Implementation of Not So Naive and Burkhard-Keller Tree algorithms in Indonesian-Hokkien dictionary application

M A Budiman, D Rachmawati, and Charity

Department of Computer Science, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Alumni No. 9-A, Kampus USU, Medan 20155, Indonesia

mandrib@usu.ac.id, dian.rachmawati@usu.ac.id, charity.ch@yahoo.com

Abstract. In Indonesia, especially in Medan, North Sumatra, Hokkien is used in everyday life within the Chinese community. However, learning the language is still done manually. Therefore, the authors intend to create an Android-based Hokkien dictionary application. The most common typing errors found were deleting and inserting single characters, substituting and reversing adjacent characters. Exact string matching can perform a quick search, but the search will fail when the incorrect input is entered. This study aims to prove the advantages and disadvantages of each algorithm in accessing information in addition to displaying the results of the running time of both algorithms. The results showed that the search time for the Burkhard-Keller Tree algorithm increased by 72% and 96% for the Not So Naive algorithm when searching on a database containing 200 words and 400 words. The Not So Naive algorithm has a mean running time of 26.9 ms and a mean running time of 53.5 ms on the Burkhard-Keller Tree algorithm.

1. Introduction

Mastering various languages is the highest skill value [1]. The Hokkien ethnicity with their language is the majority of the Chinese population in Indonesia. The number of Chinese Hokkien speakers is estimated to be around 50 million worldwide [2]. In Indonesia, particularly Medan, North Sumatra, the population is based on ethnicity. The Chinese population there is in fourth place behind the Malays, Javanese, and Batak ethnic groups with a population of 340,320 people (Head of Distribution Division of the North Sumatra, 2015). Considering that many people do not understand Hokkien, the writer intends to take advantage of the use of smartphones to create an Android-based Hokkien dictionary application that contains Hokkien words and their translations.

The approximate string matching algorithm finds substrings that are close to a specific pattern string [3]. This algorithm conflicts with an exact string match algorithm that expects a full match. In an exact string-matching approach, the text is scanned with the aid of a window which is equal in size to the pattern length m. First, the left edges of the text and the window are aligned, then the character patterns and the characters in the window are compared. After several matches with a pattern or several mismatches, the window will be shifted to the right. The shift is made until the right corner of the window exceeds the right end of the text that is looking for [3].

The Not So Naive algorithm is among the exact string-matching algorithms which are used to carry out the string-matching process. BK Tree is a tree-based data structure used to perform spell check
based on Edit Distance (Levenshtein distance). The BK tree is also used for approximate string searches. Levenshtein distance is one of the popular edit distances. This algorithm is used to solve spelling mistakes that exist in the pattern being searched. Utilizing two strings, we can find out how similar the two strings are. The most common typing errors encountered were deletion and insertion of single characters, substitution and reversal of adjacent characters [4]. Exact string matching can perform a quick search, but the search will fail when incorrect input is entered. Approximate string matching can solve the problem of spelling errors, but there is a risk that unwanted queries will be returned, while the required ones are not displayed. In addition, the measurement of the two words is based on the Levenshtein distance, so the swap will be weighed evenly so that the linguistic nuance is also lost.

Based on the problems described, the purpose of this study is to see the advantages and disadvantages of the Not So Naive Algorithm and the Burkhard-Keller Tree Algorithm in this application. The database size indicates the amount of data stored. Here we test the performance of each algorithm by using a database containing ± 900 words, ± 400 words and ± 200 words with the length of the string to be searched in the range of 2-10.

2. Methods
In this section, we describe the search process for the Not So Naive algorithm, the Levenshein Distance method and the Burkhard-Keller Tree algorithm.

2.1. Input
The input stage is the stage where the user enters the word he wants to search, whether from Indonesian to Hokkian and vice versa.

2.2. Process
The search process on the system starts from the process of selecting the algorithm to be used whether the Not So Naive algorithm or the Burkhard-Keller Tree algorithm. For both algorithms, the user enters the input in the form of the word he wants to search. For the Not So Naive algorithm, the system will display search results, while for the Burkhard-Keller Tree algorithm, the system will display search results along with several other similar words found in the system. The running time of the search is displayed for both algorithms. When the search process fails or the word the user wants to search for is not in the database, a fail dialog box is displayed.

2.2.1. Not So Naive Algorithm. The Not So Naive algorithm was first published by Christopher Hancart in 1992. It is similar to the brute force algorithm or the Naive algorithm, but the shift in the string-matching process in the Not So Naive algorithm can be done two times for different conditions. This algorithm search process begins by first processing two cases to determine whether when the character pattern matching process will be shifted by 1 position or 2 positions to the right. Unlike the Naive algorithm or the brute force algorithm, which keeps shifting to the right 1 time.

| Table 1. Step one of matching the text with the pattern |
|---------------------------------|
| y | Text | x | Pattern |
|---|------|---|--------|
| 0 | T    | A | M      |
| 1 | I    |   |        |
| 2 | A    |   |        |
| 3 | M    |   |        |

A comparison of the 1st pattern character with the 1st text character (x [1]! = y [1]) has a mismatch. In the next experiment, the position of x will increase by 1 position to the right according to the value of the variable k (j += k), namely j = 1.
Table 2. Step two of matching the text with the pattern

| y  | Text | 0 | 1 | 2 | 3 |
|----|------|---|---|---|---|
| x  | Pattern | A | M |   |   |

A comparison of the 1st pattern character with the 2nd text character (x [1]! = y [2]) has a mismatch. In the next experiment, the position of x will increase by 1 position to the right according to the value of the variable k (j + = k), namely j = 2.

Table 3. Step three of matching the text with the pattern

| y  | Text | 0 | 1 | 2 | 3 |
|----|------|---|---|---|---|
| x  | Pattern | A | M |   |   |

A comparison of the 1st pattern character with the 3rd text character (x [1] == y [3]) matches. The three characters of the pattern match the text, so the output will be displayed.

2.2.2. Burkhard-Keller Tree Algorithm. The Burkhard-Keller Tree was first proposed by Walter Austin Burkhard and Robert M. Keller in 1973, in a paper entitled "Some Approaches to Best Match File Searching". BK-Tree can be defined as follows: any element a ∈ S is specified as root, each tree element is constructed recursively from all elements in S based on its distance from root a [5]. In building the BK-Tree, the Levenshtein distance value between the new word and the root will be calculated, and the edge d (new word, root) value is assigned. This example as shown as Figure 1

![Figure 1. Example of burkhard-keller tree](image)

Example of the BK-Tree algorithm search process:

Source string = "chamka"
The target string = "cham"
The accepted error is (T) = 2

The search starts at the root and checks the edit distance value with the input string, namely d (chamka, cham) = 2. Exploring nodes in the range [2-2, 2 + 2] = [0, 4]. Nodes with edit distance values in the range d = 0 to d = 4 will be explored. Starting from "cha" with edit distance d = 1. Edit distance d (cham, chamkuan) = 4 and d>= T so the string "chamkuan" is not added to the word list to be displayed.

2.2.3. Levenshtein Distance Method. Levenshtein distance is a measure of the similarity between two strings, the source string(s) and the target string (t) [6]. Levenshtein's algorithm works by calculating the distance from one word to another so that the degree of similarity of two words based on the distance can be determined. The greater the distance value obtained from the Levenshtein Distance operation, the greater the difference between the two strings. Levenshtein distance algorithm works as follows [6].

Step 1: Initialization
- a. Set the variable n to the length of the source string.
- b. Set the variable m to the length of the target string.
- c. Construct a matrix containing m + 1 rows and n + 1 columns.
- d. First line initialization.
- e. Initialize the first column.

Step 2: Processing
- a. Check each character s (i starts from the 1st index to the nth index).
- b. Check each character t (j starts from index 1 to index m).
- c. If s [i] equals t [j], the cost is 0, if s [i] does not equal t [j], the cost is 1.
- d. Set the cell value d [i, j] of the matrix equal to the minimum of:
  - a. d [i-1, j] + 1.
  - b. d [i, j-1] +1.
  - c. d [i-1, j-1] + 1.
- e. After the iteration steps (a, b, c, d) are completed, the distance is found in cells d [n, m].

|   | c | h | a | m | k | a |
|---|---|---|---|---|---|---|
| c | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| h | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 2 | 1 | 0 | 1 | 2 | 3 | 4 |
| m | 3 | 2 | 1 | 0 | 1 | 2 | 3 |

2.3. Output
The output stage is the stage where the system displays the search results for words and their translations to the user's screen.

3. Result and Discussion
The research uses Intel (R) Core (TM) i5-7200U CPU equipped with an NVIDIA GeForce GT 930MX GPU. The operating system used is Windows and the Java programming language version 8 is used for programming.
3.1. Not So Naive Algorithm
After comparing the running time of the Not So Naive algorithm at different database sizes, the results in Figure 2 show that the Not So Naive algorithm has an average running time of 6.4ms, 12.5ms, and 20ms for each database contains ± 200 words, ± 400 words, and ± 900 words.

![Not So Naive Algorithm Running Time Graph](image)

**Figure 2.** Not so naive algorithm running time graph over time

3.2. Burkhard-Keller Tree Algorithm
After comparing the running time of the Burkhard-Keller Tree algorithm at different database sizes, the research results in Figure 3 show that the Burkhard-Keller Tree algorithm has an average running time of 15.6ms, 26.9ms, and 53.5ms for each database containing ± 200 words, ± 400 words and ± 900 words.

![Burkhard-Keller Tree Algorithm Running Time Graph](image)

**Figure 3.** Burkhard-keller tree algorithm running time graph over time
4. Conclusion
The search times for each algorithm for databases with a size of ± 200 words, ± 400 words, and ± 900 words were compared. The results of running time show that the search time for the Burkhard-Keller Tree Algorithm has increased by 72% from the size of the database containing ± 200 words and by 99% of the size of the database containing ± 400 words.

The results showed that the Not So Naive algorithm has a mean running time of 20 ms and a mean running time of 53.5 ms on the Burkhard-Keller Tree algorithm for databases containing ± 900 words. There is a sizable difference between running time on the Not So Naive algorithm and running time on the Burkhard-Keller Tree algorithm. In the Burkhard-Keller Tree algorithm, the process of building a tree before performing the search is calculated so that the running time results increase as the size of the database increases. The Not So Naive algorithm and the Burkhard-Keller Tree Algorithm can be applied to the Indonesian-Hokkien Dictionary Application.

References

[1] Al-Khowarizmi A-K, Fauzi F, Sari I P and Sembiring A P 2020 The Effect of Indonesian and Hokkien Mobile Learning Application Models Journal of Computer Science, Information Technology and Telecommunication Engineering 1 1–7
[2] Gapur A, Siregar D S P and Pujiono M 2018 Language Kinship Between Mandarin, Hokkien Chinese And Japanese (Lexicostatistics Review) Aksara 30 301
[3] Hakak S I, Kamsin A, Shivakumara P, Gilkar G A, Khan W Z and Imran M 2019 Exact String Matching Algorithms: Survey, Issues, and Future Research Directions IEEE Access 7 69614–37
[4] Hall P A V and Dowling G R 1980 Approximate String Matching ACM Computing Surveys 12 381–402
[5] Baeza-Yates R and Navarro G Fast approximate string matching in a dictionary Proceedings. String Processing and Information Retrieval: A South American Symposium (Cat. No.98EX207)
[6] Haldar R and Mukhopadhyay D 2011 Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach arXiv.org