Understanding Akismet using Pattern Matching Algorithms

Jeanne D'Arc Amara Hanieka (13519082)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519082@std.stei.itb.ac.id

***Abstract*—Since its development in the 1990s, blogs are an essential part of internet activities. It allows users from all kinds of backgrounds and age to share their opinions online, making it one of the most popular platform across the web. In its recent developments, the writers and the readers are able to interact with each other with the availability of the comment section. It is more often rather than not to see incomprehensible comments occupying the comment section of many blog posts. Automattic.inc, the company behind WordPress, tried to fight these spam comments with Akismet, a spam filtering service they created to support WordPress. This paper will try to give an insight on how Akismet works by modelling it using the pattern matching algorithms.**

***Keywords*—comments, spam, Akismet, pattern matching**

I. Introduction

The history of humans trying to publish something they worked on can be tracked as far as the prehistoric ages, where people simply draw on walls, as a way to convey a message to someone who will see it in the future. With the development of writing and image processing, humans started to try to share their ideas and opinions using numerous means like posters, books, newspapers, and so on. The idea of knowledge-sharing had brought countless milestones in the development of human being, and we can say that publication had shaped the world we are now living in.

With the emergence of the World Wide Web, people started to make creative use out of it by making it as a big idea board. This can be seen with the development of weblogs. Weblogs, commonly known as blogs, are first developed as a mere bulletin boards for companies. They are usually a part of a main website, containing threads of writing used for discussion and idea sharing. Since internet are getting more commercial in the late 90s, many people started to create their own blogs and 'write' on it. These 'writings' are not limited to texts only, but also pictures, videos, links, and so on.

The one feature that almost every blog-publishing owns is a comment section. The blog writer and the reader can freely interact on the comment section of a certain post or web by simply writing on the text box available on the web. It is an easy way for the writers to get suggestions and critics, as well as praises from their readers. However, with how easy people can comment on the comment section on blogs, it's also easy for irresponsible parties to fill those comment section with unwanted repeating messages or malicious links, called spam comments.

Image 1.1 Platforms that provides blog-publishing systems (source: https://www.raghwendra.com/blog/the-top-15-best-blogging-platforms-a-detailed-synopsis/)

One of the most commonly known content management system (CMS) is WordPress, a PHP-based blog-publishing system released in 2003 by Matt Wullenweg. They founded Automattic.inc two years later and went on making numerous projects. One of the projects is called Akismet, an anti-comment spam system that are able to work in various blogging platforms and forums. Their purpose is to block unwanted comments from being posted on the platform they're working on.

While Akismet used a more complicated algorithm in order to define whether a comment is a spam or not, it is possible to model the way Akismet works using the pattern matching algorithms.

II. Basic Theory

A. Akismet

Spam mails are first found in emails. When blogs started to have comment sections, irresponsible people started to take advantage of the open comment forms that allows anyone to post comments to put spam messages inside the comment section.

As one of the fastest growing blog-publishing system during that time, WordPress aimed to control the appearance of spam mails by making a JavaScript-based plugin in 2005. However, that did not went well, as the spammers succeeded in bypassing the plugin a few hours after its launch[1]. They decided to do a different approach, which is to make a plugin that receives crowd-sourced spam reports, called 'Automattic kismet', or Akismet in short. It could identify whether a comment is a spam or not by comparing it to the reports made beforehand on its database. The more reports that came in, the more effective Akismet gets.

When a blog or a platform runs Akismet, it tries to match the comment they received with the messages available on their spam database. If it is identified as a spam, it would not be posted on the comment section or the forum. A user on that platform could report a comment as a spam, adding it to the spam database. In its development, Akismet can run not only on WordPress, but also in numerous other blog platforms and even forums using a public Akismet API.

B. Pattern Matching

Pattern matching is an algorithm to search for the appearance of a certain string (called 'pattern') in a certain sentence we call 'text'. Pattern matching usually showed the first occurence of the text in a pattern, but there are any implementations that utilize the way pattern matching algorithm works [4]. Pattern matching is used in many aspects of modern programming, such as for searching texts in a text editor, as a tool to implemet web search engine, analyzing image, analyzing the informations on bionformatics, and many more. Here's an example of text and pattern.

Text: change the words in such a way that a human

Pattern: such

Example 2.B.1 Brute Force Algorithm

There are several algorithms that can be used to find where does the text appear on the pattern. The algorithms that will be discussed in this paper are:

1. Brute Force Algorithm
2. Knuth-Morris-Pratt Algorithm (will be referred as the KMP Algorithm from this point onwards)
3. Boyer-Moore Algorithm (will be referred as the BM Algorithm from this point onwards)

Each of the algorithms have their own advantages and disadvantages, ranging from complexity to the size of the pattern or text being tested.

C. Brute Force Algorithm

This algorithm is perhaps the first one to come in mind, since it manually checks each position in the Text to see if the Pattern starts in that certain position. The Pattern moves one character at a time, from the first letter into the last letter of Text. Here's an example of Brute Force Algorithm using the Pattern 'NOT' and the Text 'NOBODY NOTICED HIM'.

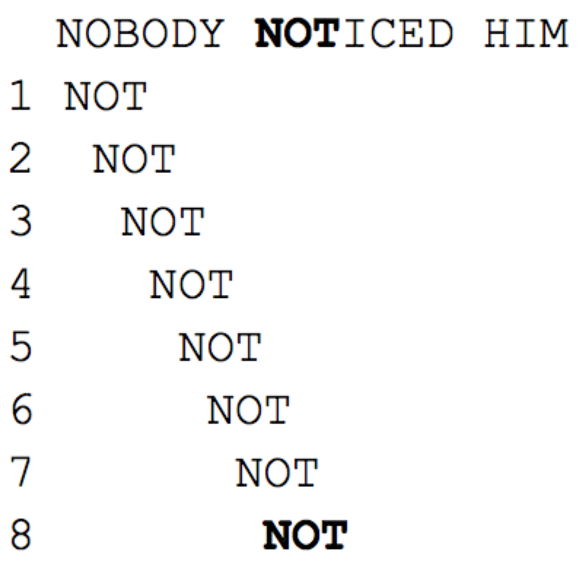


Image 2.C.1 Brute Force Algorithm (source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

Notice that even if 2nd or the 3rd word matched the text, the search keep on moving by 1 character. It simply matched the first letter of the Pattern to the current position in the Text, and when they matched, they will try to match the second letter in the Text into the next position in the Pattern.

The running time of the worst case is O(mn), when several suffix matched the Text but didn't match when they arrived at the last. The m represents the the length of the Pattern, while the n represents the length Text. The best case would be when no characters in Pattern matches the character in Text. The running time for this case is O(n). Example 2.A.1 represents the average case with a running time of O(m+n). Many classifies Brute Force Algorithm as one of the simplest pattern matching algorithm.

The algorithmic notation for the Brute Force Algorithm are as follows (with t as the array of string containing Text and p as the array of string containing Pattern):

```
function BruteForce
        posisi <- 0
        repeat while  posisi < n-m
                j <- 0
                repeat while j < m and t[posisi+j] == p[j]
                        j <- j + 1
                posisi <- posisi + 1
                if (j == m)
                return posisi
        return "Tidak ada pada Pattern"
```

D. Knuth-Morris-Pratt Algorithm (KMP Algorithm)

The Knuth-Morris-Pratt (KMP) algorithm looks for the Pattern like Brute Force, which is from left-to-right[3]. It is considerably more effective rather than the Brute Force algorithm, since it utilize the information they get from the previous matching process in order to save iterations rather than moving to the right one character at a time. One of the ways to utilize the KMP Algorithm is to use LPS array (Longest proper Prefix which is also Suffix).

The steps to make the LPS Table are [5]:

1. Make a one dimensional array (LPS) with the length equal to the length of the Pattern.
2. Set variables i = 0 and j = 0.
3. First of all, compare the character on Pattern[i] and Pattern[j]. (We treat the pattern as an array of string).
4. If the characters are identical, set LPS[j] as i+1. Add 1 to the variables i and j. Go back to step 3.
5. If the characters are not identical, check the value of i. If i = 0, set LPS[j] = 0 and add 1 to the variable. If i != 0, set i = LPS[i-1]. Go back to step 3.
6. Repeat above steps until all the values of LPS[] are filled.

For example, a pattern ABCDABD would make an LPS array of [0, 0, 0, 0, 1, 2, 0]. Here is an illustration on how to use LPS array on the test ABC ABCDAB ABCDABCDABDE.

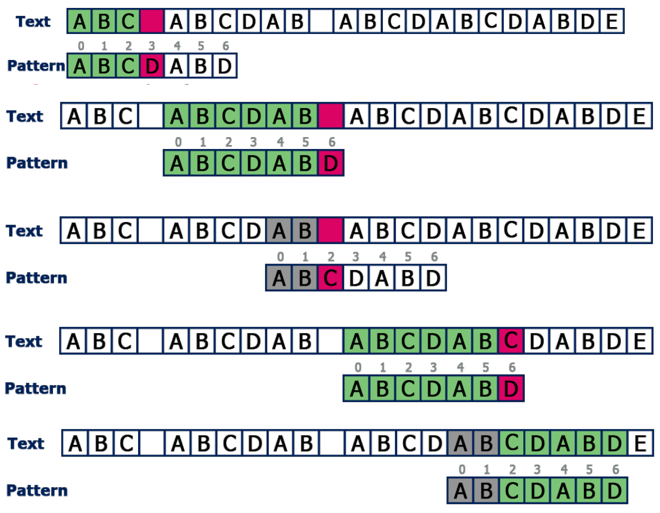


Image 2.D.1 The steps of KMP Algorithm using LPS array (source: http://www.btechsmartclass.com/data_structures/knuth-morris-pratt-algorithm.html)

The movement of the pattern is based on the number on the LPS array. For example, the first matching attempt ends on the 3rd character. We check the value of LPS[3-1] (which is LPS[2]), which is 0. Since the value is 0, the next matching process starts right after the 3rd character when we find the mismatch. The 4th matching process showed that the mismatch is on the 6th character. The value of LPS[5] is 2. Because of that, we checked from the 2nd character on the Pattern while putting the 2nd character on the place where we found the mismatch.

The complexity of KMP algorithm is O(m+n), and is much faster than the Brute Force algorithm. The implementation of KMP algorithm in Java is written below. The function computeFail is to return an array of LPS.

```
public static int kmpMatch(String text, String pattern)
{
           int n = text.length();
           int m = pattern.length();

           int fail[] = computeFail(pattern);
           int i=0;
           int j=0;
           while (i < n) {
                      if (pattern.charAt(j) ==
text.charAt(i)) {
                                 if (j == m - 1)
                                            return i - m +
1; // match
                                 i++;
                                 j++;
                      }
                      else if (j > 0)
                                 j = fail[j-1];
                      else
                                 i++;
           }
           return -1;
}
```

E. *Boyer-Moore Algorithm (BM Algorithm)*

Unlike Brute Force and KMP Algorithms, BM Algorithm is an algorithm that approaches the words backwards. It compares the string from the rightmost character, from right to left. It is also considered the most efficient string searching algorithm comparing to the 2 previous algorithms. The characters moved according to the status of the string, whether they're a full match or only a partial match. They used two functions to do the shift, which is good-suffix heuristic and bad-character heuristic.

Bad-character heuristic shifts the pattern when the character of the text doesnt match the current character on the pattern. It has two cases: it moves the pattern until the mismatch becomes a match, or it moves the pattern until it moves past the mismatched characters. The worst case took the running time of O(mn), and the best case is O(m/n).

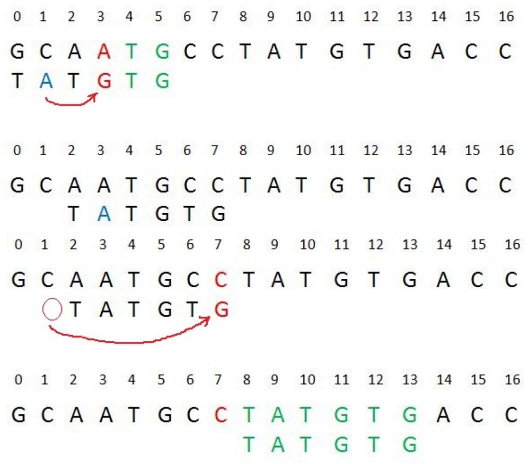


Image 2.E.1 The steps of Bad-character heuristic (source: https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/)

The Good-suffix heuristic shifts the pattern when the substring of Text matched with a certain substring of P. It shifts the pattern until there is another occurrence of the substring of Text in Pattern matched with substring of Text in Text, Pattern moves past the substring of Text, and there are a prefix of Pattern that matches the suffix of the substring of Text. It needs some preprocessing beforehand, differ for each cases.

III. Implementation and Solution

The implementation of how Akismet works using the Pattern Matching algorithm will be made using Python. It will mirror the process of getting the comments, checking it with the available database, delete the comment if the program recognizes it as a spam, and posting the comment if it does not recognize it as a spam.

It will be implemented using 2 algorithms, which are:

1. Knuth-Morris-Pratt Algorithm
2. Boyer-Moore Algorithm

These algorithms will be used to search for identifying whether the comment written resembles the spam comments available on the database. The algorithms are adapted from the author's Tugas Besar 3 and from the KMP Algorithm written by Bhavya Jain [2] with some modifications to accomodate the things the author want to show.

A. Knuth-Morris-Pratt Algorithm

The steps taken to implement the algorithm is to make the function to make the LPS array and the function to run the algorithm itself. The function is as follows:

```
def makeLPS(pattern, M, lps):
    panjang = 0

    lps[0]
    i = 1

    while i < M:
        if pattern[i]== pattern[panjang]:
            panjang += 1
            lps[i] = panjang
            i += 1
        else:
            if panjang != 0:
                panjang = lps[panjang-1]
            else:
                lps[i] = 0
                i += 1
```

It implements the steps taken to make an LPS array as mentioned on section II.D. It fills up the LPS array that is present on the KMP function. The KMP function is as follows:

```
def KMP(pattern, text):
    ada = 0
    M = len(pattern)
    N = len(text)

    lps = [0]*M
    makeLPS(pattern, M, lps)

    i = 0
    j = 0

    while i < N:
        if pattern[j] == text[i]:
            i += 1
            j += 1

        if j == M:
            ada = 1
            j = lps[j-1]

        elif i < N and pattern[j] != text[i]:
            if j != 0:
                j = lps[j-1]
            else:
                i += 1
    return ada
```

It implements the steps taken to run the KMP array as mentioned on section II.D. It returns whether there's a string that matches the pattern we input or not. The usage will be shown on the main program.

B. Boyer-Moore Algorithm

The function made to implement the algorithm is as follows:

```
def BM(pattern,text):#Boyer-Moore
    m=len(pattern)
    n=len(text)
    i=m-1
    dict={}
    for a in range(m):
        dict[pattern[a]]=a
    if(i>n-1):
        return False
    j=m-1
    while (i<n):
        if(pattern[j]==text[i]):
            if(j==0):
                return True
            else:
                i=i-1
                j=j-1
        else:
            if (text[i] in dict):
                lo = dict[text[i]]
            else:
                lo = -1
            i = i + m - min(j, 1+lo)
            j = m-1
    return False
```

It returns a true or false statement that shows whether the pattern shows up on the text or not. The implementation follows the theory mentioned on section II.E.

C. Main Program

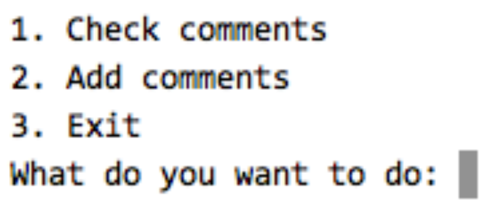
```
import csv

def addComment(spamlistarr, comlistarr):
    exist = 0
    comment = input("Add comment to this post: ")
    algo = input("Choose check algorithm (bm/kmp): ").lower()
    while (comment != "stop"):
        if (algo == 'kmp'):
            pos = 0
            while (pos < len(spamlistarr)):
                exist = KMP(comment,
''.join(spamlistarr[pos]))
                if exist:
                    print("Comment is a spam!")
                    break
                else:
                    pos += 1
            if exist == 0:
                addComDB(comment, comlistarr)
                print("Your comment is added to the comment 
section.")
        elif (algo == 'bm'):
            pos = 0
            while (pos < len(spamlistarr)):
                exist = BM(comment, 
''.join(spamlistarr[pos]))
                if exist:
                    print("Comment is a spam!")
                    break
                else:
                    pos += 1
            if exist == 0:
                addComDB(comment, comlistarr)
                print("Your comment is added to the comment 
section.")
        else:
            print("Input is wrong.")
        exist = 0
        comment = input("\nAdd comment to this post: 
").lower()
        if comment == 'stop':
            break
        algo = input("Choose algorithm (bm/kmp): ").lower()

def chat(spamlistarr, comlistarr):
    print('''1. Check comments
2. Add comments
3. Exit''')
    chatnya = input("What do you want to do: ")
    if chatnya == '1':
        no = 1
        for item in comlistarr:
            print(no,'.',''.join(item))
            no += 1
        report = input("Do you want to report? (Y/N): ")
        if report == "Y":
            commentreport = int(input("Which comment? 
(number): "))
            print(comlistarr[commentreport-1])
            addSpamDB(comlistarr[commentreport-1], 
spamlistarr)
        else:
            chat(spamlistarr, comlistarr)
    elif chatnya == '2':
        addComment(spamlistarr,comlistarr)
        chat(spamlistarr, comlistarr)
    elif chatnya == '3':
        quit()
    else:
        print("Your input is wrong.")
        chat(spamlistarr, comlistarr)


spamlistarr = readDBspam()
comlistarr = readDBcomment()
chat(spamlistarr, comlistarr)
```

The function 'chat' is the main function that combines all the previous declared functions. It basically asked whether the user wants to see available comments on a certain post, or to write a new comment on that certain post. It asked the user for the comment. For testing sake, this implementation also asks which algorithm they want to use whether the comment is a spam or not. If a comment is considered a spam, it won't get merged into the comment section, showing 'Comment is a spam!'. However, if the comment is not a spam, they will be added to the comment section.

It also allows the user to see and report a comment if they considered it a spam. That comment will be added to the spam database and the next time similar post showed up, it will be considered as a spam. There are also some read and write function for accessing the database which are not included in the code snippet above.

IV. Test

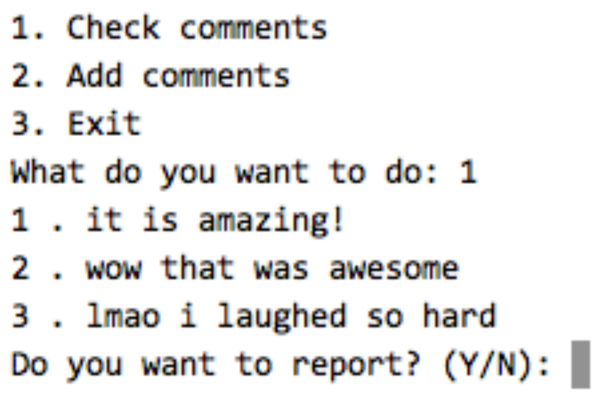
The program will first ask what action the user want to do on that certain post. They could check the comments, or add a new comment.



```
1. Check comments
2. Add comments
3. Exit
What do you want to do: 
```

Image 4.1 Choosing the action

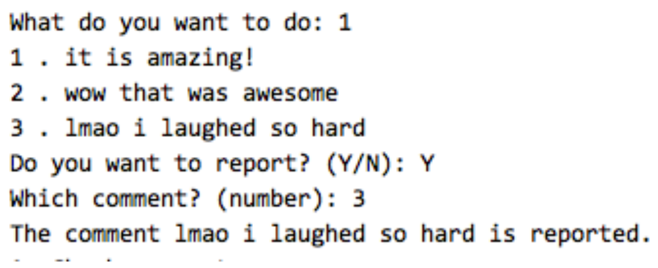
If the user chose to check the comments, all the comments on that post will be shown.



```
1. Check comments
2. Add comments
3. Exit
What do you want to do: 1
1 . it is amazing!
2 . wow that was awesome
3 . lmao i laughed so hard
Do you want to report? (Y/N): 
```

Image 4.2 Showing comments

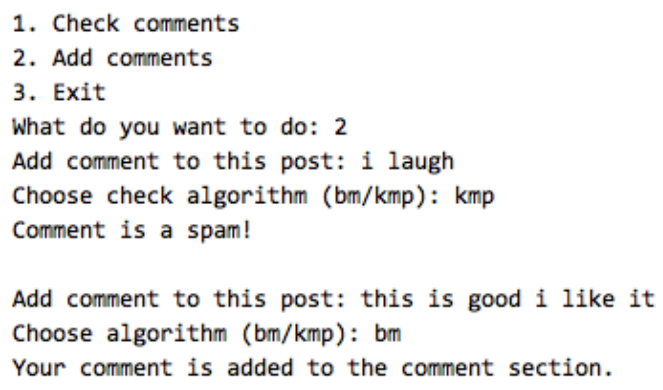
While looking at the comments, the user could report a comment they think not appropriate. For example, let's say the third comment is a spam.



```
What do you want to do: 1
1 . it is amazing!
2 . wow that was awesome
3 . lmao i laughed so hard
Do you want to report? (Y/N): Y
Which comment? (number): 3
The comment lmao i laughed so hard is reported.
```

Image 4.3 Reporting a comment

When someone writes that similar comment that we reported, they will be deemed as a spam, thus not being added to the comment section.



```
1. Check comments
2. Add comments
3. Exit
What do you want to do: 2
Add comment to this post: i laugh
Choose check algorithm (bm/kmp): kmp
Comment is a spam!

Add comment to this post: this is good i like it
Choose algorithm (bm/kmp): bm
Your comment is added to the comment section.
```

Image 4.4 Blocking a spam

When it recognizes a spam, it shows that the comment is a spam, thus not being added to the comment section. However, when the comment is not considered a spam, the comment is added to the comment section.

Of course, Akismet did not inform the spammers that their comment is a spam. Usually, it shows up on the blogger's dashboard, asking them whether they want to report it or not. It checks all the available spam comments on the database, and stopped when they see a similar one.

V. Conclusion

Spam mails has been around the internet since the start of its existence, and it will not stop anytime soon. With systems such as Akismet, we could fight the spammers altogether by crowd-reporting suspicious comments so that the system could learn all kinds of spam mails. We may not be able to outwit the spam messages when it first appeard, but we could fight it if we reported it well.

Pattern matching could be used to fight these spam messages well. It is a bery basic theory that could help many sustems to fight spam messages.

Video Link at YouTube

The author made a video to help readers understand this paper better. The video can be accessed on https://youtu.be/g7K5xQwU2l8.

Acknowledgment

The author would like to say thank you to everyone who had supported the author in the process of writing this paper. To God, who had provided the author with life and mind so that this paper could be finished in time. To Dr. Nur Ulfa Maulidevi, S.T., M.Sc as the lecturer of the Algorithm Strategies course in which the author took and got inspirations from. To friends, who gave encouragements and keeping the author's spirits up. To my parents, whom I missed so much and supported me from far away, even between the hardships they are facing.

References

[1] "Milestones: The Story of WordPress", from https://github.com/WordPress/book/blob/trunk/Content/Part%203/17-akismet.md accessed on 10th of May 2021

[2] "Python Program for KMP Algorithm for Pattern Searching", from https://www.geeksforgeeks.org/python-program-for-kmp-algorithm-for-pattern-searching-2/ accessed on 8th of May 2021

[3] "Pencocokan String (String/Pattern Matching)", from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf accessed on 11th of May 2021

[4] Philippe Jacquet, Wojciech Szpankowski, "Analytic Pattern Matching: From DNA to Twitter", 2015.

[5] "Knuth-Morris-Pratt Algorithm", from http://www.btechsmartclass.com/data_structures/knuth-morris-pratt-algorithm.html accessed on 11th of May 2021

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Mei 2021

[illegible]

Jeanne D'Arc Amara Hanieka - 13519082