Application of String Matching Algorithm on Twitter "Muted Words" Feature

Indra Febrio Nugroho - 13518016
Informatics
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl Ganesha 10 Bandung
13518016@std.stei.itb.ac.id

***Abstract*—Twitter is one of the rapidly growing social media on which their users can post and interact with messages called tweets. Twitter users can send their tweets up to 280 characters per tweet. As a character-based social media, Twitter has a feature called Muted Words. It allows their users to hide a word or a phrase so that it is not showing on their timeline. This paper will discuss about the application of String Matching algorithm on Twitter Muted Words feature.**

Keywords—twitter; string matching; muted words; algorithm; knuth morris pratt; boyer moore

I. INTRODUCTION

In this day, social media is growing at a rapid pace. It is a place of creation or sharing of information, career opportunities, ideas, and many other forms of expression. In social media we can also see what is currently happening in the world, as the information shared in social media directly hits their users at a count of seconds.

Social media gives their users infinite freedom. Freedom here means they can easily access and dig any contents shared within social media without any restrictions. This thing here gives us pros and cons.

Social media can be used as a tool for studying. Their users can use it to ask each other difficult questions that they could not resolve on their own. Or they just can simply search for the answers of their questions. They can also make a discussion group or a study group even when they are far from each other.

Social media can also be a platform to freely express themselves, especially for teenagers. They can join groups or fan pages that they are interested in which reflects their own personality. They can also share their current hobbies and skills so that the world knows they are great, thus boosting their self-esteem. This confidence they got from social media can help boost their confidence in real life too, it means that they are able to pursue their passions to the fullest.

At some points social media gives bad impacts. One example that needs to be paid attention to is cyberbullying. Users of social media are exposed to cyberbullying. It gives them negative influences. Cyberbullies can easily bully and taunt others, as it is easier for them to do bully through social media than to do it physically in real world. As nowadays social media offers us a huge level of networking, cyberbullying becomes unbearable and is getting worse day by day leaving long lasting pain to the victim.

Another example of social media bad impacts that needs to be paid attention to is the freedom of access to inappropriate contents, especially for children. Children who do not have knowledge on what kind of content that is inappropriate to view are more likely to explore anything social media can offer. If they get access to such content, it can cause mental and emotional damage to them. It can also cause them to have nightmares and change in behavior. Without any restrictions and parental control from their parents, this problem can become unbearable and get worse in the future.

One of the social media offered in this online world is Twitter. Twitter is a platform where their users can post and interact with messages called tweets. It is a character-based social media. Twitter has a feature that can restrict a word or phrase so that it is not going to be displayed in their users' timeline. It is called muted words.

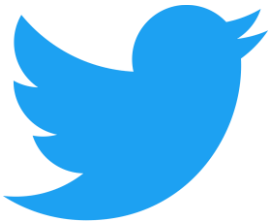

Image 1 Twitter logo (source Wikipedia)

In the next sections I am going to explain about how muted words does its job on Twitter (how it searches for word and doesn't display it on the user timeline) by using String Matching algorithm. I will also give some simulator program using Knuth-Morris-Pratt and Boyer Moore algorithm so that it can be understood better.

II. THEORETICAL BASIS

A. String Matching Algorithm

String-matching is a very important subject in the domain of text processing. String-matching algorithms are basic components used in implementations of practical softwares existing under most operating systems. They also play an

important role in theoretical computer science by providing challenging problems.

Although data are stored in various ways, text still remains the main form to exchange information. This is particularly evident in literature where data are composed of huge dictionaries. This apply as well to computer science where a large amount of data are stored in files. And this is also the case, for instance, in molecular biology. Furthermore, the quantity of available data in these fields tend to double every eighteen months. This is the reason why algorithms should be efficient even if the speed and capacity of storage of computers increase regularly.

String-matching is to find one or more all the occurrences of a string (usually called a pattern) in a text.

The pattern is denoted by x = x[0 .. m-1]; its length is equal to m.

The text is denoted by y = y[0 .. n-1]; its length is equal to n.

Applications require two kinds of solution depending on which string, the pattern or the text, is given first. Algorithms based on the use of automata or combinatorial properties of strings are commonly implemented to preprocess the pattern and solve the first kind of problem. The notion of indexes realized by trees or automata is used in the second kind of solutions.

String-matching algorithms of the present book work as follows. They scan the text which size is generally equal to m. They first align the left ends of the text, then compare the characters of the pattern and after a whole match of the pattern or after a mismatch they shift the alignment to the right. They repeat the same procedure again until the right end of the alignment goes beyond the right end of the text.

The **Brute Force algorithm** locates all occurrences of x in y in time **O(mn)**. The many improvements of the brute force method can be classified depending on the order they performed the comparisons between pattern characters and text characters et each attempt. They are: the most natural way to perform the comparisons is from left to right, which is the reading direction; performing the comparisons from right to left generally leads to the best algorithms in practice; the best theoretical bounds are reached when comparisons are done in a specific order; finally there exist some algorithms for which the order in which the comparisons are done is not relevant. In this paper only two algorithms will be discussed, they are Knuth-Morris-Pratt algorithm and Boyer Moore algorithm.

Image 2 Pattern matching illustration (source GeeksForGeeks)

B. Knuth-Morris-Pratt Algorithm

The Brute Force (Naïve) pattern searching algorithm doesn't work well in cases where we see many matching characters followed by a mismatching character. For example:

```
y = "AAAAAAAAAAAAAAAAAB"
x = "AAAAB"
```

The KMP matching algorithm uses degenerating property (pattern having same sub-patterns appearing more than once in the pattern) of the pattern and improves **the worst case** complexity to **O(n).** The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of the next window. We take advantage of this information to avoid matching the characters that we know will anyway match. Consider example below.

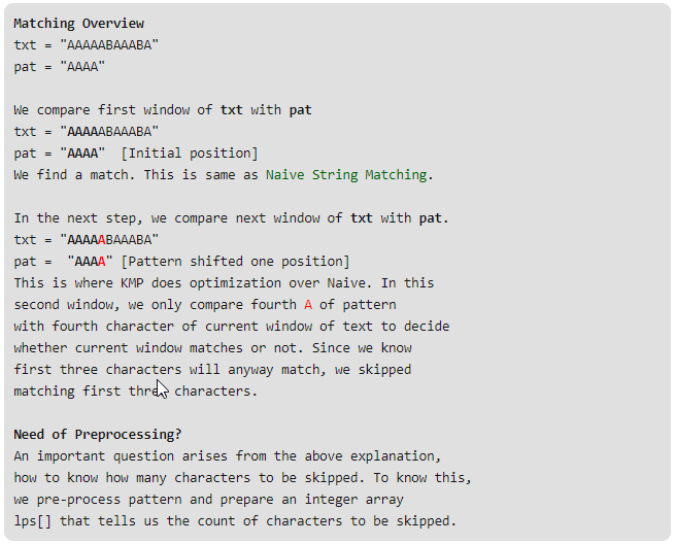

Image 3 KMP matching overview (source GeeksForGeeks)

Preprocessing Overview:

1. KMP algorithm preprocesses x and constructs an auxiliary **lps** of size m (same as size of pattern) which is used to skip characters while matching.
2. name **lps** indicates longest proper prefix which is also suffix. A proper prefix is prefix with whole string not allowed. For example, prefixes of "ABC" are "", "A", "AB" and "ABC". Proper prefixes are "", "A" and "AB". Suffixes of the string are "", "C", "BC" and "ABC".
3. We search for **lps** in sub-patterns. More clearly we focus on sub-strings of patterns that are either prefix and suffix.
4. For each sub-pattern x[0..i] where i = 0 to m-1, **lps**[i] stores length of the maximum matching proper prefix which is also a suffix of the sub-pattern x[0..i].

lps[i] = the longest proper prefix of x[0..i] which is also a suffix of x[1..i].

Note: lps[i] could also be defined as longest prefix which is also proper suffix.

Searching Algorithm:

Unlike Brute Force algorithm, where we slide the pattern by one and compare all characters at each shift, we use a value from **lps** to decide the next characters to be matched.

How to use **lps** to decide next positions (or to know a number of characters to be skipped)?

- We start comparison of x[j] with j = 0 with characters of current window of text.
- We keep matching characters y[i] and x[j] and keep incrementing i and j while x[j] and y[i] keep matching.
- When we see a mismatch
    - We know that characters x[0..j-1] match with y[i-j…i-1] (Note that j starts with 0 and increment it only when there is a match).
    - We also know (from above definition) that lps[j-1] is count of characters of x[0…j-1] that are both proper prefix and suffix.
    - From above two points, we can conclude that we do not need to match these lps[j-1] characters with y[i-j…i-1] because we know that these characters will anyway match. Let us consider above example to understand this.

Preprocessing Algorithm:

In the preprocessing part, we calculate values in **lps**. To do that, we keep track of the length of the longest prefix suffix value for the previous index (len). We initialize lps[0] and len as 0. If x[len] and x[i] match, we increment len by 1 and assign the incremented value to lps[i]. If x[i] and x[len] do not match and len is not 0, we update len to lps[len-1].

C. *Boyer Moore Algorithm*

Boyer Moore is a combination of following two approaches.

1) Bad Character Heuristic

2) Good Suffix Heuristic

Both of the above heuristics can also be used independently to search a pattern in a text. First, we understand how two independent approaches work together in the Boyer Moore algorithm. It processes the pattern and creates different arrays for both heuristics. At every step, it slides the pattern by the max of the slides suggested by the two heuristics. So it uses best of the two heuristics at every step.

Unlike the previous pattern searching algorithms, Boyer Moore algorithm starts matching from the last character of the pattern (usually called looking glass technique).

Bad Character Heuristic

The idea of bad character heuristic is the character of the text which doesn't match with the current character of the pattern is called the Bad Character. Upon mismatch, we shift the pattern until –

1) The mismatch becomes a match.

2) Pattern P move past the mismatched character.

Case 1 – Mismatch become match

We're going to look the position of last occurrence of mismatching character in pattern and this bad character exists in pattern then we'll shift the pattern such that it gets aligned to the bad character in the text.

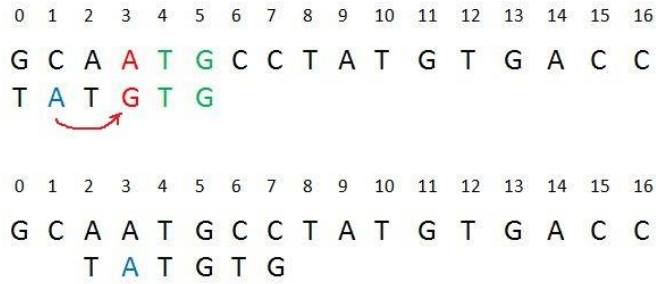

Image 4 Case 1 BM Bad heuristic (source GeeksForGeeks)

Case 2 – Pattern move past the mismatch character

We're going to look the position of last occurrence of the bad character in pattern and if it does not exist we shifts the pattern past the bad character.

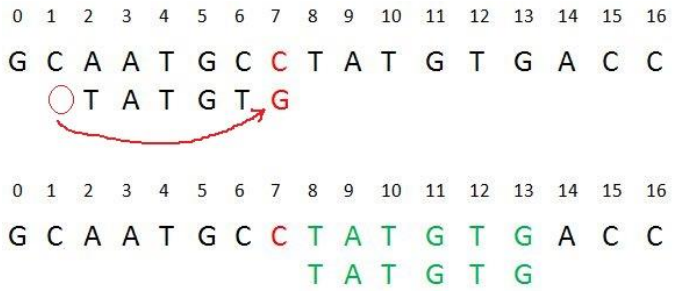

Image 5 Case 2 BM Bad Heuristic (source GeeksForGeeks)

The bad character heuristic takes O(n/m) time in the best case. The Bad Character Heuristic may take O(mn) time in worst case.

Good Suffix Heuristic

Let t be substring of text T which is matched with substring of pattern P. Now we shift pattern until:

1) Another occurrence of t in P matched with t in T.

2) A prefix of P, which matches with suffix of t

3) P moves past t

Case 1: Another occurrence of t in Pattern matched with t in Text

Pattern P might contain few more occurrences of t. In such case, we will try to shift the pattern to align that occurrence with t in text T. For example-

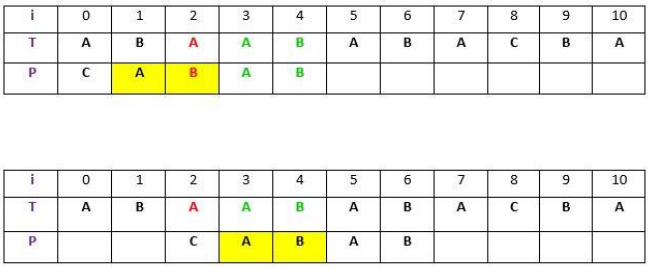

Figure – Case 1

Image 6 Case 1 BM good heuristic (source GeeksForGeeks)

Case 2: A prefix of Pattern, which matches with suffix of t in Text

It's not always likely that we'll find the occurrence of t in P. Sometimes there is no occurrence at all, in that case we can search for some suffix of t matching with some prefix of P and try to align them by shifting P. For example –

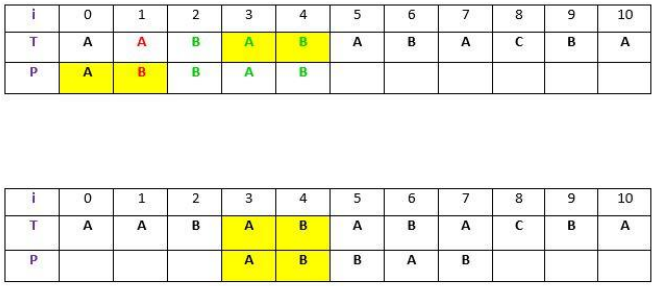

Figure – Case 2

Image 7 Case 2 BM good heuristic (source GeeksForGeeks)

Case 3: Pattern moves past t

If the above two cases are not satisfied, we will shift the pattern past the t. For example –

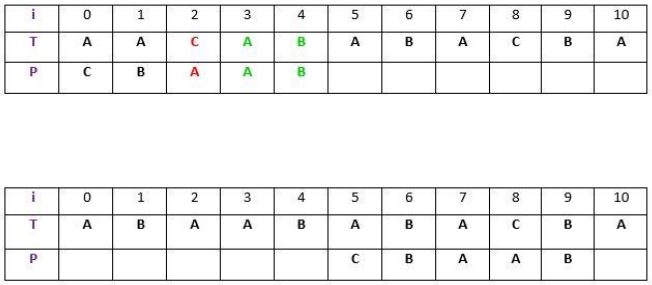

Figure – Case 3

Image 8 Case 3 BM good heuristic (source GeeksForGeeks)

III. Solution Breakdown

Before going onto the implementation, let's get to know about how Twitter timeline algorithm and muted words work (timeline is a place where tweets of users' following appear).

A. Twitter Timeline Algorithm

The Twitter timeline consists of three main sections:

- Top tweets
- "In case you missed it"
- Tweets in reverse-chronological order

Every time users visit Twitter, the algorithm will study all the tweets from accounts that users follow and give each of them a score based on several factors. Here are some of the factors:

- The tweet itself: its recency, presence of media cards (image or video), and overall engagement (including retweets, clicks, favorites, and time spent reading it)
- The tweet's author: users past interactions with this author, the strength of users' connection to them, and the origin of users' relationship

Then, Twitter will put the tweets fulfilling those factors in the first two sections — top tweets and "In case you missed it".

Top Tweets

This section will appear at the top of users' timeline and is the regular timeline. It contains tweets that Twitter thinks are relevant to users. The selected tweets might not be ordered reverse-chronologically.

"In Case You Missed It"

This section does as its name suggests. It shows users tweets that they might be interested in but might not see in the old timeline as they were from quite some time ago.

This section seems to only appear in users' timeline when they have been away from Twitter for several hours or days. Similar to the top tweets section, this section contains tweets that Twitter thinks are relevant to users. The selected tweets are

ordered according to their relevance score and might be from many hours or days ago.

Tweets in Reverse-Chronological Order

After the first two sections, users will see tweets of this section from accounts they follow in the original reverse-chronological order. Just like the old Twitter timeline.

In this section (and sometimes in the two above), users will also find retweets, promoted tweets, and suggested accounts to follow. They might even see tweets from accounts they don't follow. These are often tweets that Twitter thinks will make your timeline more relevant and interesting.

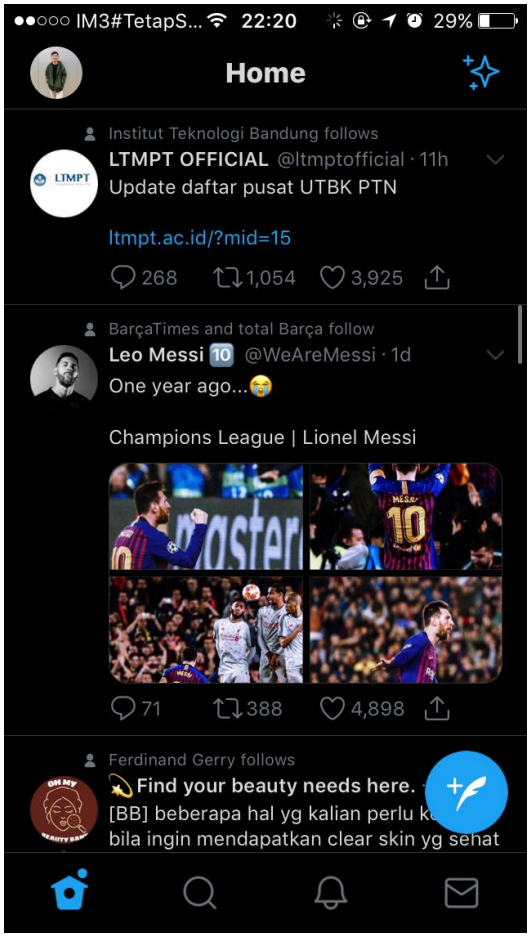

Image 9 Users timeline on Twitter (source author's Twitter)

B. *Muted Words*

When users find a content in tweets that they would like to avoid to see in their timeline, muted words are the right choice for them. After they add the words they'd like to avoid, they won't see any of them in their timeline anymore. This feature here is also useful for parents who need to do parental control on their underage child to avoid them seeing any inappropriate content.

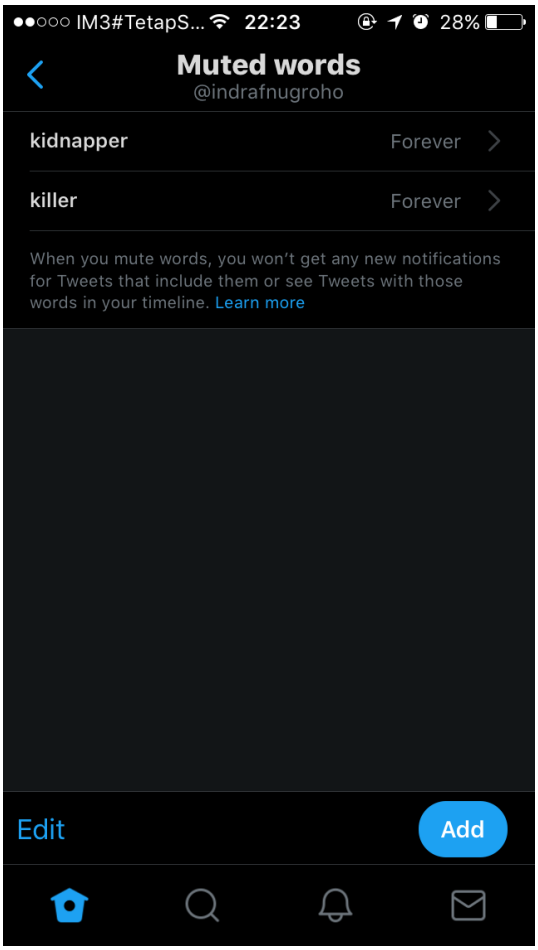

Image 10 Muted words features (source author's Twitter)

C. *Implementation Plan*

The author is going to make a program with Python who will search for a pattern (muted words) in the text (tweets) using Knuth-Morris-Pratt and Boyer Moore algorithm. First, the program will collect tweets data in a txt format. Then it will search for the muted word(s) that might be contained in the collected tweets. If any of the muted word(s) is found, the program will not show the collected tweets to the author's screen.

IV. Implementation

A. *Collecting Tweets from Twitter*

I extracted some tweets with hashtag #ApologizeToWendy that is currently a trending in Indonesia on 2nd of May 2020 at about 11 PM. I assembled it all in a txt file format. This collected tweets is going to be the test case for the simulator.

B. *Building the Program*

Before proceeding to do string matching between muted words and the tweets, the code for Knuth-Morris-Pratt and Boyer Moore must have been implemented first. After that, extract the collected tweets and save it into an array containing

tweets per user (author used split function to split the collected tweets to sentences). Search for the muted words in the sentence. If it is found, then hide the tweet from users' timeline so that it is not going to be displayed in their timeline. If it is not, show it to their timeline.

C. Runnng the Program

Firstly, the program will show tweets from users following account in their timeline. The program is showing the top 5 tweets based on Twitter algorithm.

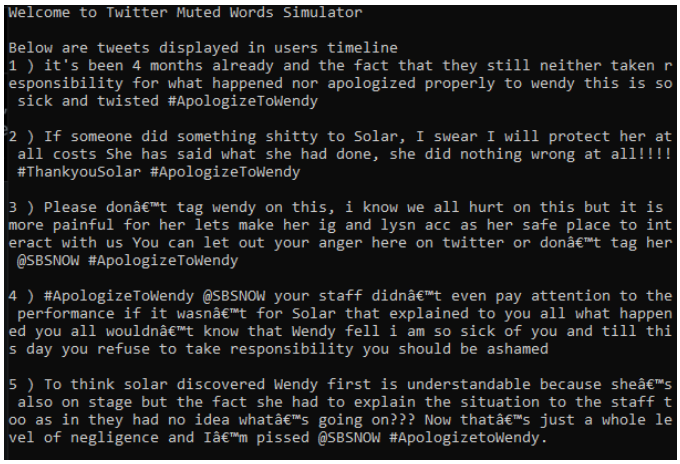

Image 11 Users timeline illustration (source author's personal program)

Secondly, the program will ask the users to input the word that is going to be included in the muted words list. Once users input the word to the muted words list, the word will not be displayed in their timeline forever unless the users itself remove the word from the muted words list. The author is going to input an explicit word that is not appropriate for children.

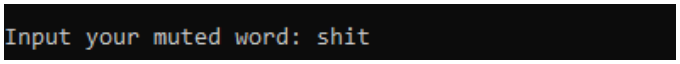

Image 12 Word to be included in the muted words list (source author's personal program)

Lastly, refresh the program. Now the program is free from the explicit word!

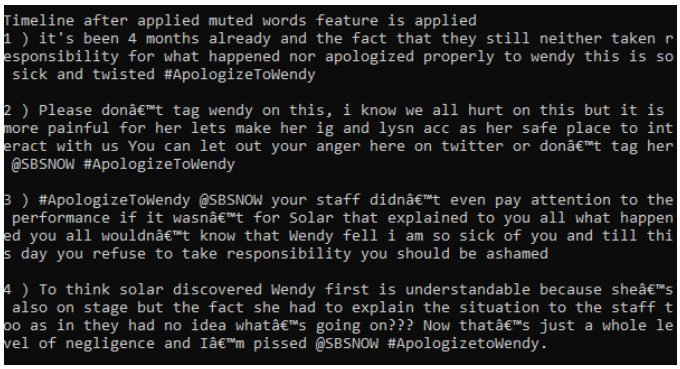

Image 13 The new users' timeline without the muted words (source author's personal program)

V. Conclusion

Muted words feature on Twitter is really helpful to avoid unwanted content to be displayed in the users' timeline. The words could be an inappropriate content, or they are just certain words that they don't want to see.

Video Link at Youtube

To understand this paper better, I made an explanation video about this topic on my YouTube channel. Check this out.

https://youtu.be/OwqIyFG6N0s

Acknowledgment

I would like to thank Dr. Masayu Leylia Khodra S.T., M.T.; Dr. Nur Ulfa Maulidevi, S.T., M.Sc; and Dr. Rinaldi Munir as the lecturers of IF2211 Algorithm Strategies course. Thank you for giving me a chance to explore something great like writing this paper. It's been an honor for me to learn from them and gain inspiration to continue studying even in this pandemic.

I would like to thank my parents too, for giving me guidance on everything, especially my mom. She is the one who keeps on supporting me and tells me to never give up.

Lastly, I would also thank my best friend, for being such a good friend for me in the last few months. Thanks for giving me such great lessons to make me a better person day by day.

References

[1] Charras, Cristian; Lecroq, Thierry. "String Matching" from https://www-igm.univ-mlv.fr/~lecroq/string/node2.html accessed on 2 May 2020

[2] Pattern Searching from https://www.geeksforgeeks.org/algorithms-gq/pattern-searching/ accessed on 2 May 2020

[3] KMP algorithm for Pattern Searching from https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/ accessed on 2 May 2020

[4] Boyer Moore Algorithm for Pattern Searching from https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/?ref=rp accessed on 2 May 2020

[5] Boyer Moore Algorithm Good Suffix heuristic from https://www.geeksforgeeks.org/boyer-moore-algorithm-good-suffix-heuristic/ accessed on 2 May 2020

[6] Lua, Alfred. “How the Twitter Timeline Works (and 6 Simple Tactics to Increase Your Reach)” from https://buffer.com/library/twitter-timeline-algorithm accessed on 2 May 2020

[7] Koumchatzky, Nicolas; Andryeyev, Anton. “Using Deep Learning at Scale in Twitter’s Timelines” from https://blog.twitter.com/engineering/en_us/topics/insights/2017/using-deep-learning-at-scale-in-twitters-timelines.html accessed on 2 May 2020

[8] Oremus, Will. “Twitter’s timeline algorithm, and its effect on us, explained.” From http://www.slate.com/articles/technology/cover_story/2017/03/twitter_s_timeline_algorithm_and_its_effect_on_us_explained.html accessed on 2 May 2020

[9] How to Use Advanced Muting Options from https://help.twitter.com/en/using-twitter/advanced-twitter-mute-options accessed on 2 May 2020

STATEMENT

I hereby declare that the paper I wrote is my own writing, not an adaptation, or a translation of someone else's paper, and not plagiarism

Bekasi, 3 May 2020

Indra Febrio Nugroho - 13518016