Editing Common Mistake in Paragraph or Text Using IF2211 Strategi Algoritma

Isabella Julia Putri 1351103
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511033@std.stei.itb.ac.id
isabella.julia@s.itb.ac.id

***Abstract*—People writes and they do common mistakes on writing. The common mistakes in writing actually can be detect and by string matching. If we identify the character of common mistakes, and make a formula of it, it will be easier to find the mistakes using string matching. This will help people especially students to edit their own writing.**

***Index Terms*—Editing, Paragraph, String Matching, Common Mistake**

I. INTRODUCTION

Writing is an activity we learn since we were children. But, after some years we learn about writing, there is still common mistakes we made when we write, especially writing articles or paper. Common mistakes like spelling mistakes, puctuation mistakes and grammar mistakes still often found in paragraph we write.

Those mistakes are hard to find if we don't use program or software to help us detect our mistakes such as compiler which detect mistakes in code we made.

To detect common mistakes in writing, we can use string matching algorithm. Those mistakes has their own pattern, and string matching will help us find if there is our sentence match with the mistakes pattern.

II. BASE THEORY

A. String Matching

There are many kinds of algorithm which is used for string matching. Two of them are Knuth Morris Praat and Boyer Moore.

2.1 KNUTH-MORRIS-PRATT

Knuth –Morris Pratt Algorithm is an algorithm which is developed by D.E.Knuth, J.H.Morris and V.R.Pratt. This algorithm used as a substitute of Bruteforce Algorithm on *string matching* process. In Bruteforce algorithm, every time it didn't match with the text, the pattern will be move to the right one character. It is different from Brute Force, the information used still kept to make some movement. The algorithm uses that information to make a farther movement, not only one character. The comparison between brute force and KMP algorithm shown in the movement of the pattern based on the text position. Which in string matching, this begin in the left side of text.

Complexity of string matching algorithm is count from sum of the comparison operation which is done. The best time complexity form brute force is $O(n)$. This best case happens if the comparison operation, every letter in pattern compared with the beginning text is same. And the worst time complexity from brute force is $O(mn)$.

If we compare KMP with brute force algorithm, KMP's algorithm complexity is $O(m+n)$. KMP algorithm does the beginning process to the pattern by counting the side function. This Side function can avoid unused movement, which usually done by brute force algorithm. Side function only depends on the character in the pattern, and don't depend on the character in text.

Recall that we defined the overlap of two strings x and y to be the longest word that's a suffix of x and a prefix of y. The missing component of the KMP algorithm is a computation of this overlap function: we need to know overlap(P[0..j-1],P) for each value of j>0. Once we've computed these values we can store them in an array and look them up when we need them.

To compute these overlap functions, we need to know for strings x and y not just the longest word that's a suffix of x and a prefix of y, but all such words. The key fact to notice here is that if w is a suffix of x and a prefix of y, and it's not the longest such word, then it's also a suffix of overlap(x,y). (This follows simply from the fact that it's a suffix of x that is shorter than overlap(x,y) itself.) So we can list all words that are suffixes of x and prefixes of y by the following loop:

```
while (x != empty) {
x = overlap(x,y);
output x;
}
```

Say that shorten(x) is the prefix of x with one fewer character. The next simple observation to make is that shorten(overlap(x,y)) is still a prefix of y, but is also a suffix of shorten(x).

So we can find overlap(x,y) by adding one more character to some word that's a suffix of shorten(x) and a prefix of y. We can just find all such words using the loop above, and return the first one for which adding one more character produces a valid overlap:

```
    z = overlap(shorten(x),y)
    while (last char of x != y[length(z)])
    {
    if (z = empty) return overlap(x,y) =
empty
    else z = overlap(z,y)
    }
    return overlap(x,y) = z
```

So this gives us a recursive algorithm for computing the overlap function in general. If we apply this algorithm for x=some prefix of the pattern, and y=the pattern itself, we see that all recursive calls have similar arguments. So if we store each value as we compute it, we can look it up instead of computing it again.

2.2 BOYER MOORE

Boyer-Moore Algorithm is one of string matching algorithm. This algorithm is made by R.M Boyer and J.S Moore. The main idea of this algorithm is to find the string by comparing character start from the right character of the text. Using this algorithm, at the average will result a faster process if we compare with the other algorithm. The reason why searching from right position (the last position of the string) shown in the example :

Image 1. Example Boyer-Moore Step1

At the example above, by comparing character from the last position of the string, we ca see that character “n” on the “kanan” string didn’t match with the character “o” on “radio” string which is searched, and character “n” is never exist in string “radio”, so the string “radio” can be moved overstepping string “kanan”, so the position will be like this:

Image 2. Example Boyer-Moore Step2

In this example, we can see Boyer-Moore can make a character jump bigger so it can make the searching goes fast by only comparing more few character, and directly know that the string searched is not found, so it can moved to the next position.

B. Common Mistakes on Writing

There are 4 main types of mistake in written language: spelling, punctuation, grammar and usage.

2.3 SPELLING MISTAKES

English spelling is irregular and even many native-speaker adults have difficulties with it. Spelling mistakes do not usually prevent the reader from understanding what the writer is trying to say, but they can create a negative impression. For this reason it is advisable to try to remove them from important pieces of writing. Diligent use of a dictionary is a good alternative. For *high stakes* writing, e.g. job applications, the piece should be given to a teacher to check over. Extensive reading in English is a very good way in the longer term to learn English spelling patterns, so that mistakes are less likely.

2.4 PUNCTUATION MISTAKES

These mistakes are due to the lack of a clear understanding of what a sentence is, and they result in fragments (incomplete sentences) or run-ons ('sentences' that do not end when they should). Punctuation mistakes can often be spotted if the student reads the writing aloud. If a natural pause in the reading does not correspond with, say, a comma or a full-stop in the written text, then it is likely that the punctuation is faulty. Important writing should be given to a competent native-speaker to check.

Extensive reading, especially of non-fiction, both in English and the mother tongue, will help students understand the concept of the sentence as the basis of good writing.

2.5 GRAMMAR MISTAKES

Grammar mistakes are the next type of error commonly made. For example, people often do not choose the correct English verb tense for expressing an idea or do not use it in its correct form. They may fail to use the articles (a/the) correctly, or place words in the wrong order in a sentence.

Some grammar mistakes are easy for learners to correct themselves, particularly if they read their writing aloud. Other grammar mistakes are not easy to find, however, because the learner simply does not yet know the correct way to express an idea in English. Looking in a grammar book will not often help in such circumstances - the best thing to do is to ask a native speaker to check the writing.

In the long term most grammar mistakes will disappear by themselves, particularly if the learner does extensive reading in English.

III. Editing Text With String Matching

A. Detecting Common Mistakes with String Matching

Common mistakes in writing actually has its own pattern. For example in spelling mistakes, computers can detect mistakes in spelling with their spell checker. The word "tabel" which should be "table", word "tesr" with "test" and so on. This is the example of spell checker on email application.

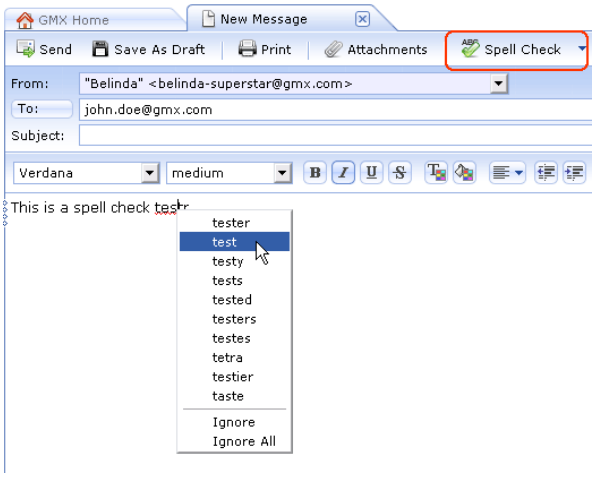

Image 3. Spell checker in email

So the mistakes in spelling has their own pattern. If we collect it and save it, it can be a resources for making software to edit common mistake in writing. This software works just like a compiler of code, in spelling mistakes it checks wheter there's spelling mistakes or no in a paragraph or text. If the the mistakes are found, it will otomaticly block the words and asking user to choose which words does he/she means.

Image 4. Example of punctuation mistake

In punctuatuion mistakes, this software works in diffterent way with spelling checker. As we know, there's a pattern how to punctuate a text. For example, fullstop "." will not places after preposition or relational term. For example, after word "in" "of" "then" "next to" will never followed by full stop. Else, question mark "?" will be place in the end of sentence which has question word such as "what" and "why". With this pattern, we can use string matching algorithm to edit a text with puctuation mistakes.

For grammar mistakes, we can see the pattern of common sentence. The subject is always placed in front of the predicate. Just a simple pattern, that can help us detect grammar mistakes on a pragraph or text. We should make a resource, which devide word into two, subject and predicate. Then looping compare two words side by side, if there is sequence of the word predicate before subject, whithout any subject before that predicat, it will give an underline on those two words. For example, sentence "I love you", will not detected as a grammar mistakes because before "love" there is subject "I". And sentence "work you today" will detected as a grammar mistakes, because there is not any subject before the word "work".

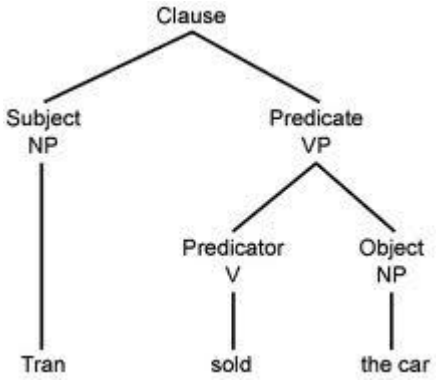

Image 5. Tree of clausa

Another mistakes that can be detected by this program is efectiveness sentence mistake. A sentence is a good sentence when it doesn't consist more than 20 words. If a sentence, having more than twenty words and not yet found a fullstop or question mark, this software will make underscore in that sentence.

String matching algorithm used in this program is the combination of KMP and Boyer Moore. For checking spelling mistakes and puctuation mistakes, Boyer Moore is more suitable. KMP used to check grammar mistakes and non effective sentence.

B. Pseudocode

```
While (not the end of text) do {
        checkSpelling();
        checkPunctuation();
        checkGrammar();
        checkEffectiveness();
}

checkSpelling(){
        if (word = misspell) {
                show alternative
        }
}
```

```
checkPunctuation(){
        temp = word before punctuation
        p = punctuation;
        if (p == "." and  ( temp == relation word or
            temp = preprosition) {
                underline(temp)
        }
}

checkGrammar(){
        temp1 = word1
        temp2 = word after word1
        temp3 = word after word2

        if ((temp1 != subject) and (temp == predicate)
            and (temp3 == subject )) {
                underline(temp2)
                underline(temp3)
        }

checkEffectiveness(){
        words = word before fullstop or question mark
        countword = number of word before fullstop or
          question mark
        if (countword >= 20) {
            underline(words)
        }
}
}
```

IV. Excess and Lack Using String Matching to Edit Text

String matching is good to detect spelling mistakes and punctuation mistakes. It will find the mistakes fast because it used Boyer Moore algorithm which check from the back of the word.

But, using string matching to edit text will need so many external resources which have to save all the mistakes pattern. It has been so many just only from spelling mistakes resource, not yet resource for checking grammar mistake.

V. Conclusion

A string matching can be used to detect common mistakes in writing, such as spelling mistakes, puctuation mistakes, grammar mistakes and uneffective sentence.

String matching algoritm which is suitable to find spelling mistakes and puctuation mistakes fast is Boyer Moore algorithm. Grammar mistakes and uneffective sentence can be detect using KNP algorithm.

References

[1] Munir, Rinaldi "Diktat Strategi Algoritma", 2009.

[2] http://www.cs.utexas.edu/~moore/publications/fstrpos.pdf access on Dec 12,2013

[3] http://www.ics.uci.edu/~eppstein/161/960227.html access on Dec 12,2013

[4] http://esl.fis.edu/learners/advice/mistakes.htm access on Dec 12,2013

[5] http://help.gmx.com/mail/messages/new/spellcheck/?si=jgSOp.1sFn38.3fnHaZ.9** (image 3 source) access on Dec 12, 2013

[6] http://money.uk.msn.com/features/how-to-spot-phishing-scam?page=3 (image 4 source) access on Dec 12, 2013

[7] http://languagetools.info/grammarpedia/clause.htm (image 5 source) access on Dec 12, 2013

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, 20 Desember 2013

ttd

Isabella Julia Putri
13511033