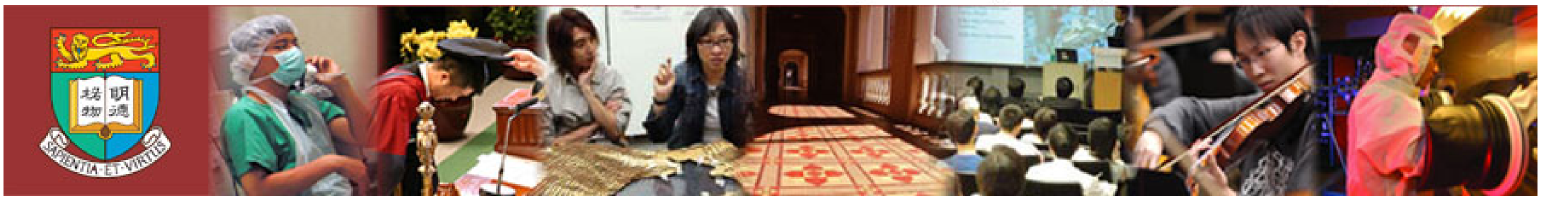

COMPUTER STUDIES PUBLICATION

FORMAL SPECIFICATION AND IMPLEMENTATION OF A CHINESE DICTIONARY

K.P. Chow and C.T. Hung

Technical Report TR-87-06

April 1987

CENTRE OF COMPUTER STUDIES AND APPLICATIONS
UNIVERSITY OF HONG KONG
POKFULAM ROAD
HONG KONG

Formal Specification and Implementation of a Chinese Dictionary[*]

K. P. Chow and C. T. Hung[**]

Computer Studies
University of Hong Kong

Abstract

Dictionary organization is a fundamental problem in computer science. Numerous works have been done on specifying and implementing the English dictionary. On the other hand, there is no published work for the Chinese dictionary. Since the Chinese dictionary is quite different from an English dictionary, we present here a formal specification and an implementation for a Chinese dictionary.

A Chinese dictionary consists of four components: a set of zis (ideographs), a zi dictionary, a set of cis (words), and a ci dictionary. The set of zis and the set of cis is huge. The number of zis is more than 10,000 and there are about 6000 commonly used ones. The number of commonly used cis is more than 20,000. On the other hand, the zi dictionary and the ci dictionary is input method dependent. We present here a input method independent formal specification and an implementation based on the pinyin input method. The whole dictionary is stored in less than 700 Kbytes.

[*]This research is partially supported by the Hong Kong and China Gas Company Limited Research Grant.

[**]An earlier version of this paper was prepared while the first author was visiting University of California, Los Angeles in 1985.

1. Introduction

Dictionary organization is a fundamental problem in computer science. For practical use (e.g., for spelling checking) an efficient implementation of the dictionary would save resources. For Chinese information processing, such as Chinese word processing, an efficient dictionary organization is very important. Since the processing of Chinese characters or words is quite different from English, it is worthwhile to study the implementation of a Chinese dictionary.

Before looking at the dictionary, let us consider different input methods for Chinese. For an English dictionary, a string of characters represent a word. On the other hand, there are over several thousands of Chinese characters, each character is itself represented by a code-string. Therefore, a character dictionary is required. In addition, we also need a word dictionary to store Chinese words. Different methods have been used for Chinese character input. Some of the popular ones are: codes used in telegram, 4-corner code (四 角 號 碼), pinyin (併 音), and cang-xie (倉 頡). Table 1 gives examples for these different methods.

Character	Pinyin	Cang-xie	Telegram code	4-corner
阿	a	NLMNR	0759	6102_0
哀	ai	YRHV	0755	0073_2
愛	ai	BBPE	1947	2044_7
挨	ai	QIOK	2179	5303_4
安	an	JV	1344	3040_4
昂	ang	AHUL	2491	6072_7

Table 1. Examples of Different Input Methods

This paper emphasizes on the implementation of the dictionary instead of input methods. Throughout the rest of the paper, pinyin is used for illustration, while the general idea can be applied to any method.

Chinese words are divided into two categories according to the way they can be represented: permanent and transient. The permanent words are listable, though there is a large number of them. On the contrary, there are unlimited number of transient words. For examples,

一個人
兩個人
三個人
十個人

The permanent word dictionary requires an efficient implementation and its design is presented first. The transient words can be generated from regular expressions, presented last briefly. One important point is that the regular expression not only generates the legal transient words, but also generates some illegal words!

2. Terminology

Each Chinese character is stored as a bitmap (ideograph). Each ideograph is called a *zi* (字). The collection of all the zis form the *zi universe* (*ZIu*). Each zi is addressed by an index. Let *charset* be the alphabet for input. Its actual value depends on the particular input method being used. For example, 0-9 for 4-corner code, A-Y for (cang-xie), and about 50 latin pinyin symbols[1] for pinyin. A *code* is a non-empty string from charset. *Codeset* is the set of all possible input codes. In general, each code may correspond to zero, one or more zis. In some input methods, each zi has an unique code, e.g., cang-xie. The *zi dictionary* (*ZID*) is a data structure implementing this relationship. Besides a zi dictionary, there is also a Chinese word dictionary. A Chinese word is a string make up of two or more zis, it is called a *ci* (詞). The collection of common used cis form the *ci universe* (*CIu*). The relation between the input codes (zis) and their corresponding cis is incorporated in the *ci dictionary* (*CID*). Table 2 summarizes these definitions.

Ideograph :	A chinese character (a bitmap).
ZIu :	Set of all ideographs.
charset :	Character set of input code (e.g. Pinyin characters and 4-corner digits)
Codeset :	$charset^+$
ZICODE :	An input code of a ZI. An element of Codeset.

[1]Usually the English alphabet is used to represent the pinyin. Since the number of pinyin symbols is larger than 26, some pinyin symbols are represented by more than one letter.

ZICODEs :	Set of input codes of ZIs. A subset of Codeset.
ZID :	ZICODEs $\times 2^{ZIu}$. A relation from set of input codes to the set of ideographs.
CI :	An ordered-tuple of ZIs (Ideographs).
CIu :	$\bigcup_{i=1}^{N} ZIu^i$ where ZIu^i means the set of ordered *i*-tuple of ZIu, N is the maximum number of zis in a ci. N is implementation dependent.
CICODE :	An input code of CI. An element of $ZICODEs^+$.
CICODEs :	Set of all codes of CIs.
CID :	CICODEs $\times$ CIu.

Table 2. Some Definitions

3. Specification of Chinese Dictionary

This section describes a formal algebraic specification for the Chinese dictionary. We first describe different operations on the Chinese dictionary. Then the formal algebraic specification is presented. The method by Guttag and Horning [4] is used.

3.1. Zi dictionary

Following is the set of operations that can be performed on the zi dictionary. *New_ZID* is the function that creates an empty zi dictionary. A new zi with its corresponding code is added to the dictionary by the function *add_zi*. With *add_zi*, the new ideograph is added to the ZIu, and an index is returned. The index is then linked with the code and added to the zi dictionary. *Is_zi* is a function to check if a given zi code corresponds to an existing zi in the dictionary. It returns true if the zi has been added to the dictionary before. Finally, *retrieve_zi* is to retrieve the ideograph(s) corresponding to the given code in the zi dictionary. An empty set is returned if no zi has been added for that code.

Table 3 is the formal definition of the zi dictionary:

Domain definitions

New_ZID $:\rightarrow$ ZIu $\times$ ZID
add_zi : ZICODE $\times$ Ideograph $\times$ ZIu $\times$ ZID $\rightarrow$ ZIu $\times$ ZID
is_zi : ZICODE $\times$ ZIu $\times$ ZID $\rightarrow$ Boolean
retrieve_zi :ZICODE $\times$ ZIu $\times$ ZID $\rightarrow$ Set of ideographs

Axioms
Declare x, x_1 : ZICODE,
y, y_1 : Ideograph,
U : ZIu,
D : ZID.

is_zi (x, New_ZiD) = False

is_zi (x, add_zi (x, y, U, D)) = True

add_zi (x, y, add_zi (x_1,y_1, U, D)) = add_zi (x_1,y_1, add_zi (x, y, U, D))

```
retrieve_zi (x, U, D) =
        If (U, D) == add_zi (x, y, $U_1$,$D_1$)  (for some ZIu $U_1$ and some ZID $D_1$)
        then {y} $\cup$ retrieve_zi (x, $U_1$,$D_1$)
        else $\varnothing$
```

Table 3. Zi Dictionary

3.2. Ci Dictionary

The operations for the ci dictionary are similar to those for the zi dictionary. These operations depend on the operation for the zi dictionary. For the function *add_ci*, all of the zis that occurs in the new ci to be added must have been added to the zi dictionary. Otherwise, an error will occur. Two ci checking functions are provided for convenience.

One is for checking if an input code corresponds to an existing ci in the CID (*is_ci_code*). The other checks if an ordered list of zis forms an existing ci (*is_ci*). Two functions are needed because one input code may correspond to more than one zi. Table 4 summarizes the specification for the ci dictionary.

Domain definition New_CID :→ CIu × CID
add_ci : CICODEs × CI × CIu × CID → CIu × CID
is_ci : CI × CIu × CID → Boolean
is_ci_code :CICODEs × CIu × CID → Boolean
retrieve_ci :CICODEs × CIu × CID → A set of CIs

Axioms
Declare X, X_1 : CICODE,
Y, Y_1 : CI,
U : CIu,
D : CID.

is_ci (Y, New_CID) = False

is_ci_code (X, New_CID) = False

add_ci (X, Y, U, D) = ERROR
if Not ($\forall$ y $\in$ Y, is_zi (y, ziu, zid))
where ziu is the current ZIu, and zid is the current ZID.

add_ci (X, Y, add_ci (X_1,Y_1, U, D)) = add_ci (X_1,Y_1, add_ci (X, Y, U, D))
if add_ci (X_1,Y_1, U, D) $\neq$ ERROR
and add_ci (X, Y, U, D) $\neq$ ERROR.

is_ci (Y, add_ci (X, Y, U, D)) = True
if add_ci (X, Y, U, D) $\neq$ ERROR.

is_ci_code (X, add_ci (X, Y, U, D)) = True
if add_ci (X, Y, U, D) $\neq$ ERROR.

```
retrieve_ci (X, U, D) =
        If D == add_ci (X, Y, $U_1$,$D_1$)  (for some CIu $U_1$ and some CID $D_1$)
        then {Y} $\cup$ retrieve_ci (X, $U_1$,$D_1$)
        else $\varnothing$
```

Table 4. Ci Dictionary

4. Implementation of the Chinese Dictionary

In this section, an implementation of the dictionary of permanent words is discussed, and the storage analysis is given in the next section. The whole dictionary consists of the following components:

- Zi Universe
- Zi Dictionary
- Ci Universe
- Ci Dictionary

The implementation of the dictionary presented here can be expanded dynamically and the storage utilization is also efficient. Each part of the dictionary consists of two components: the main data area and an overflow area. During the initialization phase, e.g., an application software requests usage of the dictionary, the dictionary is initialized with everything thing in the main data area. Overflow area is for dynamic expansion.

The implementation of the dictionary given here is based on pinyin input method. Table 5 gives some notations used throughout the paper.

Notation	Representation	Size	Approximate total number
a b c	phonetic symbols	1 byte	26
[ang] [ci]	pinyins	2 bytes	500
阿 癌	ideographs	2 bytes	6,000
[公 証]	Chinese words	2 bytes	20,000

Table 5. Some Notations

The zis and cis are indexed by sequences of pinyin symbols. Each pinyin corresponds to more than one zis. Details are given below.

4.1. Zi Universe (ZIu)

The ZIu initially consists of about 6000 commonly used ideographs. Though each ideograph is a bitmap, it is treated as a single unit. Let us consider the main ZIu area first. This area can be viewed as a set of ordered pairs of indices and bitmaps (N, bitmap), such as:

{ (1, 阿) , (2, 哀) , (3, 爱) , (4, 挨) , (5, 癌) , ... }.[2]

Each ideograph is uniquely identified by its index. Since the zis are stored sequentially in the main area, it is not necessary to store the index. Furthermore, all zis with the same pinyin code are stored adjacent to each other to facilitate searching. The overflow area is used for expansion, i.e., adding zis. In general, zis may be added in any order. Therefore, a pointer is stored with each zi to link with other zis of the same code to form a linear list in the overflow area. Fig. 1 shows part of the ZIu of a pinyin dictionary. Fig. 2 shows part of the overflow area.

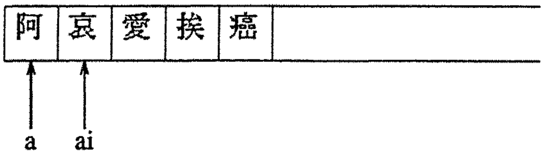

Fig. 1 Part of the ZIu

[2]This mapping depends on the pinyin input method. GB2312-80 is the standard coding scheme used in China. A one-one mapping can be established between the index used here and GB2312-80.

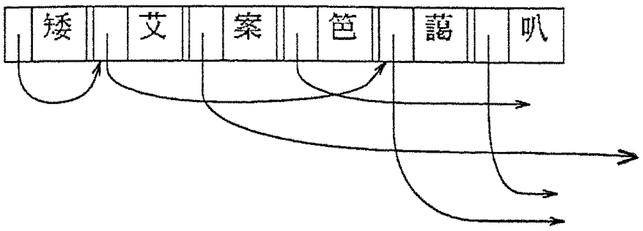

Fig. 2 Part of the ZIu Overflow Area

The ZIu is indexed by the zi dictionary, ZID. Given an input code, we can find the index to the ZIu from the ZID. In general, the input codes of zis are of varying size, e.g., phonetic spellings for pinyin input method. An appropriate data structure for ZID is the trie. This is called the *zi trie*. A trie is a tree in which the branch at any level is determined not by the entire key value but only by a portion of it [5]. In the zi trie, the branching at the *i*th level is determined by the *i*th character of the code. In our pinyin dictionary, the phonetic spellings of zis at the beginning of the ZIu are:

{a, ai, an, ang, ao, ba, bai, ban, bang, ... }

Fig. 3 is a picture description for this part of the zi trie.

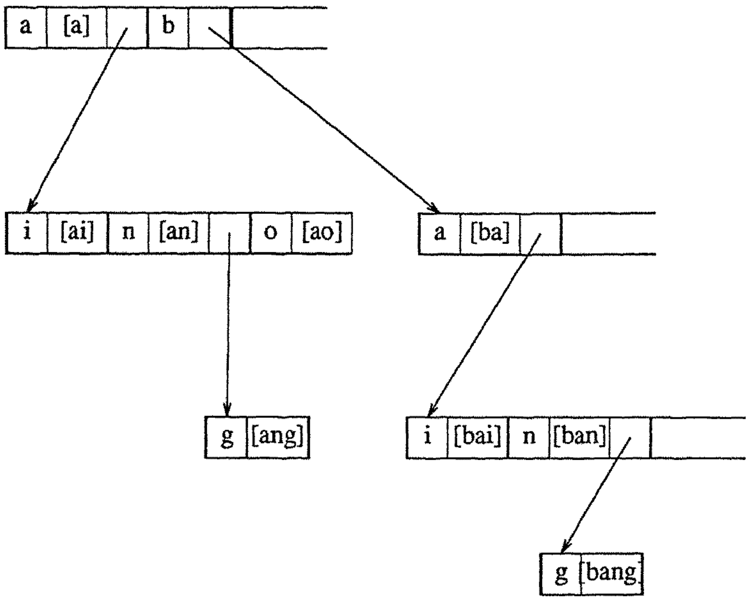

Fig. 3 The Zi Trie

In order to utilize storage efficiently, we used a linear representation for the zi trie instead of a pointer representation, i.e., the trie is stored using an array.

Each node of the zi trie is a variable size list of structures. Each structure is one of the three possible types:

- Grey: an ordered triple of a phonetic symbol, a pointer to the next node, and a representation of the pinyin.
- Black: an ordered pair of a phonetic symbol and a pointer to the next node.
- White: an ordered pair of a phonetic symbol and a representation for the pinyin spelling.

Fig. 4 is a picture description of different kinds of structures.

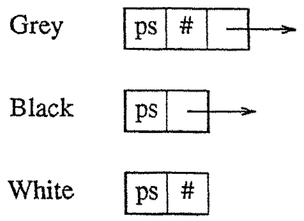

Fig. 4 Node Structure of the Zi Trie

The phonetic symbol in each structure is part of the whole key, and it is used to construct the original key. Key values in structures along a path from root to a grey node or a white node represent a valid pinyin code. We called this valid pinyin a *coded-pinyin* (the # in Fig. 4). In implementation, it is the index to the zi universe. The nodes of the trie are varied in size since the number of different phonetic symbols appears as the *i*th character of a spelling is different. A node can have at most 26 children (set of all possible phonetic symbols). The structure of a node are stored together sequentially for sequential search through a node to locate a correct partial key. Adding an extra field to the beginning of a node and some modifications to the node structure allow binary search. More modifications are required for a dynamic expandable dictionary. All these modifications would be shown in the implementation of the ci dictionary.

A more efficient implementation is available for pinyin zi dictionary. This implementation may not be applicable to other input methods because this method is based on some characteristics of the pinyin structure. With some study of pinyin method of Chinese characters, we found that the pinyin of a zi can be divided into 2 parts – an *initial* (聲母) and a *final* (韻 母) [7]. Intials and finals are the actual phonetic symbols in pinyin while "phonetic symbols" has been used to refer to the possible characters used for pinyin in previous sections. There are also different kinds of finals: *single finals*, *nasal finals*, and *compound finals*. All initials and finals are non-empty strings of characters. Furthermore, there are only 21 initials and 36 finals. By making use of this properties, we can implement the zi dictionary with a table. The rows are indexed with initials and the columns are indexed with finals. An extra row indexed by null is needed since there may be phonetic spellings with finals only. Fig. 5 displays part of this construction.

	finals (韻 母)					
initials (聲 母)	a	o	e	-i	er	ai
b	ba	bo			bai	
p	pa	po			pai	
m	ma	mo	me		mai	
f	fa	fo				
d	da		de		dai	
t	ta		te		tai	
n	na		ne		nai	
l	la		le		lai	

Fig. 5 Initials Finals Combination Table.

Each table entry (representing a certain pinyin) consists of 3 parts:

1. An index to ZIu (main area) for the first zi having the phonetic spelling (or pinyin).
2. An integer which is the number of zis of the pinyin that can be found in the main area of ZIu.
3. Another index pointing to first zi of that pinyin in the overflow area of ZIu.

As shown later, this implementation occupies less storage than the zi trie.

4.2. Ci Universe (CIu)

There is no need to store the bitmaps in the ci universe (CIu). Indices to the ZIu are used to refer to the zis making up a ci. The cis can be divided into different groups according to their length. There are tables of 2-ideograph cis, 3-ideograph cis, ..., in the CIu. The number of these tables depends on the length of the longest word. Usually, the maximum length is five. Words usually contain no more than five ideographs. The 2-ideograph table is a list of ordered pairs of indices to the ZIu. Each pair represents a legal word of two ideographs. Words with the same phonetic spelling are grouped adjacent to each other. Only one index is necessary to address a group of ideographs with the same phonetic spelling. The arrangement can be pictured in Fig. 6 below.

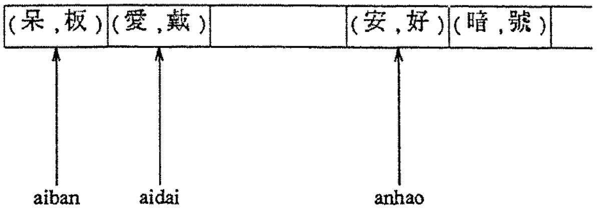

Fig. 6 The 2-ideograph Table

The 3-ideograph table is a list of ordered triples of indices to ZIu. Each of these triples represents a word of three ideographs. Similar to the 2-ideograph table, the words with the same phonetic spelling are stored adjacent to each other. Fig. 7 is an example.

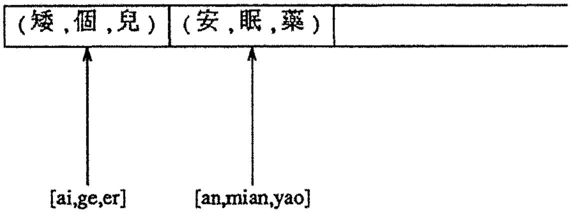

Fig. 7 The 3-ideograph Table

In general, *n*-ideograph table is a list of ordered *n*-tuples of indices to the ZIu. Each of these *n*-tuples represents a word of *n* ideographs. The words with the same phonetic spelling are stored adjacent to each other. To allow it to be dynamically expandable, a header node is added to each group of cis with the same pinyin. The header node consists of two pieces of information:

number of cis found in that group
a pointer to first ci with the same pinyin in the overflow area

The overflow area consists of cis being added dynamically. It is also separated into several areas according to the length of cis, that is, 2-ideograph cis are added to one table, 3-ideograph cis are added to another, and so on. Similar to the ZIu, an extra pointer is added with ci in the overflow area to keep the cis with same pinyin in a linear list.

4.3. Ci Dictionary

The zi dictionary is indexed by pinyin of the zis which are sequences of characters. Using pinyin input, the ci dictionary is indexed by sequences of pinyins of words. Each ci is a sequence of zis and each zi is a sequence of pinyins. So the ci dictionary is implemented as a trie with the nodes at the *i*th level being a representation of the *i*th zi of a ci. This representation is the coded-pinyin of the zi. The actual zis that correspond to a ci are stored by the zis index to the ZIu. Fig. 8 is an abstract picture for part of the ci trie of our pinyin dictionary. A coding scheme uses less storage without losing any efficiency. Since all zis with the same pinyin are stored together, a zi can be retrieved by its pinyin together with its displacement from the first zi in the list. This trie is called the *ci trie*.

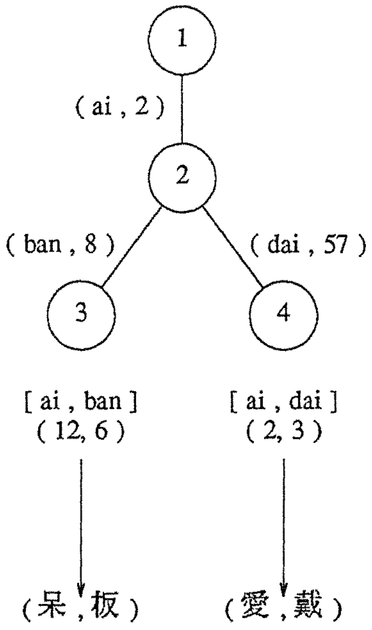

Fig. 8 Detail of a Ci Trie

In our pinyin dictionary, each internal node of the ci trie is also a variable list node structures. Similar to the ZID, each node is a union of 3 possible types of structures:

- Grey: an ordered triple of a coded-pinyin, a pointer to the next node, and a pointer to the Ci table.
- Black: an ordered pair of a coded-pinyin, and a pointer to the next node.
- White: an ordered pair of a coded-pinyin, and a pointer to the dictionary table.

The coded-pinyin in the structure is part of a word. For the grey nodes and the white nodes, the pointer to the dictionary table represents the word(s) constructed from the sequence of coded-pinyin along the path from the root of the trie to the current node. Depending on the level of the node, the pointer points to the appropriate ideograph table, e.g. the node at level 3 points to the 3-ideograph table. Fig. 9 is a detail description of the ci trie with the words "aiban" (呆板) and "aidai" (愛戴).

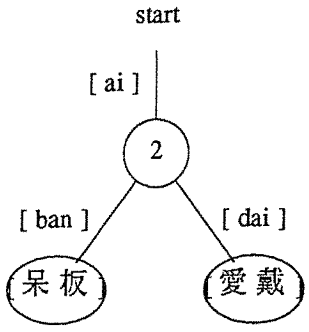

Fig. 9 The Ci Trie

The implementation is similar to the zi trie with the following exceptions:

1. The phonetic symbol in the zi trie is replaced by the coded-pinyin in the ci trie.
2. The coded-pinyin in the zi trie is replaced by the pointer to the dictionary table in the ci trie.

Another major difference between zi trie and the ci trie is the size of a node in the ci trie is much bigger than the one in the zi trie. In the zi trie, the maximum number of structure in one node is 26, while in the ci trie, the maximum number of structure is about 400 (from a counting of a phonetic spelling dictionary, the number of different phonetic spellings is 416). The node structure is modified to allow binary search on a node. A header containing a number which is the size of the node is added to each node. The structure is also modified to be uniform in size. The grey node is changed to a 4-byte structure, with the first two bytes storing the coded-pinyin, and the second two bytes storing a pointer to an ordered pair. The ordered pair contains a pointer to the dictionary table and a pointer to the next node. The ordered pair is stored outside the node. Fig. 10 is a pictorial description of the modified structure.

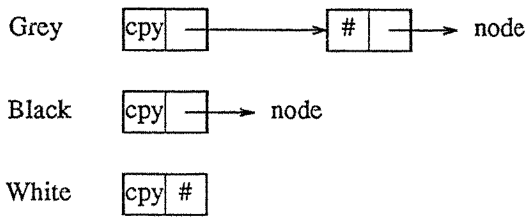

Fig. 10 Node Structure of the Ci Trie

Furthermore, in order to allow it to be expandable, we reserve overflow area for each level of the ci trie. One extra field is needed at the header of a node for the index to the overflow area.

5. Storage Analysis

We now give an estimate of storage utilization of the whole pinyin dictionary. In the analysis, we assume that 1 byte consists of eight bits and 1 K is 1024. For each part, we reserve 256 positions for overflow.

5.1. Zi Universe

In ZIu, each zi is a bitmap. Research has been done on how to compress the storage utilization for these ideographs. Some work has been done on implementing this database in hardware for fast retrieval and low storage requirements. This problem will not be discussed any further. Our implementation stores each ideograph by a 24X24 bitmap. Each bitmap can be represented by 72 bytes. The total storage required is 432 Kbytes for 6000 zis (72 × 6000). Two bytes are used to index a zi in the Chinese National Standard Code for Information Interchange (GB2312-80) [1]. In the overflow area, each zi is still 72 bytes long, but one extra byte pointer is required for each zi. So, the zi universe requires 19 Kbytes overflow area ((72+1)×256).

5.2. Zi Dictionary

Consider the array implementation. The array contains totally 792 (22×36) entries. Each entry consists of 4 bytes:

- 2 bytes for index to main area of zi universe
- 1 byte for the number of zis (with the same pinyin) in the main area
- 1 byte for index to the first zi of the same pinyin in the overflow area

Hence, the zi dictionary requires 792×4 bytes, which is less than 4 K bytes. Table 4 summarizes the storage requirement of the zi dictionary.

Parts	Calculation	Total (bytes)
Zi Universe	$72 \times 6000 = 432$ Kbytes	432K
Zi Overflow	$(72 + 1) \times 256 = 18688$ bytes	19K
Zi Dictionary	$22 \times 36 = 792$ entries	
	$4 \times 792 = 3168$ Bytes	4K
	Total (a)	455K

Table 4. Storage Utilization of Zi Dictionary

5.3. Ci Table

The storage utilization of the ci tables depends on the dictionary implemented. Our analysis confines to an implementation of the dictionary 漢語併音詞匯 [3]. The number of commonly used cis of each length is found and used in the analysis. The maximum length of ci allowed is six. When we are retrieving a ci, we should have gone through the ci trie already. Since we have a field of coded-pinyin of each zi in the ci trie, we can find the group of zis with that pinyin in the zi universe. Actually, we can use the index to the first zi of the pinyin as a representation of coded-pinyin. Therefore, it is not necessary to keep a 2-byte index for each zi of the ci in the ci table. We only require, for each zi of a ci, an offset from the first zi of that pinyin. One byte is enough for each zi. So an *n*-ideograph ci requires *n* bytes storage in the table. About 34 Kbytes is required to store all cis. Each group of ci with the same pinyin is proceeded by a 2-byte header. One byte for the number of cis in that group, and one byte for an index to overflow area. Since there are totally 15,435 ci in the ci table, so 30 K bytes (15,435×2) would be a loose upper bound for the storage requirement of the headers. Besides, different overflow areas are reserved for cis of length two to six. As in the ZIu, one more pointer is stored together with each ci in overflow area. Therefore, an *n*-ideograph ci requires *n*+1 bytes storage. 7 Kbytes are required for the overflow area of ci table. Table 5 summarizes the storage utilization of the ci table.

Types	Entries	Size
2-ideograph cis (2 bytes)	11993	23986
3-ideograph cis (3 bytes)	3376	10128
4-ideograph cis (4 bytes)	62	248
5-ideograph cis (5 bytes)	2	10
6-ideograph cis (6 bytes)	2	12
Subtotal	15435	34384
Heading node for each group of ci with same pinyin $2 \times 15435 = 30870$		30K
Overflow area of Ci table (*n*+1 bytes of each *n*-ideograph ci)	$3 \times 256 = 768$ Bytes $4 \times 256 = 1024$ Bytes $5 \times 256 = 1280$ Bytes $6 \times 256 = 1536$ Bytes $7 \times 256 = 1792$ Bytes	
	Subtotal 6400 Bytes	7K
	Total (b)	71K

Table 5. Storage Utilization of Ci Table.

5.4. Ci Trie

Ci trie is the second largest area in the pinyin dictionary. The basic unit of memory cell of the ci trie is two bytes long. The pointer to the dictionary table is two bytes, and the size of the coded-pinyin is also two bytes. Actually, 10 bits is enough for the coded pinyin. Two bits from this field can be used for distingushing different types of node structures. If the whole trie can be implemented within 128K bytes (verify later), two bytes are enough to store a pointer. The header of each node requires 2 bytes: one for the number of different structures in that node, another is an index to the overflow area. In the first level, all structures must be black nodes. 416 node structures are needed at this level. Hence, it requires 416×4 bytes and a header of two bytes. For the second level, there may be occurrence of all 3 kinds of nodes. It is not difficult to see that there are (11,993 + 3376 + 62 + 2 + 2) 4-byte structures in level 2. For example, if the first two zis of a 3-ideograph ci do not form a ci, there would be a black node in level 2 for that ci (4 bytes), otherwise, it would be a grey node (8 bytes). We also count twice in the expression if it is the later case. Other levels are counted similarly. Hence the ci trie adds up to 85 Kbytes storage. In the overflow area, 5-byte structure are used (1 more byte for pointer) and 256 positions are reserved for each level. It requires totally 14K bytes.

Summing up all these entries, 625K bytes are required for the whole dictionary. A summary of the storage analysis is given in Table 6.

Since the size of the overflow area is limited, reorganisation of the database may be required if the database is expanded frequently. For convenience, one may allow expandable overflow area, but it should be noted that if the overflow area is allowed to be expanded to have more than 256 positions, those pointers into or inside the overflow area should also be enlarged.

Level	Calculation	Total (bytes)
1st	$416 \times 4 + 2$	1666
2nd	$(11993 + 3376 + 62 + 2 + 2) \times 4 + 2 \times 416$	62572
3rd	$(3376 + 62 + 2 + 2) \times (4 + 2)$	20652
4th	$(62 + 2 + 2) \times (4 + 2)$	396
5th	$(2 + 2) \times (4 + 2)$	24
6th	$2 \times (4 + 2)$	12
	Total	85322
Overflow area	$256 \times (4 + 1) \times 6 = 14280$ Bytes	14K
	Subtotal (a)	455K
	Subtotal (b)	71K
	Total	625K

Table 6. Storage Utilization of the Ci Trie

6. The Transient Words

Besides the permanent words discussed in last section, there exists an infinite number of transient words or temporary words. These transient words are single word syntactically. Their usage in Chinese text is very common. Unfortunately, it is impossible to list all of them. For example, an infinite number of words can be constructed from a numeral, followed by *ge* (個), and followed by *ren* (人), such as:

yigeren	(一 個 人)	(one person)
lianggeren	(兩 個 人)	(two persons)
sangeren	(三 個 人)	(three persons)
shigeren	(十 個 人)	(ten persons)

In this section, methods to generate these transient words are discussed. These methods do not cover all aspect of transient word generation. They just provide some ideas of how to handle these infinite sets of words. Implementation details are not presented.

6.1. Numerals

The first group of transient words to be discussed is the *numerals.* Similar to Roman numerals, there exists an infinite number of Chinese numerals. These Chinese numerals can be generated by a regular expression. Unlike Roman numerals, the regular expression can generate some numerals that are illegal. This will not cause any problem since the dictionary is used in detecting the existence of a legal word instead of checking the correctness of a word. Following is the regular expression.

D (digit) = 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 九 | 零

N (number) = D^+

C = 十 | 百 | 千 | 萬 | 億

N = $(DC)^+$ | $(DC)^+$ | N 零 N

F (fraction) = C_1 分 之 C_2 |

  百 分 之 *N* |

  N 又 *F* |

  N 點 *N* |

  N 倍

Since a regular expression can be represented by a deterministic finite automaton, it can be implemented by extending the ci trie to a deterministic finite automaton.

6.2. Numeral-Measure-Noun Combination

In English, the phrase ''one pair of chopsticks'' consists of the following components: ''one'' is a numeral, ''pair'' is a measure, and ''chopsticks'' is a noun. In Chinese, the measure *ge* (個) is used with the noun *ren* (人), *wenti* (問 題), *xuexiao* (學 校), and *jihui* (機 会). There does not exist any general rule that governs the Measure-Noun combination pair. The only possible method is to list all of them. A fairly complete list of these combinations is in Chao's book [2]. Usually the Measure-Noun pair is proceeded by a Numeral.

A specifier (SP) may exist in front of a Nu-M-N construct. Part of the set of specifiers are listed as follow:

{ zhe (這), na (那), ge (各), di (第), tou (頭) }

Other types of transient words that can be generated with the combination of DFA and exhaustive listing are the *place words* and *time expression*.

6.3. Affixes

Similar to English, there exist affixes in Chinese language. Affixes are bounded morphemes that are added to other morphemes to form larger units. Other affixes are grammatical morpheme indicating number and aspect. Chinese has few affixes. The three kinds of affixes – prefixes, suffixes, and infixes – are discussed in the following sections. The list of affixes is extracted from Li and Thompson's book [6].

6.3.1. Prefixes

Following is the list of prefixes and their constructs:

PREFIXES	CONSTRUCT	EXAMPLE
lao (老)	lao-Surname	lao-Zhang (老 張)
xiao (小)	xiao-Surname	xiao-Zhang (小 張)
di (第)	di-Numeral	di-liu (第 六)
chu (初)	chu-Numeral	chu-er (初 二)
ke (可)	ke-Verb	ke-ai (可 愛)
hao (好)	hao-Verb	hao-kan (好 看)
nan (難)	nan-Verb	nan-kan (難 看)

6.3.2. Infixes

"–de–" (得) and "–bu–" (不) are the only infixes in Chinese. They are called potential infixes of verb compounds. For example,

shuo-de-qingchu (說 得 清 楚)
shuo-bu-qingchu (說 不 清 楚)

6.3.3. Suffixes

Following is a list of common suffixes.

SUFFIXES	CONSTRUCT	EXAMPLE
-men (門)	Human Noun-men	xuesheng-men (學 生 門) (students)
	Human Pronoun-men	wo-men (我 門) (we)
-xue (學)	Subject Name-xue (equivalent to *-ology*)	xinli-xue (心 理 學) (psychologist)
-jia (家)	Subject Name-jia (equivalent to *-ist*)	wulixue-jia (物 理 學 家) (physicist)
-zi (子)	Noun-zi	ti-zi (梯 子) (ladder)
-tou (頭)	Noun-tou	gu-tou (骨 頭) (bone)

7. Summary

In this report, an implementation of the permanent word dictionary and the construction of the transient word dictionary are discussed. The permanent word dictionary provide a fundamental framework for the Chinese dictionary. The transient word dictionary is augmented to the permanent word dictionary. It is just a technique to compress the size of the dictionary and store an infinite number of words. The tradeoff is that some illegal words are stored. Since the goal is to store all legal words instead of checking whether a word is legal or not, these minor defects will not cause any problem.

8. References

1. National Standards Bureau, Code of Chinese graphic character for information interchange (GB2312-80). 1980.
2. Y. R. Chao, *A grammar of spoken chinese*, University of California Press (1970).
3. Hanyu pinyin cihui, , Wenzi Gaige Chubanshe, Peking (1958).
4. J. V. Guttag and J. J. Horning, The algebraic specification of abstract data types, *Acta Informatica* **10** (1978), 27-52.
5. E. Horowitz and S. Sahni, *Fundamentals of data structures in Pascal*, Computer Science Press (1984).
6. C. N. Li and S. A. Thompson, *Mandarin Chinese: a functional reference grammar*, University of California Press, Berkeley (1981).
7. J. Sheng, A pinyin keyboard for inputting chinese characters, *Computer* (January 1985).