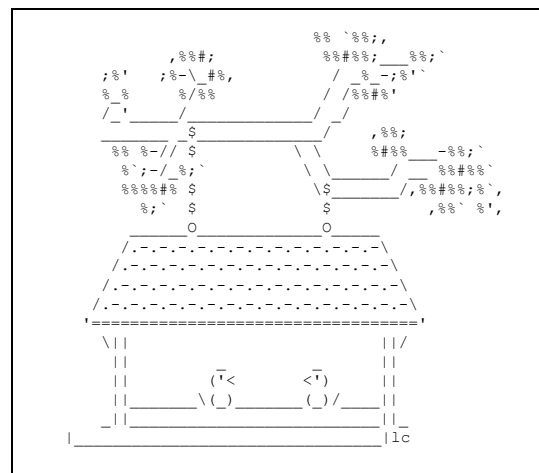
Program testing here is supposed to test whether the matching between two images succeeds or not. Also, it will be shown how line of ASCII art effects the algorithm performance.

1. Test Case 1

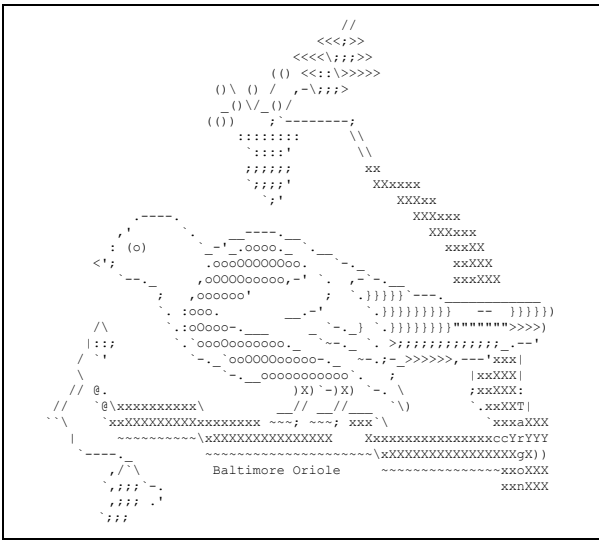
Here will be compared the algorithm performance according to ASCII art complexity. Images used as comparison are three ASCII art images below as seen on Fig. 5, Fig. 6, and Fig. 7. Each image has its plagiarized version which is the exact match with the original version. There will be three pattern matching processes done for each original ASCII art images with its plagiarized version. The comparison is based on execution time for each ASCII art image.



Source | <http://www.chris.com/ascii/>
 Fig. 5 Low Complexity ASCII Art, Total 6 Lines



Source | <http://www.chris.com/ascii/>
 Fig. 6 Medium Complexity ASCII Art, Total 23 Lines



Source | <http://www.chris.com/ascii/>
 Fig. 7 High Complexity ASCII Art, Total 32 Lines

Screenshots for pattern matching on each image are shown below on Fig. 8, Fig. 9, and Fig. 10:

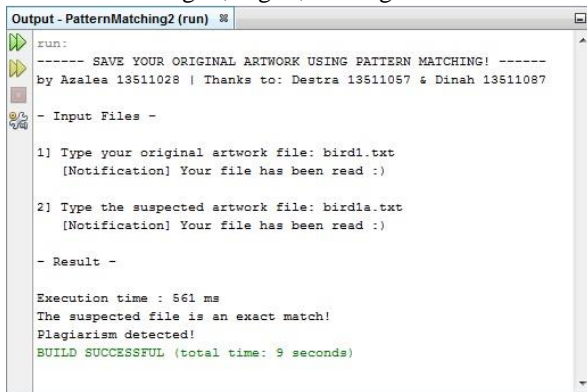


Fig. 8 Pattern Matching Result for Fig. 4

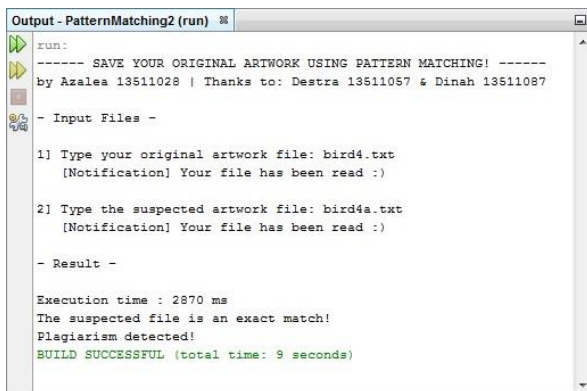


Fig. 9 Pattern Matching Result for Fig. 5

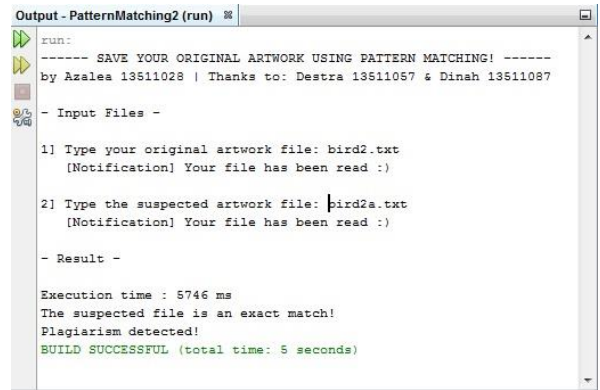


Fig. 10 Pattern Matching Result for Fig. 6

The result is shown on Fig. 11:

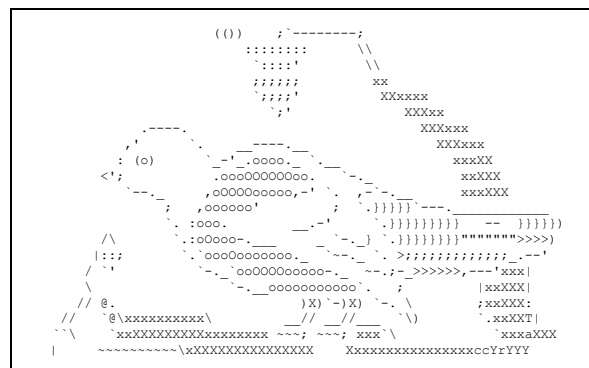
Test No.	Execution time (ms)		
	Fig. 4	Fig. 5	Fig. 6
1	561	2870	5746
2	720	3803	7134
3	678	3923	7167
Average result	653	3532	6682.33

Fig. 11 Table of Execution Time on Fig. 4, Fig. 5, and Fig. 6

As seen on the program results above, it is obvious that the execution time is longer as number of line in the ASCII art increases.

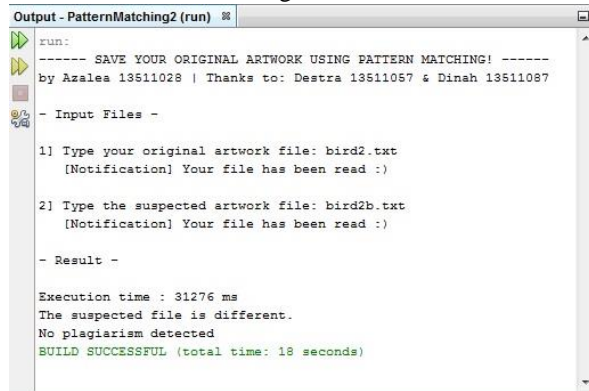
2. Test Case 2

Here will be proved that the program doesn't support non-exact match. Inputs used are the same ASCII art image as Fig. 6 (as the original version) and the plagiarized one which applied cropping on the original eliminating the artist's signature. As seen on Fig. 12, the bottom of the image has no more author's signature as in the original one.



Source | <http://www.chris.com/ascii/>
 Fig. 12 Plagiarized Version of Fig. 6

The result is shown on Fig. 13:



```
run:
----- SAVE YOUR ORIGINAL ARTWORK USING PATTERN MATCHING! -----
by Azalea 13511028 | Thanks to: Destra 13511057 & Dinah 13511087

- Input Files -

1) Type your original artwork file: bird2.txt
   [Notification] Your file has been read :)

2) Type the suspected artwork file: bird2b.txt
   [Notification] Your file has been read :)

- Result -

Execution time : 31276 ms
The suspected file is different.
No plagiarism detected
BUILD SUCCESSFUL (total time: 18 seconds)
```

Fig. 13 Pattern Matching Result for Fig. 12

As seen on the result above, it is seen that the program cannot detect the plagiarized image as plagiarism from the original image due to exact matching on pattern matching.

V. CONCLUSION

Pattern matching can be used to detect plagiarism on artworks, such as ASCII arts. ASCII art is made of ASCII characters so pattern matching can be done as easily as matching between two strings. The algorithm used to do the match is Knuth-Morris-Pratt Algorithm (KMP). The performance of the algorithm, however, depends on the complexity of the ASCII art image. Also, the definition of similarity between two ASCII art images is still an exact match.

VI. FUTURE POSSIBLE ACTIONS

The program is still very simple, but it can be improved more by doing these in the future:

1. Compare the pattern matching with Brute Force, Boyer-Moore, and other techniques to find out the most proper technique for matching ASCII art images.
2. Exact match isn't a good solution for detecting plagiarism. How if the image is cropped from the original one? The program algorithm (outside the pattern matching algorithm between two strings) needs to be modified. There are two ideas:
 - a. Use percentage. The higher the similarity percentage, the higher the probability the image is plagiarized from the original one.
 - b. Be flexible. The pattern can be the original image or the suspected image. If any match is found, there must be plagiarism.
3. How if the image is rotated from the original one? Rotate the original one and do the pattern matching again using the rotated version.
4. How if some text in the image is erased (e.g. the original artist's signature)? Use percentage as said before.

5. According to the program made, at least one ASCII art has to be the original one as a pattern between two ASCII arts. How if the original one isn't known? The idea is to check the date that the file was created. The smallest date is assumed to be the original.
6. As said in Chapter III, KMP is also good for processing very large files that are read through a network stream. The program may be deployed to the Internet to detect plagiarism on files distributed in the Internet just like TinEye plugin mentioned in Chapter I.

REFERENCES

- [1] 1991. The Macquarie Dictionary: The National Dictionary Second Edition. Macquarie University: The Macquarie Library. Page 1353
- [2] Bin-Habtoor, A. S. and M. A. Zaher. "A Survey on Plagiarism Detection Systems". International Journal of Computer Theory and Engineering Vol. 4, No. 2, April 2012
- [3] <http://www.ascii-art.de/ascii/faq.html>. Last accessed Wednesday, December 18, 2013 at 18:29
- [4] Munir, Rinaldi. 2009. Diktat Kuliah IF2211 Strategi Algoritma.

STATEMENT

I hereby state that this manuscript I wrote is my own writing; not an adaptation or translation from another manuscript, also not a plagiarism.

Bandung, December 20, 2013


Azalea Fisitania
13511028