The Use of Pattern Matching Algorithms for Braille Documents Search Engine

David Setyanugraha and 13511003[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13511003@std.stei.itb.ac.id

***Abstract*—Braille system is a system used by blind people to read and write. This great invention has helped many blind people to learn about many literatures despite of their lack of vision. Braille consists of dot patterns which were assigned to letters according to their position within the alphabetic order. Determining alphabets from braille (or otherwise) isn't easy task to do for people who aren't used to do it. Nowadays, many commercial/non-commercial software company have implemented products to help this kind of work. Our focus here is about the search engine. The author wants to compare the performances of two search methods in braille search engine. One of the methods is search using braille pattern directly. The second method is translate first the braille and search using alphabeth. In this paper, the writer is also trying to relate some pattern matching algorithms (like KMP and Boyer-Moore Algorithm) to produce an efficient Braille Documents pattern search engine.**

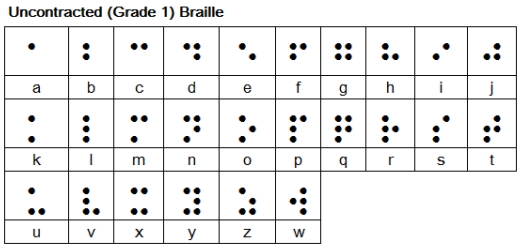
***Index Terms*—braille system, pattern matching. braille translation, dots.**

I. INTRODUCTION

Braille is one of a remarkable invention which had completely changed the way people approached education for the blind. Before the invention of Braille, blind people didn't have many opportunities for education or employment. The few existing schools for the blind were more like residential workshops, teaching basic trade skills while ignoring reading, writing and other academic studies. Braille changed all that by giving blind people an efficient method for communication and learning. Because of its profound impact on education and literacy, Braille is as important an invention as written language.[1]

There are some different versions of braille code existed. Each version of braille code is categorized by their grade and complexity. There are three different version of braille code: Grade 1 consists of 26 letters of alphabet and punctuation, Grade 2 consists of the 26 standard letters of the alphabet, punctuation and contractions, and the last, Grade 3 is used mainly in personal letters, diaries, and notes.[2]

In this paper, our focus is limited to Grade 1 Braille which is mainly used by people who just started reading braille. Grade 1 Braille is considered as the most simple alphabets-braille encoding. These fix braille patterns have been used as a standard in many braille text book, We will use those dot patterns to translate Braille Documents into an alphabets document. The other dots pattern in grade 1 is represented in picture belows.



Grade 1 Braille Alphabets pattern

There are two kinds of pattern matching algorithm that we will use in the translation. Given a braille documents and a braille/word pattern, the writer will try to find the position of this braille/word pattern in the document using pattern matching algorithm. Nowadays, braille/word pattern search is an essential task to do for a braille computer. We will find the best algorithm to do this kind of search. These two algorithms are Knuth-Morris-Pratt algorithm and Boyer-Moore algorithm. Brute force algorithm will not be discussed in this paper as the complexity algorithm is O(mn) (worst case). We want to use efficient algorithms to handle this search.

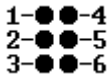
Knuth-Morris-Pratt (KMP) algorithm makes use of the information gained by previous symbol comparisons. It never re-compares a text symbol that has matched a pattern symbol.[3] The Knuth-Morris-Pratt (KMP) algorithm looks for the pattern in the text in a left-to-right order (like the brute force algorithm). KMP utillize the border function which is defined as the size of largest prefix of P[1..k] that is also a suffix of P[1..k].[4]

Besides KMP, Boyer-Moore algorithm is considered as the most efficient pattern-matching algorithm in usual applications. The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses two precomputed functions to shift the pattern to the right. These two shift functions are called the good-suffix shift (also called matching shift and the bad-character shift (also called the occurrence shift).[3]

II. SOME THEORIES

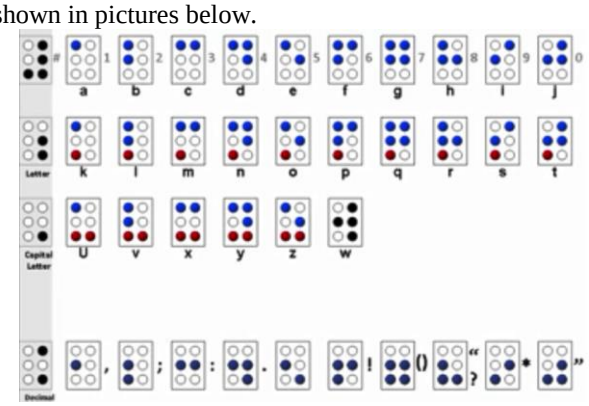
Grade 1 Braille Pattern

As explained before, we will use the unicode Braille of grade 1 in search engine. Grade 1 of Braille can be view as the most systematic braille compare to other kinds of braille. In order to understand the systematic pattern of braille, we will make a number to every dot. The convention below will be used these convention in the whole paper.



Number Conventions used in this paper

If we look it clearly, there are some systematic pattern in grade 1 braille pattern. Alphabets 'a' until 's' got almost similar dots pattern with Alphabets 'k' until 't'. For example, take a look into dots pattern of alphabeth 'a' and alphabeth 'k'. The differences between this two patterns is just only on position 3 of dot. This pattern also applies into dots pattern of alphabeth 'b' and 'l'. Alphabeth u until w also get the simillar nature works where the dots difference is on position 3 and 6 of a dots pattern. The brief similiarty pattern of every alphabeths is shown in pictures below.



Dots Pattern which shown in every characters

Braille To Text Translation Algorithm

One of the Braille translation system has been well known called The UMIST Translation system. This translation system even has been integrated into the microsoft word. This algorithm utilizes the use of Finite state machines to determine the translation performed. This algorithm may be still the basis for commercial products.[5] The main idea used state tables that perform left and right context checking, limited dictionary definitions for common but particularly irrational translations, and simple finite state machines. In this paper, the detail algorithm and performance of UMIST Translation system will not be presented here. We assume that the braille translation system has been implemented. Our focus here is to compare the performance of a braille search engine with two kinds of method. A dissertation from King (University of Manchester Institute of Science and Technology)[5] explain more about this braille translation algorithm.

String Matching algorithm

Knuth-Morris-Pratt (KMP) Algorithm

KMP algorithm utilizes a smarter way in shifting the pattern. KMP check the pattern form left to right just like brute force. The main difference is in the shifting algorithm. If missmatch is occurs in pattern P at P[j],the algorithm will shift the pattern from the largest prefix of P[1..j-1] that is a suffix of P[1..j-1]. KMP make a good use of a border function of a pattern in order to shift the pattern from one position to next available position

The example is like when the largest prefix that is a suffix from alphabeth string pattern "abacab" is about to be found. For example b(j) is the size of the largest border. We can to find b(3) from the size largest prefix of "aba". In this case, b(3) = 1 because the largest prefix can be found here : "aba" and the suffix that is also the largest prefix can be found here : "aba". For each checking method, if character of a pattern matches character of a string, then next character is checked. Otherwise, if character of a pattern doesn't matches to a character of string, pattern is shifted to the next recommended character text position based on Border function that we have made. We have generate the border function table of pattern abacab.

j	1	2	3	4	5	6
P[j]	a	b	a	c	a	b
b(j)	0	0	1	0	1	0

Border function table

KMP algorithm will produce the first index of match pattern in a text. If we can't find the pattern, then index ← -1. The full kmp algorithm is shown below:

```
Procedure KMPSearch(input m,n : integer,
input : Pattern : array[1..m] of char,
input Text: array[1..n] of char, output idx
: integer)

{function to compute the index of specified
pattern in the text using Knuth Morris
Pratt algorithm, -1 if the pattern is not
found in the text}
```

```
Declaration
i,j : integer
found : boolean
b: array[1..m] of integer

Procedure borderfunction (input m :
integer, P : array[1..m] of char, output b:
array[1..m] of integer)

{calculate b[1..m] of pattern[1..m]}
```

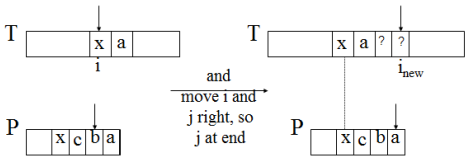
```
Algorithm
Borderfunction(m,Pattern,b)
j←0
i←1
ketemu←false
while (i<= n and not found) do
   while ((j > 0) and (P[j+1]!= T[i])) do
      j←b[j]
   endwhile

   if P[j+1] = T[i] then
      j←j+1
   endif

   if j = m then
      found ← true
   else
```

```
        i←i+1
    endif
endwhile

if found then
    idx←i-m+1
else
    idx←-1
endif
```

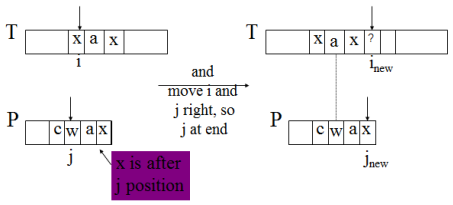
Boyer-Moore (BM) Algorithm

BM algorithm is pattern matching algorithm which utilizes the looking glass technique and character jump technique.Looking glass technique means the algorithm will find Pattern in Text by moving backwards through the Pattern starting at its end.

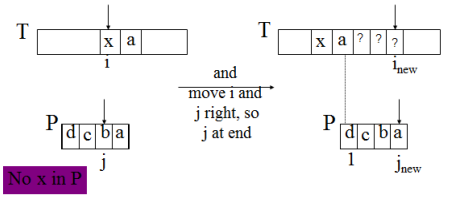
The character jump technique means when a missmatch occurs at a character in text with a character in pattern then there are 3 possible cases to handle. Case 1 occurs when there is a pattern contains x somewhere, then try to shift the Pattern right to align the last occurrence of x in Pattern with the character in text. Case 2 occurs when there is a pattern contains x somewhere but a shift right to the last occurrence is not possible, then shift pattern right by 1 character to text[i+1]. Case 3 occurs when cases 1 and 2 do not apply. If it occurs then shift Pattern to align pattern[1] with text[i+1].



Case 1



Case 2



Case 3

Based on the need of match pattern with the text, BM algorithm need a last occurrence Function. Last Occurrence Function is defined as the largest index i such that Pattern[i] == x, or -1 if no such index exists. Usually Last Occurrence Function is stored as an array.

BM algorithm will produce the first index of match pattern in a text. If we can't find the braille pattern then index ← -1. The full BM algorithm is shown below:

```
Procedure BMSearch(input Pattern : string,
Text :string ,output idx : integer)

{function to compute the index of specified
pattern in the text using Boyer-Moore
algorithm, -1 if the pattern is not found
in the text}
```

```
Declaration
m,n,i,j : integer
found : boolean
arr_index : array[1..sizeofarray] of
integer
arrp,arrs: array[1.. sizeofarray] of char
to lower case
mp : map<char,integer>

Procedure lastOccurencefunction (input
pattern :array[1..sizeofarray] of char,
output mp: map<char,integer>)

{calculate mp of pattern[1..sizeofarray]}
```

```
Algorithm
lastOccurencefunction(Pattern,mp)
while (i < n - 1) do
  if (arrp[j] != arrs[i]) then
        //character jump tecnique
        int lo = mp.get(arrs[i]);
        i = i + m - Math.min(j, 1 + lo);
        j = m - 1;
  else
        //looking glass technique
        if (j == 0) then
            found = true;
            arr_index.add(i);
            i = i + arrp.length;
        else
            i--;
            j--;

if found then
//first occurrence pattern in text
   idx←arr_index.get(0)
else
   idx←-1
endif
```

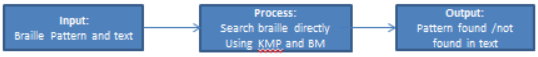
III. IMPLEMENTATION

Problem & Implementation Idea

At the beginning, we had a full braille document fully translated, we want to find a pattern "play" which is ⠏⠇⠁⠽ in a paragraph of a textbook that consist of braille document. We will implement a simple search engine to handle this kind of problem through experiments.

There are two idea to solve this kinds of problem. At last, both of these two ideas will be handled using KMP and BM algorithm.The difference is just in the type of pattern and text that we use. For each Method, the writer have made

Method 1 : Search using braille pattern directly



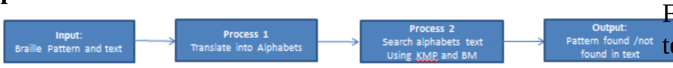
Flow chart 1

First, we got the pattern of braille (for example brailles ⠏⠇⠁⠽ (play)). All we need to do is just compare all braille text with braille pattern that we got. We use KMP and BM algorithm to implement this kind of search. Writer has made a source code in java represent to test the performance of this method which is represented by picture below.

```java
import java.util.ArrayList;

public class main {
    public static void main(String[] args) throws Exception
    {
        BMSearch bm = new BMSearch();
        String pattern = new String("[illegible]");
        String source = new String("[illegible]
        System.out.println("Braille Text : "+source);
        System.out.println("Braille Pattern : "+pattern);
        System.out.println("==================================");
        System.out.println("---Method 1 : Search using braille pattern directly--");
        System.out.println("---BM--");
        ArrayList<Integer> arrpos = new ArrayList<Integer>();
        arrpos = bm.solve_bm(pattern,source);
        for (Integer integer : arrpos) {
            System.out.println("pattern "+pattern+" found at index : "+integer);
        }

        System.out.println("---KMP--");
        KMPSearch K = new KMPSearch(pattern,source);
        K.countBorderFunction();
        K.KMPSearch();


        for (Integer integer :K.foundMatch) {
            System.out.println("pattern "+pattern+" found at index : "+integer);
        }
    }
}
```

Source Code Method 1 in Java

Method 2 : Translate first and search using alphabets pattern



Flow chart 2

The main idea of this implementation is so simple. First we translate all braille text and braille pattern to alphabets pattern. Using alphabets we got, we implement KMP and BM algorithm to search for the braille pattern in alphabets data. Similar to method 1 Writer has also made a source code in java represent to test the performance of this method which is represented by picture below.

```java
public class main {
    public static String transBraileTextToAlphabet(String t){

    public static String transBrailePatToAlphabet(String pattern){

    public static void main(String[] args) throws Exception
    {
        BMSearch bm = new BMSearch();
        String pattern = new String("[illegible]");
        String source = new String("[illegible]
        System.out.println("==================================");
        System.out.println("---Method 2 : Translate first and search using alphabets patte

        //translate pattern and source into alphabets format
        String asciipattern = transBrailePatToAlphabet(pattern);
        String asciisource = transBraileTextToAlphabet(source);

        System.out.println("---BM--");
        ArrayList<Integer> arrpos = new ArrayList<Integer>();

        //solve using BM
        arrpos = bm.solve_bm(asciipattern,asciisource);
        for (Integer integer : arrpos) {
            System.out.println("pattern "+pattern+" found at index : "+integer);
        }

        System.out.println("---KMP--");
        //solve using KMP
        KMPSearch K = new KMPSearch(asciipattern,asciisource);
        K.countBorderFunction();
        K.KMPSearch();

        for (Integer integer :K.foundMatch) {
            System.out.println("pattern "+pattern+" found at index : "+integer);
        }
    }
```

Source Code Method 2 in Java

IV. EXPERIMENTS

Testcase

Given 2 Braile test case consist of Pattern and Text. We want to search the position of certain braille pattern in the text. Two test cases has been represented

Testcase 1

Braille Pattern : [illegible]

Braille Text :

[illegible]

Testcase 2

Braille pattern : [illegible]

Braille Text :

[illegible]

Result

From the experiment that have been made toward three test cases using a java program, we got different result of each test case given between method 1 (Search using braille pattern directly) and method 2 (Translate first and search using alphabets pattern). The Summary Result of each test case is represented by table below:

Test Case	Method	Index found using KMP	Index found using BM
1	1	0 , 277	0 , 277
1	2	0 , 275	0 , 275
2	1	354	354
2	2	344	344

Summary Results Table

```
<terminated> Main [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 19, 2013 5:18:50 AM)
Braille Text : [illegible]
Braille Pattern : [illegible]
=====================================================================
---Method 1 : Search using braille pattern directly--
=====================================================================
---BM--
pattern [illegible] found at index : 0
pattern [illegible] found at index : 277
---KMP--
pattern [illegible] found at index : 0
pattern [illegible] found at index : 277
=====================================================================
---Method 2 : Translate first and search using alphabets pattern--
=====================================================================
Translation into alphabets
-------------------------
Braille Text : Braille is a method of representing characters through a pattern of raised
Braille Pattern : Braille
---BM--
pattern 'Braille' found at index : 0
pattern 'Braille' found at index : 275
---KMP--
pattern 'Braille' found at index : 0
pattern 'Braille' found at index : 275
=================================
```

Result Screenshot of test case 1 (Java)

```
<terminated> Main [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Dec 19, 2013 5:31:13 AM)
Braille Text : [illegible]
Braille Pattern : [illegible]
=====================================================================
---Method 1 : Search using braille pattern directly--
=====================================================================
---BM--
pattern [illegible] found at index : 354
---KMP--
pattern [illegible] found at index : 354
=====================================================================
---Method 2 : Translate first and search using alphabets pattern--
=====================================================================
Translation into alphabets
-------------------------
Braille Text : Much of our work on language, speech, translation, and visual processi
Braille Pattern : Google
---BM--
pattern 'Google' found at index : 344
---KMP--
pattern 'Google' found at index : 344
=================================
```

Result Screenshot on test case 2 (Java)

V. ANALYSIS

String Matching Algorithm Analysis

From the result we got in experiment, there are same result between algorithm KMP and BM. About the performance, KMP algorithm usually runs in complexity time O(n+m) which is very fast. It utillizes the information gained by previous symbol comparisons. KMP is designed that it never compares a text symbol that has matched a pattern symbol. It differ the performance of KMP with brute force. Ironically, KMP is not good to be used when the size of the alphabet increase because they got more chance of mismatch. KMP algorithm got complexity time in O(m) to calculate the border function and the main search got complexity time O(n).That is why the time complexity of KMP is O(m+n). But, KMP is faster when the mismatches of a character occur later. For the braille Documents, KMP is considered good to be used as the algorithm for searching braille pattern in the small size of braille documents because of the intelligent shift feature used by KMP which is very efficient toward braille encoded character.

Similar to KMP algorithm, Boyer-Moore Algorithm is significantly faster than Brute force. Boyer-Moore Algorithm has worst case running time in O(mn+A). However, if the the pattern is not found in the text, the complexity for BM algorithm is O(m+n). Boyer-Moore algorihm is fast when the character/alphabet is large, but it's slow when the character number is small. Boyer Moore algorithm is considered faster than KMP algorithm because the tendency of the braille characters vary is big. As every character in braille has it own dot patterns. For example, capital alphabet of 'D' has two dot patterns which are ⠠ and ⠙. That's why we can say that Boyer-Moore Algorithm is faster than KMP algorithm.

Although, Boyer-Moore Algorithm is considered better to be used in braille documents than KMP algorithm (because of the size of the variation pattern in braille document is big), but for some case (like small case), KMP is considered to be faster than Boyer-Moore algorithm. For example, in a test case (which get small variation), the performance time of KMP algorithm is faster than BM algorithm. It will be explained in Method 1 vs Method 2 algorithm analysis.

Method 1 vs Method 2 Algorithm

The experiments show that there are some difference between the index result of the pattern we got from method 1 (Search using braille pattern directly) and method 2 (Translate first and search using alphabets pattern). This result occurs to all 3 test case that we have tried. Actually, we have tested 3 test case for each method we got. But, based on our consideration, this two test cases is enough to represent the differences between two methods we got. If we take a look at the result in Summary Result Table, we will see that there are some tendencies that show the index of character found in method 1 is bigger than method 2. Based on our analysis this anomaly occurs because of the difference structure of braille and alphabets.

The main idea of method 1 is doing a search toward braille documents directly using braille pattern that we have got. This first method is considered to be a natural way of searching a pattern. The main reason why index from method 1 is bigger than method 2 is because the variation of braille character is too many. For example let consider two words "read" and "Read". These two words have a different braille translation. (Braille Translation of "read" = ⠗⠑⠁⠙ and the Braille translation of "Read" = ⠠⠗⠑⠁⠙). The Braille translation of "Read" has got 5 dot patterns and Braille translation of "read" has got 4 dot patterns. That's the reason why index number resulted from both method is different because the variation of braille.

read != Read

⠗⠑⠁⠙ ⠠⠗⠑⠁⠙

Braille of word "Read" and "read"

Method 2 do a search toward a braille documents by translate first the braille representation character into alphabet character first. This method actually implements some braille translation algorithm before doing a search. We can consider it as the extra algorithm to do before doing a pattern matching algorithm. The advantages of method 2 is we will got small variation of text as the variation of alphabet is smaller than braille character. These advantages can lead the method 2 into a faster searching algorithm. Consider the test case 2 results. The index found in method 2 test case 2 is smaller ten index than the index in method 1. As there is a big variation in a complex text book, we can utilize this special variance to get the result from KMP algorithm faster than BM algorithm (only if we don't consider the braille translation algorithm). The problem is we can not ignore easily these braile to Alphabet translation algorithm. We must consider this algorithm through entire performances because as the number of character in text increase, we will process a huge number of translations.

Both method 1 and method 2 from test case 1 and 2 show that there is the same index result between KMP and BoyerMoore algorithm. This result can lead us into an assumption that both kmp and boyer-moore algorithm can be used into the search engine. The problem is which algorithm and method that we are going to use. From the analysis of the author, Boyer-Moore algorithm with method 1 algorithm (Search using braille pattern directly) has got a better performance on most braille text book. We can't ignore the fact that there will be a huge number of character variation in a text book. Boyer-Moore algorithm works better to process this huge number of braille encoding character than KMP algorithm. Method 1 (Search using braille pattern directly) algorithm is also still considered as the best method to handle this searching method as they don't need to be translated again. Although we can't ignore the fact that for a small number (a sentence or a paragraph) of braille encoding characters, kmp with method 2 is still the best way to do a searching method. Despite of this fact, we assume that a braille

document will have a huge number of braille character encoding. That's why we suggest the use of method 1 (Search using braille pattern directly) with boyer-moore in braille search engine.

VI Conclusion

From the analysis and experiment, we conclude that Braille search engine using braille pattern with Boyer-Moore algorithm got the best performance in a braille documents with high number of contents and variances. Braille search engine using alphabets pattern is also considered to be used together with KMP algorithm to handle a braille documents when the braille documents content is small. We also get the best performance while searching a braille document using braille pattern directly than using alphabet,

References

[1] Brailleworks. Braille. December 18, 2013 (9.10 PM) http://www.brailleworks.com/Resources/BrailleAlphabet.aspx

[2] Howstuffworks. Braille. December 18, 2013 (9.00 PM) http://www.howstuffworks.com/braille.htm

[3] IGM. Boyer-Moore Algorithm. December 18, 2013 (8.50 PM) < http://www-igm.univ-mlv.fr/~lecroq/string/node14.html>

[4] International Braille Research Center, International Braille Research Center (IBRC), December 18, 2013 (8.30 PM) < http://www.braille.org/>

[5] King, A. 2001, *Text and Braille Computer Translation*, University of Manchester Isnstitute of Science and Technology.

[6] Munir,Rinaldi. 2009. *Diktat Kuliah IF3051 Strategi Algoritma*. Program Studi Teknik Informatika STEI ITB

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, 19 Desember 2013

ttd

David Setyanugraha 13511003