Trie based dictionary for Chinese

Daixuan Chen Shulin Wen Fan Zhang

Computer Science Computer Engineering Computer Engineering

Stevens Institute of Technology Stevens Institute of Technology Stevens Institute of Technology

***Abstract* —** **This paper proposes a highly efficient technique for the retrieval of Chinese characters by implementing the trie data structure in the search process. The trie data structure enables effective partitioning and storage of Chinese characters, ensuring they can be easily accessed from root to leaf nodes. Concurrently, the integration of a hashmap offers direct mapping, which expedites the retrieval of these characters. We present both pinyin and shuangpin methods as viable strategies for searching Chinese characters. Through rigorous experimental analysis, it is demonstrated that the shuangpin method outperforms the pinyin method in terms of insertion time, search time, and space utilization.**

***Key words*—Tries, Hashmap, Chinese, Java**

Ⅰ. Introduction

Research indicates that the scale of data and databases in various industrial sectors is experiencing exponential growth on a daily basis. Concurrently, there is a rising number of web search engines, such as Google, emerging to facilitate information retrieval from the vast expanse of the Internet. The Internet's unparalleled growth and application are reshaping the operational patterns of businesses [1]. A search engine uses an inverted index to respond to queries [2].

The TRIE algorithm, extensively utilized in information processing. Its name, 'TRIE', originates from the word 'retrieval'. [3] coined this term for TRIE memory and [4] introduced the classical algorithm on TRIE. The TRIE algorithm matches a pattern by segmenting all patterns and storing them within a tree structure. Each child node shares a common prefix. Pattern matching involves identifying matches from the root to a leaf node. One of the advantages of trie is that it allows fast access to entries with common prefixes, which is very important when searching similar entries [5].

The skill of hashing is also employed to maintain the mapping since it is always supposed to be an efficient accessing method of data from key sequences to Chinese characters [6]. It facilitates direct retrieval of Chinese characters from the map.

In this paper, we employ an efficient data structure, leveraging both a trie and a hashmap, to store and search for Chinese words. Our paper is structured as follows. Section 1 presents our ideas for searching Chinese characters at the beginning of the project. Section 2 discusses the classes and its functions of the pinyin and shuangpin methods. Section 3 elaborates on the experimental results and subsequent correlation analysis. Finally, Section 4 draws conclusions from our study, outlining the advantages and disadvantages of the pinyin and shuangpin methods discussed.

Ⅱ. Problem Statement

There are two possible ways to implement a trie data structure, pinyin and strokes.

(1) The Chinese characters are broken down into pinyin based on their pronunciation and stored in a similar way to English words by using this method.

There are two examples that show how a Chinese character is broken down into pinyin.



Fig. 1 “Hello” in Chinese, with tone, which can be decomposed as “ni” and “hao”



Fig. 2 “Chinese word” in Chinese, with tone, which can be decomposed as “han” and “zi”

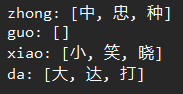


Fig. 3 Sample output

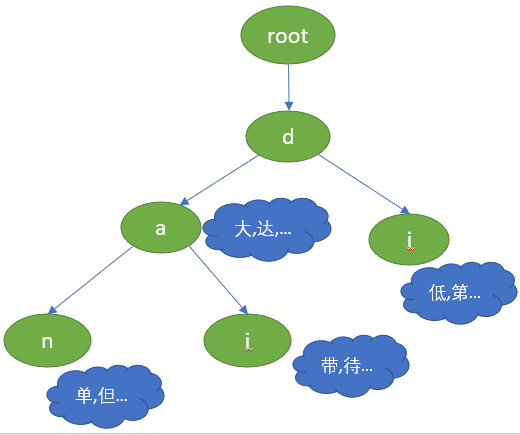


Fig. 4 Data structure of a trie

(2) The Chinese characters are broken down into strokes in order, and each stroke becomes a node in the trie data structure.

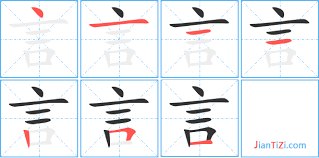


Fig.5 “Language” in Chinese with strokes

A Chinese character can be decomposed of the order of strokes, for example, dot, horizontal line, vertical line, left-failing horizontal, right-failing horizontal. In our project we chose pinyin to implement the tries. The following are the reasons:

1. More dataset available
2. More optimization approaches
3. More understandable
4. More GitHub projects and Java libraries that may be useful

(2) The optimization: Shuangpin Trie

**Introduction to Shuangpin**

Based on the pinyin trie, we improve its performance by implementing Shuangpin Trie. Shuangpin is still based on the Pinyin System, but instead using at most six letters, it represents each Pinyin with only two letters. Shuang means “double” in Chinese. It breaks down each Pinyin into its vowel and consonant components, each of which represented by a single letter. For example, the Pinyin for “Shuang” is divided into “sh” for the vowel and the “uang” for the consonant. In Shuangpin, “u” represents “sh” and “l” represents “uang”. Thus, “Shuang” becomes “ul” in Shuangpin, with a much shorter length.

By adopting the Shuangpin keyboard layout diagram, designed to assign the orange section to vowels and the blue section to consonants [7], we leverage the efficiency of using single-letter keys to represent both vowels and consonants. To illustrate, let's consider the word "Shuang" once again: the vowel "sh" is encoded by the key "u," while the consonant "uang" is represented by the key "l." Consequently, the character "双" undergoes a remarkable transformation, as its pinyin counterpart, which traditionally spans a length of 6, is condensed into a concise shuangpin representation of just 2 characters. As evident from the analysis, leveraging the inherent pattern within the pinyin system, which involves the interplay between vowels and consonants, allows for the utilization of a compact trie structure with a height of three. Thus, the memory usage and search/insert time could be significantly reduced.

Fig.6 Shuangpin layout diagram

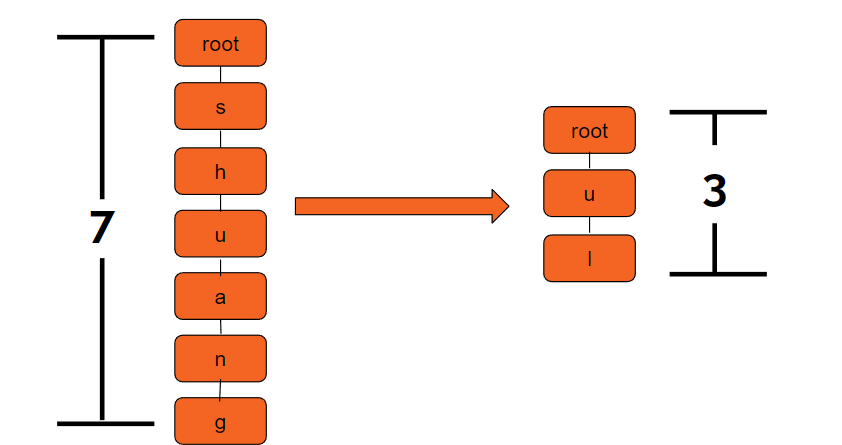


Fig.7 The advantage of shuangpin

**The transformation from pinyin to shuangpin**

Even though knowing that the pinyin has a certain pattern, we need to consider it carefully. Because there are different combinations depending on the length of vowels and consonants.

The pinyin system consists of 23 vowels and 33 consonants, which can be combined to form different pinyin representations to phonetically express Chinese characters. In terms of length, vowels are categorized into two types: monophthongs with a length of 1, and diphthongs with a length of 2. The consonants are classified into four categories: monophthongs (length of 1), diphthongs (length of 2), triphthongs (length of 3), and nasal consonants (length of 4). To enhance algorithm efficiency, this study adjusts the traditional pinyin system by classifying the vowels and consonants based on length rather than strict definitions. The following table illustrates these classifications.

|  |  |  |
| --- | --- | --- |
| Category | Content | Length |
| single vowels | q, w, e, r, y, o, p, a, s, d, f, g, h, j, k, l, z, x, c, b, n, m | 1 |
| double vowels | sh, ch, zh | 2 |

Table.1 Classification of vowels

|  |  |  |
| --- | --- | --- |
| Category | Content | Length |
| monophthongs | e, u, i, o, a, v | 1 |
| diphthongs | iu, ei, ue, ve, un, uo, ie, ai, en, an, ou, ua, ia, ao ,ui, iu | 2 |
| Triphthongs | uang, ong, eng, ang,uai, ing, iao, ian | 3 |
| Nasal consonants | iong, uang, iang | 4 |

Table.2 Classification of consonants

Pinyin encompasses a wide range of combinations, totaling at least 410 variations ranging from 1 to 6 characters in length. However, based on the length of the pinyin, we can categorize them into six fundamental types, each with its own combinations. For pinyin with a length of 1, we can confirm that it consists of a monophthong, as a vowel cannot constitute a pinyin on its own according to the rules while consonant can. When the length of the pinyin is 2, there are two possibilities: it could be a combination of a single vowel (length of 1) and a monophthong (length of 1), or a diphthong (length of 2). The comprehensive set of possibilities is presented in the following table.

|  |  |
| --- | --- |
| Length of pinyin | combination |
| 1 | a monophthong |
| 2 | a single vowel + a monophthong |
| a diphthong |
| 3 | a single vowel + a diphthong |
| a double vowel + a monophthong |
| a triphthong |
| 4 | a single vowel + a triphthong |
| a double vowel + a diphthong |
| 5 | a single vowel + a nasal consonant |
| a double vowel + a triphthong |
| 6 | a double vowel+ a nasal consonant |

Table.3 The combination of pinyin

Given the established combinations, we have observed three scenarios in which the pinyin does not contain any vowels, which contradicts the pattern mentioned earlier that pinyin consists of a vowel and a consonant. In the Shuangpin system, this phenomenon is referred to as "null vowel" [7]. To accommodate the Shuangpin format, we have devised three rules to handle null-vowel situations. When the pinyin has a length of 1, we simply duplicate the pinyin, resulting in a length of 2. If the length is 2, we leave it unchanged. For pinyin with a length of 3, we extract the initial letter and combine it with the corresponding projected key to form a Shuangpin with a length of 2.

With the comprehensive handling of various scenarios, the transformation of pinyin into shuangpin has become attainable.

Ⅲ. Design Process

(1) Using pinyin to load data

In this method, a Trie (prefix tree) data structure is used to efficiently store and query Chinese characters based on their pinyin (Romanized representation). Here is a brief description of the approach:

1. Define a TrieNode class, which represents a node in the Trie. Each TrieNode contains a mapping of child nodes and a mapping of pinyin to a list of corresponding characters.
2. Define a ChineseTrie class, which contains the root TrieNode. The ChineseTrie class has methods for inserting a character with its associated pinyin and searching for characters based on their pinyin.
3. To use the ChineseTrie class, insert pinyin and character pairs into the Trie, and then query the Trie with a given pinyin to retrieve the associated characters.

TrieNode class: The TrieNode class represents a node in the Trie. Each TrieNode has the following attributes:

1. children: A mapping of characters to their respective child TrieNodes.
2. pinyinToCharacters: A mapping of pinyin strings to a list of Chinese characters that share the same pinyin.
3. isWordEnd: A boolean flag indicating whether the current TrieNode represents the end of a valid pinyin string.

ChineseTrie class: The ChineseTrie class contains the Trie's root node and provides methods for inserting and searching characters based on their pinyin. Here are the main methods:

1. Insert (String pinyin, Character character): This method is responsible for inserting a character with its associated pinyin into the Trie. It starts at the Trie's root node and iterates through the characters in the pinyin string. For each character, it checks whether a child node with the same character exists; if not, it creates a new TrieNode and adds it as a child. It then proceeds to the next character in the pinyin string, using the child node as the new starting point. When the end of the pinyin string is reached, it sets the isWordEnd flag of the final TrieNode to true and adds the Chinese character to the pinyinToCharacters mapping of that TrieNode.
2. Search (String pinyin): This method queries the Trie for Chinese characters based on their pinyin. It starts at the Trie's root node and iterates through the characters in the pinyin string. For each character, it retrieves the corresponding child TrieNode and proceeds with the next character. If a child TrieNode is not found at any point, the search is terminated, and an empty list is returned. If the Trie traversal reaches the end of the pinyin string and the final TrieNode's isWordEnd flag is true, it returns the list of Chinese characters associated with the given pinyin in the pinyinToCharacters mapping of the final TrieNode. If the isWordEnd flag is false, an empty list is returned.

The time complexity for insert and search function is O(L). The space complexity for this datastructure is O(nL).

To use the ChineseTrie class, create a new instance and insert the pinyin and character pairs as required.

(2) Implementation of shuangpinTrie

Since the Shuangpin Trie is an optimization of the Pinyin Trie, the overall structure remains consistent, with the addition of efficiency-enhancing features. Therefore, it is unnecessary to provide redundant details regarding the structure. We will focus on introducing the specific features that contribute to its optimization.

Building upon the pinyin trie data structure, we have introduced the getShuangpin method to establish a hashmap called allShuangpin, which captures the mapping between pinyin and shuangpin representations. This enables us to conveniently retrieve and utilize shuangpin format during insertion and searching operations. The implementation of this method involves the utilization of a hashset: vowelSet, which encompasses single vowels, and a hashmap: consonantMap, the value of which encompasses consonants and double vowels. The purpose of consonantMap is to provide key projections for double vowels, while single vowels can be directly represented. Here is a brief description of the approach:

**Create a hashmap with key is pinyin and value is shuangpin**

1. we first define a HashSet named vowelSet, which includes all the monophthongs and diphthongs. This set is used to check if a given pinyin belongs to a monophthong.Next, we create a HashMap named consonantMap to store the mapping between consonants and shuangpin keys. Each key-value pair represents a consonant and its corresponding shuangpin key. For example, "sh" corresponds to the key "u", "ch" corresponds to "i", "zh" corresponds to "v", and so on.
2. Then, we define a HashMap named allShuangpin and initialize it by calling the getAllShuangpin() method. Inside the getAllShuangpin() method, we iterate through each key-value pair in consonantMap and retrieve the corresponding shuangpin representation based on the pinyin. We store the pinyin as the key and the shuangpin as the value in the allShuangpin map.
3. The getShuangpin method takes a pinYin (pinyin) string as input and returns its corresponding shuangpin representation. The method first checks the length of the input string to handle different cases. If the length is 1, it duplicates the character to create the shuangpin. If the length is 2, it checks if the input is a consonant pair already present in the consonantMap. If so, it returns the input itself. Otherwise, it separates the vowel and consonant parts of the pinyin and retrieves the corresponding shuangpin representation from the consonantMap.
4. For pinyin with a length of 3 or more, the method follows a similar logic. It checks if the input is a consonant pair in the consonantMap, and if so, it returns the corresponding shuangpin representation. If the input starts with a vowel pair (length 2), it combines the shuangpin representations of the vowel and consonant parts. If the input starts with a single vowel (length 1), it combines the vowel and consonant parts in the shuangpin format.
5. The getAllShuangpin method creates a HashMap named allShuangpin and populates it with the mappings between pinyin and shuangpin. It iterates through each entry in the extractCol map, which contains the pinyin and its corresponding column pair. It retrieves the pinyin from the column pair, calls the getShuangpin method to obtain the shuangpin representation, and stores the mapping in the allShuangpin map.
6. The getShuangpin and getAllShuangpin methods work together to generate a HashMap named allShuangpin that contains the complete mapping between pinyin and shuangpin representations for the given dataset.

The time complexity of the getShuangpin method is O(1) for pinyin strings with a length of 1 or 2, as the operations performed are constant time. For pinyin strings with a length of 3 or more, the time complexity is O(1) as well since the number of cases handled remains constant.

The time complexity of the getAllShuangpin method is O(N), where N is the number of total pinyin. The method iterates through each entry and performs constant time operations for each entry.

The space complexity of the getShuangpin method is O(1), as it uses a constant amount of space to store the variables and return the shuangpin string.

The space complexity of the getAllShuangpin method is O(N), where N is the number of total pinyin. The method creates a HashMap named allShuangpin that stores N mappings between pinyin and shuangpin representations.

In a nutshell, by initializing vowelSet, consonantMap, and allShuangpin, we obtain a HashMap named allShuangpin that contains the complete mapping between pinyin and their respective shuangpin representations. This HashMap is used in the implementation to convert pinyin into shuangpin, providing a more efficient data structure and search capability.

**Use cache to improve searching efficiency**

The cache is a data structure used to store previously computed results for faster retrieval. It is implemented as a HashMap where the key is the pinyin string and the value is the corresponding search result.

When a search is performed, the method first checks if the pinyin string is already present in the cache. If it is, the cached result is returned directly, avoiding the need for further computations. This significantly improves the performance of subsequent searches for the same pinyin.

If the pinyin is not found in the cache, the method proceeds with the search operation. It retrieves the corresponding shuangpin representation from the allShuangpin hashmap and traverses the shuangpin trie to find the matching node. If the node is not found, an empty list is returned.

Once the search is complete and the result is obtained, it is stored in the cache for future use, associating it with the corresponding pinyin. This caching mechanism reduces the computational overhead of repeated searches and enhances the overall efficiency of the algorithm.

Ⅳ. Results and Analysis

We use chinses pinyin data set collected from the internet, combined with Pinyin4j, which is a popular Java library used for converting Chinese characters to their Pinyin representations. The efficiency of this trie data structure is determined by its reading speed (the time required to read the entire database) and querying speed (the time required to query a specific character). The reading speed is measured by recording the time loading a txt file that contains 3500 Chinese characters and 9901 characters. The searching speed is measured by implementing a random retrieval of a Chinese character from the constructed Trie and test the average search speed.

The time required for original pinyin Trie to insert 3,500 commonly used Chinese characters is 25 milliseconds and for a dataset with 9901 Chinese characters is 39 milliseconds.

The average time needed to randomly search for a pinyin in the Trie and find the corresponding character is 477.7 nanoseconds and for a dataset with 9901 Chinese characters is 594.0 nanoseconds.

Ⅴ. Conclusion

In summary, the pinyin-based Trie method offers an efficient and space-saving way to store and query Chinese characters based on their pinyin. However, it has some limitations, such as incomplete pinyin-to-character mappings, reliance on accurate pinyin input, and potential memory overhead for large character sets. The following is its advantages and disadvantages.

Advantages:

1. Efficient querying: Trie data structure allows for efficient querying of characters based on their pinyin, with a time complexity of O(n), where n is the length of the pinyin.
2. Prefix searching: Trie supports prefix searching, allowing it to find all characters sharing a common pinyin prefix, if required.
3. Space efficiency: Trie stores pinyin in a space-efficient manner by sharing common prefixes across multiple characters.

Disadvantages:

1. Limited scope: The implemented Trie does not include a complete pinyin-to-character mapping, and the user needs to add all required pinyin and character pairs manually.
2. Pinyin accuracy assumption: The implementation assumes that the input pinyin is accurate and does not handle cases with incorrect pinyin input.
3. Scalability: The size of the Trie can grow significantly with many characters, potentially leading to increased memory usage.

References

1. D'Mello, D. A., & Ananthanarayana, V. S. (2010). Dynamic selection mechanism for quality of service aware web services. Enterprise Information Systems, 4(1), 23-60.
2. Yang, L., Xu, L., & Shi, Z. (2012). An enhanced dynamic hash TRIE algorithm for lexicon search. Enterprise Information Systems, 6(4), 419-432.
3. Fredkin, E. (1960). Trie memory. Communications of the ACM, 3(9), 490-499.
4. Knuth, D. E. (1997). The art of computer programming (Vol. 3). Pearson Education.
5. Lam, Y. K., & Huo, Q. (2005, August). A data structure using hashing and tries for efficient chinese lexical access. In Eighth International Conference on Document Analysis and Recognition (ICDAR'05) (pp. 506-510). IEEE.
6. Chang, C. C., Chen, T. S., & Lin, Y. (2000, November). An efficient accessing technique of Chinese characters using Boshiamy Chinese input system. In Proceedings of the fifth international workshop on Information retrieval with Asian languages (pp. 61-67).
7. Flypy: A Shuangpin Input Method for Chinese. Retrieved from https://www.flypy.com/