Simple and Fast Shiritori Program Using Boyer-Moore Algorithm

Jeremy Joseph Hanniel - 13510026[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510026@std.stei.itb.ac.id

Abstract*—Shiritori *is a kind of word chain game originated from Japan. Like most word chain game, the game is played by chaining the word; the last syllable of the previous word becomes the first syllable of the next word. There has been many* shiritori *program created on the internet using rather complex algorithms, but this paper tries to create the program by simple algorithms with Boyer-Moore algorithm at its core to speed up the search for words.

Index Terms*—Boyer-Moore algorithm, database, *hiragana, katakana, kanji, romaji

I. SHIRITORI

Shiritori (しりとり) or otherwise known as *word chain* game in English; is a word game originated from Japan. The game can be verbal or written, though usually it is played verbally since it is more fun that way. Translated literally, *shiritori* means "taking the end". The game has one main rule: players are required to name a word which begins with the final *kana* of the previous word.

Figure 1 Example of *Shiritori* Program [10]

Before going deeper into the game, there is a basic knowledge about Japanese language which is important to know. Modern Japanese language has 4 components for its writing system, which are:

- *Kanji*, which adopts from Chinese characters
- *Kana*, which is Japanese syllabary which consists of *hiragana* and *katakana*
- *Romaji*, which is Romanized version of *kanji* and *kana*, used to help foreigners understand Japanese

Kanji : 日本語
Hiragana : にほんご
Katakana : ニホンゴ
Romaji : Nihongo

Figure 2 Differences in Japanese writing system

See from Figure 1 above that even though the word is written differently, all of them actually mean the same, which is "Japanese language". *Shiritori* can be played using all 4 components above. Figure 2 below depicts an example on how the game is played.

Figure 3 *Shiritori* Game Example

Notice that *hiragana* word can be followed by *katakana* word and vice versa. If *kanji* is used, the game becomes more complex since one pronunciation can be represented by more than one *kanji* character, but results in different meaning of the word. There are more rules in *Shiritori*:

- Basic Rules
  - Two or more people take turn to play.
  - Only nouns, common pronouns, and place names are permitted.
  - The player who plays a word which ends by *n* (ん) loses, since no Japanese word begins with that character.
  - Words that have been played before cannot be repeated.

- Phrases connected by *no* (の) are permitted, but only in those cases where the phrase is sufficiently fossilized to be considered a word.

- Advanced Rules
  - Words are limited to a certain genre.
  - Instead of only using the last *kana*, the final 2 *kana* must be used as the beginning of the next word. In this case, only the first kana must not be *n* (ん).
  - Every word played must be three or more syllables long.
  - Long vowels at the end of a word can be terminated for the beginning of the next word.

In *Shiritori*, rules can be added arbitrarily on agreement among players before playing. There are always more rules to add to the excitement of the game, but for the sake of simplicity, this paper only uses the basic rules.

II. Boyer-Moore Algorithm

Pattern matching or also known as string matching is a technique to check whether one perceived sequence of characters matches with another. This technique is also used to find a sequence of characters' occurrences in another sequence. Many algorithms have been found to be able to perform such technique; one of which is Boyer-Moore algorithm.

Boyer-Moore algorithm compares the pattern (the sequence of characters to search) and the text (the sequence of characters in which the pattern's occurrence is checked) from right to left. If the character in the text which is compared with the rightmost pattern character does not occur in the pattern at all, then the pattern can be shifted by *m* positions after the text character. The idea of Boyer-Moore algorithm is illustrated in Figure 3 below.

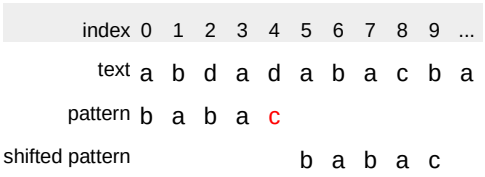


Figure 4 Boyer-Moore Algorithm Illustration [4]

Boyer-Moore algorithm first compares the last or rightmost character of the pattern with character in the text which has the same index. It is seen above that pattern[4] = 'c' does not match with text[4] = 'd'. To add to this, 'd' is not found at all in pattern. In response to this, the pattern is shifted as many as the pattern length and the comparison begins again between pattern[4] = 'c' and text[9] = 'b'. This method is called bad character heuristics. In the case where pattern[4] ≠ text[4], but text[4] = 'd' is found somewhere in the pattern, then the pattern is shifted so that the occurrence of 'd' is aligned with text[4] itself. See Figure 4 below for the illustration.

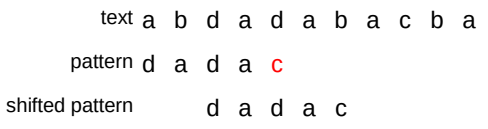


Figure 5 Bad Character Heuristics Illustration [4]

Again, comparison between pattern[4] = 'c' and text[4] = 'd' results in a mismatch. However, 'd' is found to have occurred twice in pattern, which is at index 0 and 2. Therefore, pattern is shifted so that the last occurrence of 'd', pattern[2], is aligned with text[4].

Bad character heuristics can also fail. Look at Figure 5 below.

Figure 6 Good Suffix Heuristics [4]

Illustrated in Figure 5 is that pattern[4] = text[4] = 'd' and pattern[3] = text[3] = 'a', but pattern[2] = 'd' does not match with text[2] = 'a'. If we use bad character heuristics, then the Boyer-Moore algorithm would scan the pattern for the mismatch character occurrences, which is 'a', from right to left, which would result in pattern[3] and pattern[1]. Looking at this result, the pattern is supposed to shift by -1 position, which is undesirable. Then comes another method of Boyer-Moore algorithm named good suffix heuristics.

Notice in Figure 5 that 'ab' in the pattern has matched with that in the text. Instead of shifting the pattern by -1 position like bad character heuristics would result, Boyer-Moore algorithm scans for other 'ab' occurrences in the pattern. Looking at the pattern, 'ab' is found again at pattern[1] and pattern[2], so the pattern would be shifted to align that occurrence. Having three kinds of methods to deal with various conditions, Boyer-Moore algorithm proves to be one of the more reliable pattern matching algorithms in existence.

III. Shiritori Game Databases

A. Necessary Databases

Not only is *Shiritori* simple, it is also a very good game to learn about Japanese vocabulary. Nowadays the game is not only played verbally, instead there are many *shiritori* games on the computer and internet. On digital platform, the most essential thing a *shiritori* game needs is a database containing Japanese nouns, common pronouns, and name of places. At user input, the game will search the database to find the matching word to validate the input. One simple means to do this is using a pattern matching technique.

Before getting into the details about the algorithm, it is wise to take a look at how the database is structured since

it can help reduce string comparison between the pattern which is the user input and texts which are records in the database. There are papers regarding *shiritori* which promotes string matching directly by each Japanese characters building the word, e.g. is た the same as な, is ち the same as ろ, etc. This can be done as long as the compiler that builds the program supports character encoding which supports Japanese characters such as Unicode. Even so, direct character matching works fine if the character represents the same pronunciation, but a problem arises when the word is represented in *kanji*. As explained above, one way of pronunciation can be represented in many *kanji* characters, making direct character matching a little bit complex to do. Note that the meaning of the words played is beyond our concern. What we need to check is only whether the pronunciation of the word does exist and understandable as Japanese word.

Considering the issue above, one simple suggestion is to do all string matching entirely by its pronunciation, that is, by first converting *kanji*, *hiragana*, and *katakana* into *romaji*, its Roman alphabet form. This way, 3 databases are needed; the first being conversion of *kanji* characters to *romaji*, the second being conversion of *hiragana* characters to *romaji*, and the last being conversion of *katakana* characters to *romaji*. The three databases can be represented as Table 1, Figure 6, and Figure 7 respectively.

Table 1 *Kanji* Conversion to *Romaji*

Kanji Letters	Pronunciation	
	1	2
大	dai	tai
小	shoo	
止	shi	
行	koo	gyoo
上	joo	
下	ka	ge
刀	too	jin
刃	en	
金	kin	kon

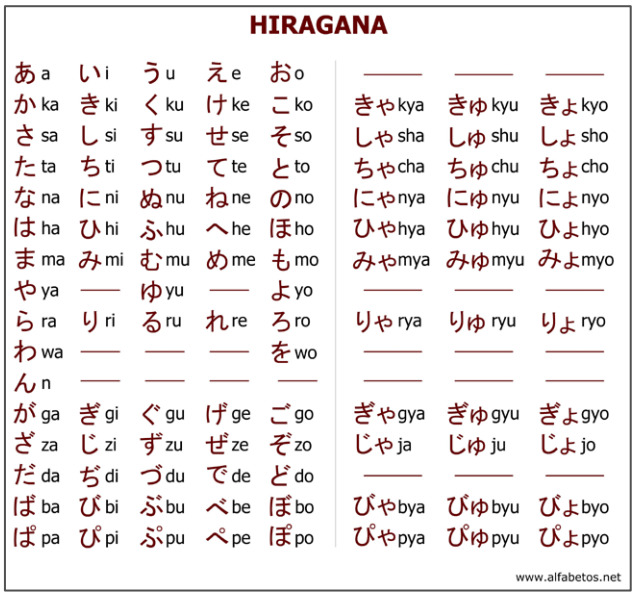


Figure 7 Conversion from *Hiragana* to *Romaji* [5]

Figure 8 Conversion from *Katakana* to *Romaji* [6]

Since *kanji* characters number high in variety, their representation in *romaji* is not disclosed here. However, to help ease the making of the database, many applications available through the internet offer conversion from *kanji* to *romaji* automatically although with uncertain preciseness. Note that if the player plays a word already in *romaji* form, then there is no need for any conversion. Finally, we need another database which consists of all Japanese words in *romaji* form. This is the database which is used to match user input and registered Japanese words whereas the mentioned three databases on the above are for the sake of converting user input only. This database is the referred to as “database” from this point on.

B. Database Indexing

Database indexing makes searches which are based on indexing criteria complete faster. In the case of Shiritori program, there are two interactions of databases used. The first is to check whether user input exists in the database, and the second is to pick a word to play as the next word. Therefore, it is useful to index the database based on the

first syllable of the word. Not only this makes validation process faster, it also makes searches for the next word faster.

IV. SHIRITORI PROGRAM

The first thing *Shiritori* program would do is to check whether the user input is registered as a valid Japanese word. This is done by converting the user input from any form to romaji. The first step of conversion is detecting whether the input is kanji, hiragana, or katakana. The main point is to check the Unicode of every inputted Japanese character. For complete reference of the Unicode representation for Japanese characters, see [7]. Here is an example algorithm to detect *kanji*, *hiragana*, and *katakana* using JavaScript [9]:

```
if (/^[\u4e00-\u9faf]+$/.test(userInput)) {
        // userInput is a kanji
} else if (/^[\u3040-
\u309f]+$/.test(userInput)) {
        // userInput is a hiragana
} else if (/^[\u30a0-
\u30ff]+$/.test(userInput)) {
        // userInput is a katakana
}
```

Take note that input conversion is done character by character. While the conversion process is running, the program should extract the first syllable of the input. This step is necessary to efficiently utilize the database which has been indexed and sorted on the first syllable of each word. This will make input validation faster because the words compared on the database are only those which start with the same syllable as the input. Other than extracting the first syllable, it is very useful to also extract the last syllable of the user input. The last syllable is used to again utilize the database index to point to the words in the database which start with the last syllable of user input. This speeds up the process of picking the next word to play.

One last step to further quicken the validation of user input, words that are going to be matched with the input can be qualified. This can be done by setting a condition that the words compared must be the same length as the user input for them to be matched. At this point, user input クルマ that has been converted to “kuruma” would only be compared with words in the database which also start with the syllable “ku” and are 6-character long, such as “kusuri”, “kurasu”, etc.

After all the process above, now is the time when string matching really begins. To match user input with records of Japanese words in the database, Boyer-Moore algorithm is chosen as it is more preferable than brute-force approach or Knuth-Morris-Pratt (KMP) algorithm. This decision is based on experiment which results in Table 1 below performed using Brute Force, KMP, and Boyer-Moore animation program [8].

Table 2 Brute Force, KMP, and Boyer-Moore Efficiency

Text (word in database)	Pattern (user input)	Total Comparison		
		Brute-Force	Knuth-Morris-Pratt	Boyer-Moore
kusuri	kuruma	3	7	1
kurasu	kuruma	4	7	1
nihongo	nichigo	3	9	3

The experiment on Table 1 tests total characters comparison using the three basic pattern-matching algorithms: Brute Force, KMP, and Boyer-Moore. It is clearly seen that KMP algorithm would not be suited to use for the game. The algorithm performs all character comparison possible between the pattern and the text, making it worse even from Brute Force algorithm. On the other hand, Brute Force algorithm performs better with fewer character comparisons than KMP, but still cannot beat the efficiency Boyer-Moore algorithm. This way, Boyer-Moore algorithm only performs many character comparisons only if the text and the pattern have the same first and last syllables.

If the search process successfully found an exact match of the user input on the database, it means that the user input a valid word. The next step is for the computer to reply to the user. Notice that the last syllable of user input has been extracted before in conversion process, so it can be used directly to jump to the records of database which contains words starting with the syllable. Here, the length of the word does not matter anymore. The next word can be chosen randomly, or if the program offers difficulty feature, than it would, for example, choose words that end with less common syllable for medium and hard difficulty. This can also utilize database index with grouping of easy, intermediate, and hard words in the database.

V. CONCLUSION

Shiritori is a word chain game that can have many ways of programming if built on computer or the internet, and this paper only describes one of many ways of how *shiritori* program works. Remember that the *shiritori* game in this paper only uses the basic rules, whereas optional and advanced features would need more calculation and more complex structure. There are even *shiritori* programs which do not use whole string matching at its heart, but graph instead.

REFERENCES

[1] "Nihongo - Japanese writing - Katakana," [Online]. Available: http://www.nihongostudy.com/aprenderj/escritura/katakana_e.php. [Accessed 15 December 2012].

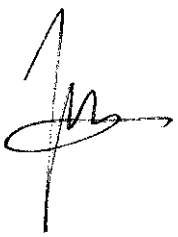
[2] "Nihongo - Japanese writing - Hiragana," [Online]. Available: http://www.nihongostudy.com/aprenderj/escritura/hiragana_e.php. [Accessed 15 December 2012].

[3] P. Seyfi, "Shiritori: The Japanese word game | Japanese LinguaLift blog," 14 December 2010. [Online]. Available: http://japanese.lingualift.com/blog/shiritori-japanese-word-game/. [Accessed 17 December 2012].
[4] H. Lang, "Boyer-Moore algorithm," 23 November 2008. [Online]. Available: http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm. [Accessed 17 December 2012].
[5] "Learn Hiragana Alphabet," Copy Left, 2008. [Online]. Available: http://www.alfabetos.net/japanese/alfabeto-hiragana/learn-hiragana.php. [Accessed 21 December 2012].
[6] B. Kirk, "Katakana," 4 April 2011. [Online]. Available: http://students.cis.uab.edu/bpkirk/Katakana.html. [Accessed 21 December 2012].
[7] "Unicode Kanji Code Table" [Online]. Available: http://www.rikai.com/library/kanjitables/kanji_codes.unicode.shtml. [Accessed 21 December 2012].
[8] "The Pattern Matching Algorithm Demo [Online]. Available: http://www.enseignement.polytechnique.fr/informatique/profs/Jean-Jacques.Levy/00/pc4/strmatch/e.html. [Accessed 21 December 2012].
[9] K. Felix, "regex - Detecting a unicode character (kanji) from a range of characters in Javascript? - Stack Overflow", 11 August 2011. [Online]. Available: http://stackoverflow.com/questions/7150633/detecting-a-unicode-character-kanji-from-a-range-of-characters-in-javascript. [Accessed 21 December 2012].
[10] "Shiritori - しりとり - 尻取り" [Online]. Available: http://shiritori.org/. [Accessed 21 December 2012].

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, 21 Desember 2012



Jeremy Joseph Hanniel
13510026