KMP Algorithm in Auto Correct for Text Messaging

Muhammad Azka Widianto 13516127
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
azkawidyanto@gmail.com

Abstract—Text Messaging is very necessary in our life to make us connected to each other with our smartphone. As we are using text messaging, we also familiar to do some typographical error while we do text messaging. Some typographical error while we type the message will disturb us because it can make some meaning changed. Some smartphone has already have a feature of auto correct for text messaging, which is reduce typographical error from text messaging. This paper will further discuss about tolls to reduce typographical error with auto correct in text messaging for smartphone and how string matching algorithm (KMP algorithm) implemented in analysis.

Keywords—KMP, Text Messaging, string, matching.

I. INTRODUCTION

Typographical was usually happen as technical error from the keyboard or it was human error when they type message in the phone. Typographical errographical error or we known as typographical error in daily life was very familiar to us, typographical error usually just a small mistake when we typed some message, but in other case it will be a very big mistake. For some people who involve in a formal business or formal education, typographical error will affect respect from someone or maybe from your lecturer.

Normally everyone will try to give the best message when they text the lecturer or business partner, but in other case as we are human, sometimes we do mistake when it text to lecturer or business partner. Sometimes people want to diminish typographical error when they texting to others, but they haven’t turned on the feature of auto correct on their phone, which is very useful to them to diminish typographical error when texting. Auto correct in some smartphone will be default to be always on to reduce some small mistake, and make the user easier to use.

Auto correct feature in smartphone use string matching to make the text messaging easier. String matching have some variant, but i will tell more about KMP algorithm for string matching. In this case, we just using sample of implementation KMP algorithm in auto correct for text messaging, but other algorithm which is about string matching still used by string matching in auto correct feature for text messaging.

Auto correct and text messaging will be a unity that in this age will be very useful to used, but it haven’t known by lots of people right now. This feature still have some bug sometimes, like to much replacement or replace word that we don’t type, but it have already help people reduce typographical error further now.

II. THEORY

A. Auto correct feature

Auto correct or text replacement is an automatic data validation function commonly found in word processors and text editing interfaces for smartphones and tablet computers. Its principal purpose is as part of the spell checker to correct common spelling or typing errors, saving time for the user. It is also used to automatically format text or insert special characters by recognizing particular character usage, saving the user from having to use more tedious functions.

In certain situations, automatic corrections can cause problems. This is particularly so in technical and scientific writing. For example, the biochemical cyclic adenosine monophosphate is commonly referred to as "cAMP", which in turn is abbreviated to "cAMP". A text replacement function may regard this capitalization to be erroneous, and so change it to "Camp", which in the context of biochemistry is incorrect.

Older automatic-correction algorithms can cause problems even in nontechnical writing; the Cupertino effect was an example: cooperation (which some dictionaries would not recognize unless hyphenated cooperation) became Cupertino.

Some writers and organizations choose to consistently replace some words with others as part of their editorial policy, with occasionally unforeseen results. For example, the American Family Association chose to replace all instances of the word, "gay", on its website with the word, "homosexual".

This caused an article about US Olympic sprinter Tyson Gay to be littered with confusing sentences such as, "In Saturday's opening heat, Homosexual pulled way up, way too soon, and nearly was caught by the field, before accelerating again and lunging in for fourth place".

B. String matching

String matching algorithms, are an important class of string algorithms that try to find a place where one or
several strings (also called patterns) are found within a larger string or text.

String matching algorithm have some single pattern algorithm, there are naive string algorithm, Boyer-Moore string search algorithm, Rabin-Karp string search algorithm, Knuth-Morris-Pratt algorithm, Bitap algorithm, Two way string matching algorithm, Backward Non-Deterministic Dawg Matching, and Backward Oracle Matching.

1. String Matching Concept
   String matching method in brief:
   1. The algorithm is based on the preprocessing/filtering/verification paradigm.
   2. The preprocessing phase generates all strings of length, and computes their minimum distance over the set of patterns.
   3. The filtering phase searches (approximately) text -grams from the patterns, using the precomputed distance table, accumulating the differences.
   4. The verification phase uses dynamic programming algorithm, and is applied to each pattern separately

   The concept of string matching:
   Suppose S is a string with length m.
   \[ S = x_1, x_2, \ldots, x_m \]
   A prefix of S is substring \[ S[0..k] \].
   A suffix of S is substring \[ S[k..m-1] \].
   Where \( k \) is an index between 0 until m-1.

   Examples:
   All possible prefixes of STIMA are “S”, “ST”, “STI”, “STIM”. Meanwhile all possible suffixes of STIMA are “A”, “MA”, “IMA”, “TIMA”.

Many symbol systems include characters that are synonymous (at least for some purposes):
- Latin-based alphabets distinguish lower-case from upper-case, but for many purposes string search is expected to ignore the distinction.
- Many languages include ligatures, where one composite character is equivalent to two or more other characters.
- Many writing systems involve diacritical marks such as accents or vowel points, which may vary in their usage, or be of varying importance in matching.
- DNA sequences can involve non-coding segments which may be ignored for some purposes, or polymorphisms that lead to no change in the encoded proteins, which may not count as a true difference for some other purposes.
- Some languages have rules where a different character or form of character must be used at the start, middle, or end of words.

Finally, for strings that represent natural language, aspects of the language itself become involved. For example, one might wish to find all occurrences of a "word" despite it having alternate spellings, prefixes or suffixes, etc.

Another more complex type of search is regular expression searching, where the user constructs a pattern of characters or other symbols, and any match to the pattern should fulfill the search.

For example, to catch both the American English word "color" and the British equivalent "colour", instead of searching for two different literal strings, one might use a regular expression such as: 

\[ colour?r \]

where the "?" conventionally makes the preceding character ("u") optional.

C. Knuth–Morris-Pratt (KMP) Algorithm
Knuth–Morris–Pratt string searching algorithm or KMP algorithm searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself personify sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.

The algorithm was conceived in 1970 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris.

This is the first linear time algorithm for string matching. The three published it jointly in 1977. Independently, in 1969, Matiyasevich discovered a similar algorithm, coded by a two-dimensional Turing machine, while studying a string pattern-matching recognition problem. KMP Algorithm may be specified as follows

algorithm kmp_search:
input: an array of characters, S (the text to be searched)
      an array of characters, W (the word sought)
output: an array of integers, P (positions in S at which W is found)
define variables:
an integer, nP (positions in S at which W is found)
an integer, j, nP (positions of numbers)

let \( j \leftarrow 0 \) (the position of the current character in S)
let \( k \leftarrow 0 \) (the position of the current character in W)

let nP \leftarrow 0
while \( j < \text{length}(S) \) do
  if \( W[k] = S[j] \) then
    let \( j \leftarrow j + 1 \)
    let \( k \leftarrow k + 1 \)
  if \( k = \text{length}(W) \) then

(occurrence found, if only first occurrence is needed, m may be returned here)

let \( P[nP] \leftarrow j - k, nP \leftarrow nP + 1 \)
else
  let \( k \leftarrow T[k] \) (T[length(W)] can’t be -1)
let \( j \leftarrow j + 1 \)
let \( k \leftarrow k + 1 \)
A string matching algorithm wants to find the starting index $m$ in string $S[]$ that matches the search word $W[]$.

The most straightforward algorithm is to look for a character match at successive values of the index $m$, the position in the string being searched, i.e. $S[m]$. If the index $m$ reaches the end of the string then there is no match, in which case the search is said to "fail". At each position $m$ the algorithm first checks for equality of the first character in the word being searched, i.e. $S[m] = W[0]$. If a match is found, the algorithm tests the other characters in the word being searched by checking successive values of the word position index, $i$. The algorithm retrieves the character $W[i]$ in the word being searched and checks for equality of the expression $S[m+i] = W[i]$. If all successive characters match in $W$ at position $m$, then a match is found at that position in the search string.

Consider an attempt at a left position $j$, that is when the the window is positioned on the text factor $y[j .. j+m-1]$. Assume that the first mismatch occurs between $x[i]$ and $y[i+j]$ with $0 < i < m$. Then, $x[0 .. i-1] = y[j .. i+j-1] = u$ and $a = x[i] neq y[i+j] = b$.

When shifting, it is reasonable to expect that a prefix $v$ of the pattern matches some suffix of the portion $u$ of the text. Moreover, if we want to avoid another immediate mismatch, the character following the prefix $v$ in the pattern must be different from $a$. The longest such prefix $v$ is called the tagged border of $u$ (it occurs at both ends of $u$ followed by different characters in $x$).

This introduces the notation: let $kmpNext[i]$ be the length of the longest border of $x[0 .. i-1]$ followed by a character $c$ different from $x[i]$ and $-1$ if no such tagged border exits, for $0 < i \leq m$. Then, after a shift, the comparisons can resume between characters $x[kmpNext[i]]$ and $y[i+j]$ without missing any occurrence of $x$ in $y$, and avoiding a backtrack on the text (see figure 7.1). The value of $kmpNext[0]$ is set to $-1$.

Fig 1 Shift in the Knuth-Morris-Pratt algorithm ($v$ border of $u$ and $c \neq b$).

The table $kmpNext$ can be computed in $O(m)$ space and time before the searching phase, applying the same searching algorithm to the pattern itself, as if $x=y$.

Assuming the prior existence of the table $T$, the search portion of the Knuth–Morris–Pratt algorithm has complexity $O(n)$, where $n$ is the length of $S$ and the O is big-O notation. Except for the fixed overhead incurred in entering and exiting the function, all the computations are performed in the while loop. To bound the number of iterations of this loop; observe that $T$ is constructed so that if a match which had begun at $S[m]$ fails while comparing $S[m+i]$ to $W[i]$, then the next possible match must begin at $S[m+i+1]$ to $W[i]$, then the next possible match must begin at $S[m+i+T[i]]$ to $W[i]$. In particular, the next possible match must occur at a higher index than $m$, so that $T[i] < i$.

This fact implies that the loop can execute at most $2m$ times, since at each iteration it executes one of the

---

**Algorithm kmp_table:**

**Input:**
- An array of characters, $W$ (the word to be analyzed)
- An array of integers, $T$ (the table to be filled)

**Output:**
- Nothing (but during operation, it populates the table)

**Define variables:**
- An integer, $pos \leftarrow 1$ (the current position we are computing in $T$)
- An integer, $cnd \leftarrow 0$ (the zero-based index in $W$ of the next character of the current candidate substring)

let $T[0] \leftarrow -1$

while $pos < \text{length}(W)$ do
  if $W[pos] = W[cnd]$ then
    let $T[pos] \leftarrow T[cnd]$, $pos \leftarrow pos + 1$, $cnd \leftarrow cnd + 1$
  else
    let $T[pos] \leftarrow cnd$
    let $cnd \leftarrow T[cnd]$ (to increase performance)
  let $cnd >= 0$ and $W[pos] <> W[cnd]$ do
    let $cnd \leftarrow T[cnd]$
  let $pos \leftarrow pos + 1$, $cnd \leftarrow cnd + 1$

let $T[pos] \leftarrow cnd$ (only need when all word occurrences searched)
two branches in the loop. The first branch invariably increases i and does not change m, so that the index \( m + i \) of the currently scrutinized character of \( S \) is increased. The second branch adds \( i - T[i] \) to \( m \), and as we have seen, this is always a positive number. Thus the location \( m \) of the beginning of the current potential match is increased. At the same time, the second branch leaves \( m + i \) unchanged, for \( m \) gets \( i - T[i] \) added to it, and immediately after \( T[i] \) gets assigned as the new value of \( i \), hence new_\( m + new_i = old_m + old_i - T[old_i] + T[old_i] = old_m + old_i \). Now, the loop ends if \( m + i = n \); therefore, each branch of the loop can be reached at most \( n \) times, since they respectively increase either \( m + i \) or \( m \), and \( m \leq m + i \) if \( m = n \), then certainly \( m + i \geq n \), so that since it increases by unit increments at most, we must have had \( m + i = n \) at some point in the past, and therefore either way we would be done.

Thus the loop executes at most \( 2n \) times, showing that the time complexity of the search algorithm is \( O(n) \).

Here is another way to think about the runtime: Let us say we begin to match \( W \) and \( S \) at position \( i \) and \( p \). If \( W \) exists as a substring of \( S \) at \( p \), then \( W[0..m] = S[p..p+m] \). Upon success, that is, the word and the text matched at the positions \( W[i] = S[p+i] \), we increase \( i \) by 1. Upon failure, that is, the word and the text do not match at the positions \( W[i] \neq S[p+i] \), the text pointer is kept still, while the word pointer is rolled back a certain amount \( (i = T[i] \), where \( T \) is the jump table), and we attempt to match \( W[T[i]] \) with \( S[p+i] \). The maximum number of roll-back of \( i \) is bounded by \( i \), that is to say, for any failure, we can only roll back as much as we have progressed up to the failure. Then it is clear the runtime is \( 2n \).

III. PROBLEM SOLVING ANALYSIS

Suppose that you’re type something in your phone to text it to a friend, the phone will be checking the word which we typed and replace it with auto correct if there is a typographical error. Replacement will be have some issues that cause of much word near and other problems found to this auto correct feature. This followchart below describes the process from we type the message until it replaced when typographical error.

**FIG 2 FLOW CHART OF PROBLEM SOLVING**

A. CHECKING MESSAGE

Message that typed by the user will be checked by the auto correct feature on smartphone. It will implement KMP searching algorithm that matching what we typed and the words that refer to it, which is word that contain in the dictionary or usually used in daily life. As the system check the message, it will categorize the word that user typed into 2 categories, typographical error or graphical error. If it typographical error, so the word must replace by the word that refer in dictionary or daily use, and if not the word isn’t replace by any words. KMP algorithm is implement in this section, to know whether the words is typographical error or not, and in next section just do swapping to words refer to it.
B. REPLACE THE MESSAGE

After we get the typographical error word in our message, the system will searching for the word refer to that words to replace it. We already got a piece of the words when we matching the words we typed and the words from the system. Example, if we have typographical error in “pory”, but we want to type prt, so the replacement will be refer to pork or port, and it given option to the user which word they want if it have multi reference.

IV. IMPLEMENTATION AND ANALYSIS

A. SAMPLE MESSAGE

This message below is type to be sent to author friend, there are some typographical error and it may refer to other words.

<table>
<thead>
<tr>
<th>If I could go back in time, I would return and punch myself in the gut.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That way, I could feel the pain of hurting you before I did something silly. I am sorry.</td>
</tr>
<tr>
<td>I hipe that these flowers can make up for the horrible things that I habe done. I am truly sorry.</td>
</tr>
<tr>
<td>I thought about it and realized I was wrong. Please forgive me.</td>
</tr>
</tbody>
</table>

B. RESULT MESSAGE

This message below doesn’t have any typographical errographical error again.

<table>
<thead>
<tr>
<th>If I could go back in time, I would return and punch myself in the gut.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That way, I could feel the pain of hurting you before I did something silly. I am sorry.</td>
</tr>
<tr>
<td>I hipe that these flowers can make up for the horrible things that I habe done. I am truly sorry.</td>
</tr>
<tr>
<td>I thought about it and realized I was wrong. Please forgive me.</td>
</tr>
</tbody>
</table>

C. ANALYSIS

Let us see from the message, there are 2 typographical errographical error in hipe ant habe, which is this word doesn’t ahve meaning or this words os disrupting the meaning of the sentence. As the KMP algorithm start looking for mtaching string when they typed to the text editor messaging, they looking for the words which contain strange letter on it.

For example, the fist word we type is could, and then string matching algorithm works on system, first it checks all the letters taht contain in that words. In this part, all letter in whole sentence is viewed as a char, but we still looking for a group of char that make a words.

Second a whole letter that can’t make a word and categorized as a typographical errographical error, it will be matched by another words in word data to be replaced after that.

There are some problem when we wanna to replace the words:

1. Reference word to replace still contain another meaning to the sentence
2. Make another typographical error cause of miscasting

   This can be a bug when we used auto correct, but through it there is so much advantages to use when we used it.

Another problem that we faced it when we used autocorrect is there is typographical error that the words doesn’t contain in dictionary or rarely used in daily life. This one is a big problem that auto correct can’t handle it.

The message that have been replaced from typograhical error had contain some sentence that ready to be sent to another user. This message, or all words in that sentence have been checked by KMP algorithm and already replaced with reference words.

This algorithm more eficient than other stirng matching algorithm in this problem because KMP matching words from the prefix and it doenst need to check when whole of message has been completed, it can be checked when we typed. If we use biyer moore for this problem, it will take much time than we use KMP cause of boyer morre need whole words that have been complete to be checked.

V. CONCLUSION

Further now, Auto correct feature in smart phone have been used by so many people, it very useful to help them while text messaging. Auto correct used dictionary based and implement KMP algorithm as our string matching algorithm. Hence we got that auto correct feature, even used in notebook is implement string matching as their based and replacement the words. It very useful to use and secure to your device.
VI. ACKNOWLEDGMENT

Special thanks for my beloved family who have given their support so that I can finish this paper, and also to my friend who have giving me support to finish this paper.

REFERENCES


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 2018

Muhammad Azka Widyanto 13516127