# Project report: Chinese Words Trie

Group members: Daixuan Chen, Shulin Wen, Fan Zhang

**Introduction**

This project is for course 593 about topics Tree or HashMap based dictionary for other languages.

A trie is a tree-like data structure used for efficient storage and retrieval of data, particularly for strings. When applied to Chinese words, a trie can be used to store and search Chinese text in a compact and efficient manner.

Our objective is to create an efficient data structure for storing and searching Chinese words using a trie.

**Pinyin Tries**

There are two possible ways to implement a trie data structure.

1. Method 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.

Here 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”

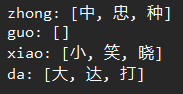


Fig. 3 Sample output

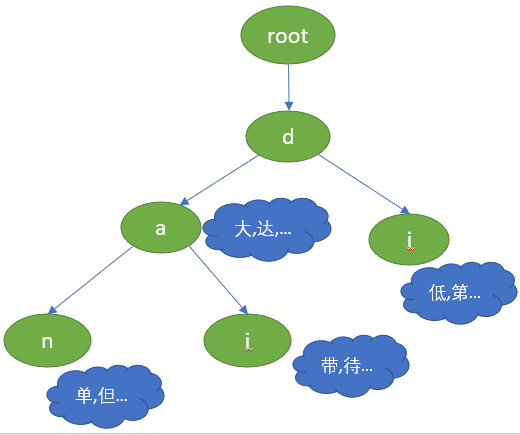
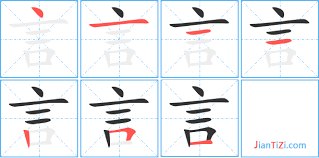


Fig. 4 Data structure of a trie

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



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:

* More dataset available
* More optimization approaches
* More understandable
* More GitHub projects and Java libraries that may be useful

**Methods**

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:

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.

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.

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:

children: A mapping of characters to their respective child TrieNodes.

pinyinToCharacters: A mapping of pinyin strings to a list of Chinese characters that share the same pinyin.

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:

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.

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).

**Usage**:

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

2. **Using stroke order (bihua) to load data**

To implement a trie data structure for retrieving Chinese characters based on bihua (stroke order), we test a stroke-based trie:

We analogize the basic strokes composing Chinese characters to the letters in English, and then incorporate them into a trie data structure.

Each type of the stroke can be encoded by a number, and use bihuaMap to convert it.

The node structure for the trie consist of the following fields:

A boolean flag to indicate if the current node represents the end of a word.

An array of child nodes, each corresponding to a stroke of the Chinese character.

An array of integers to store the frequency or other metadata of the Chinese character that ends at the current node.

Define a function to insert a Chinese character into the trie based on its bihua sequence. The function takes the following steps:

Start at the root node and traverse the trie based on the stroke order of the Chinese character.

For each stroke, check if there is a child node corresponding to the stroke. If not, create a new node and add it as a child of the current node.

Once all strokes have been processed, set the boolean flag of the last node to indicate that a Chinese character ends at this node, and store any metadata associated with the character in the node.

Define a function to search for a Chinese character in the trie based on its bihua sequence. The function takes the following steps:

Start at the root node and traverse the trie based on the stroke order of the Chinese character.

For each stroke, check if there is a child node corresponding to the stroke. If not, return null to indicate that the character is not in the trie.

Once all strokes have been processed, check if the boolean flag of the last node is set. If so, return the metadata stored in the node. Otherwise, return null to indicate that the character is not in the trie.

**Experimental Design**

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.

**Results**

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.

**Discussion and 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:

* 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.
* Prefix searching: Trie supports prefix searching, allowing it to find all characters sharing a common pinyin prefix, if required.
* Space efficiency: Trie stores pinyin in a space-efficient manner by sharing common prefixes across multiple characters.

Disadvantages:

* 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.
* Pinyin accuracy assumption: The implementation assumes that the input pinyin is accurate and does not handle cases with incorrect pinyin input.
* Scalability: The size of the Trie can grow significantly with many characters, potentially leading to increased memory usage.