2011 2nd International Conference on Networking and Information Technology
IPCSIT vol.17 (2011) © (2011) IACSIT Press, Singapore

Compression of Double Array Structures for Fixed Length Keywords

Masao Fuketa, Hiroya Kitagawa , Takuki Ogawa, Kazuhiro Morita and Jun-ichi Aoe

Department of Information Science and Intelligent Systems, Faculty of Engineering,
University of Tokushima, 2-1Minami josanjima,
Tokushima, 770-8506, Japan
{fuketa, kitagawa , kam, aoe}@is.tokushima-u.ac.jp, ogawa-takuki@iss.tokushima-u.ac.jp

Abstract— Trie is one of the data structures for keyword matching. The trie is used in natural language processing, IP address routing, and so on. It is represented by the matrix form, the link form, the double array, and LOUDS. The double array combines retrieval speed of the matrix form with compactness of the list form. LOUDS is a succinct data structure using bit-string. Retrieval speed of LOUDS is not faster than that of double array, but the dictionary size is smaller. This paper proposes a compression data structure of the double array by dividing trie into each depth and removing the BASE array from that double array. From experimental results, the retrieval speed was almost the same as double array, and the size of the presented method was the most compact in other methods including LOUDS for a large set of keywords with fixed length.

Keywords- trie; double array; fixed length; compression

1. Introduction

Keyword matching is very important in many applications. Trie is one of the data structures for the keyword matching. Trie is an ordered tree data structure merged the prefixes of keys. The trie is used in information retrieval systems[1], natural language processing[2], IP address routing tables[3], packet filtering[4], and so on.

Traditionally, the trie was represented either by a two-dimensional array called matrix form or by a two-dimensional linked list called the link form. Retrieval speed of the matrix form is very fast, but it needs large spaces. The space of the link form is efficient, but the retrieval speed is slow. Aoe presented an efficient data structure for trie called double array[5]. This method uses two arrays called BASE and CHECK, and combines the retrieval speed of the matrix form with compactness of the list form. As schemes reduce the dictionary size of the double array, a method that CHECK array keeps character codes are proposed[6]. This method is useful to make the double array compact and the retrieval speed is almost as fast as the original double array. However, the dictionary size is still big.

Level-Order Unary Degree Sequence (LOUDS) for succinct data structure is proposed as a structure of trie[7]. The dictionary size of this method is much smaller than that of the double array, but the retrieval speed is not so fast.

This paper proposes the compression data structure of the double array called single array by deleting the BASE array. Furthermore, the proposed method divides the trie for each depth. In this method, as BASE array is deleted and CHECK array keeps character codes, the dictionary size is very small. As this data structure keeps features of the double array, the retrieval speed is very fast.

Section II describes the trie and the double array. In Section III, the data structure of single array is defined and a retrieval algorithm using those data structures is described. Theoretical and experimental

evaluations are given in Section IV. Finally, Section V concludes the presented algorithm and describes further works.

2. Double Array

Trie is an ordered tree data structure merged the prefixes of keys. Fig.1 shows the trie of a keyset K = {ab#,abc#,b#,bac#,bd#}. The special end marker # is added to the ends of all keys. The marker is used to avoid confusion between keys “ab” and “abc”. A retrieval on trie starts from root node(state number 1) and traces each character of key step by step.

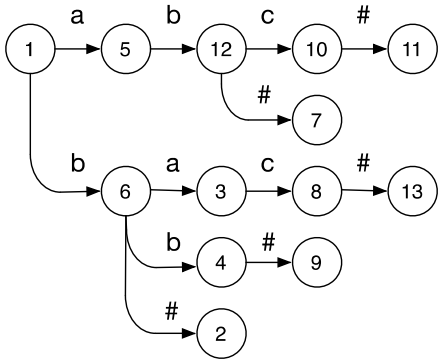

Figure 1. A trie for K

	#	a	b	c
1		5	6	
2				
3				8
4	9			
5			12	
6	2	3	4	
7				
8	13			
9				
10				11
11				
12	7			10
13				

Figure 2. A matrix form for K

	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	3		4	8	9	1		12		10		6	
CEHCK		6	6	6	1	1	12	3	4	12	10	5	8

	#	a	b	c
CODE	1	2	3	4

Figure 3. A double array for K

	1	2	3	4	5	6	7	8	9	10	11	12	13
BASE	3		4	8	9	1		12		10		6	
CEHCK		#	a	b	a	b	#	c	#	c	#	b	#

Figure 4. A compact double array for K

The trie is generally represented by an array and list forms. Fig. 2 shows a matrix form for keyset K. The retrieval speed of the matrix form is very fast, but it takes large spaces. The space of the link form is efficient, but the retrieval speed is not so fast.

A double array is proposed by Aoe[5] is one of data structures of the trie and combines fast access of an array with compactness of a list form. In double array, when a character *c* of key traces from parent state *s* to child state *t*, it is satisfied as follows:

$$t = \mathrm{BASE}[s] + \mathrm{CODE}[c] \quad (1)$$
$$\mathrm{CHECK}[t] = s$$

A character *c* of key traces from state *s* to state *t* by the first equation. What state t connects state s is checked by the second equation. CODE[*c*] represented a numerical code of character c. CODE[*c*] for all characters are unique not to trace to the same child state number by a different character from the parent state number. Fig. 3 shows a CODE array and a double array of keyset K. The state numbers of the double array are corresponded to indexes of BASE and CHECK, the state numbers are defined during registering keys. In Fig. 1, character 'a' traces from state number 6 to state number 3. The child state number t is set to BASE[6]+CODE[a]=3, and then it is checked that t is equal to CHECK[3].

Moreover, a compact double array proposed by Yata[6]. In this data structure, a character c of key is stored as a value of CHECK array. That is to say, equation 1 is changed to as follows:

$$t = \mathrm{BASE}[s] + \mathrm{CODE}[c] \quad (2)$$
$$\mathrm{CHECK}[t] = c$$

BASE values must have unique values, because more than one character may trace to the same state. Double array of Fig. 3 can be changed to compact double array because each BASE value is unique. Fig. 4 shows a compact double array of keyset K. CODE is the same as Fig. 3. As character 'a' traces from state number 6 to state number 3, CHECK[3] is 'a'.

In an original double array proposed by Aoe (called ODA), CHECK stores the parent state number. When the large number of keys is registered to a trie, state numbers in the trie are increased. Then, CHECK arrays are represented by 2 bytes or 4 bytes with growing the number of states for a large keyset. But in a compact double array proposed by Yata (called CDA), CHECK stores a character. CHECK arrays are always represented by 1 byte. The size of BASE array is the same in both methods.

We propose a new data structure that can reduce the size of double array by removing BASE array.

3. Single Array with Multi Code

3.1 Data structure

If BASE[s] is set to s, equation 1 is as follows;

$$t = s + \mathrm{CODE}[c] \quad (3)$$
$$\mathrm{CHECK}[t] = s$$

In this method, BASE array is able to be removed. This method is called Single Array. The dictionary size of this method is smaller than that of double array. When the dictionary of double array is built, CODE values are fixed first and BASE values are determined[5] by orders of the depth. But, in case of using equation 3, only CODE values must be determined, because BASE values are fixed as s (the state number). The child state number isn't able to be determined without determining the parent state number. For example, let us consider keyset is 'ab' and 'ba'. This time, the CODE value of character 'a' is not determined without determining the CODE value of character 'b'. The CODE value of character 'b' is not determined without determining the CODE value of character 'a'. Therefore, all CODE values must be determined at the same time. When the number of characters in trie is bigger or the number of keys is bigger, it is more difficult to build a dictionary using equation 3. To solve this problem, this paper proposes a method to divide the trie into each depth, and the CODE have values for each depth.

First of all, the following condition is applied for all state numbers.

$$x < y \ (x \in D_k, \ y \in D_{k+1}, \ 1 \leqq k \leqq n-1)$$

where D_k is a set of state numbers of trie for depth *k*, *n* is the maximum depth of trie. CODE has different values in each depth by dividing trie into each depth. Here, CODE array is changed to two-dimensional arrays (called multi CODE). When a character c of key traces from parent state *x* to child state *y*, the following equation is defined;

$$y=x+\mathrm{CODE}[k][c]\ (x\in \mathrm{D}_k,\ y\in \mathrm{D}_{k+1},\ 1\leqq k\leqq n+1)$$

When CODE values for all characters are unique in depth k, it is impossible to trace the same child state number for depth k+1 from the parent state number x for depth k by character c. When there are CODE for each depth, CODE values can be determined depending on orders of the depth.

In case of equation 2, BASE[s] are unique not to trace the same state number by more than one character for storing character c to CHECK. In case of equation 3, s is

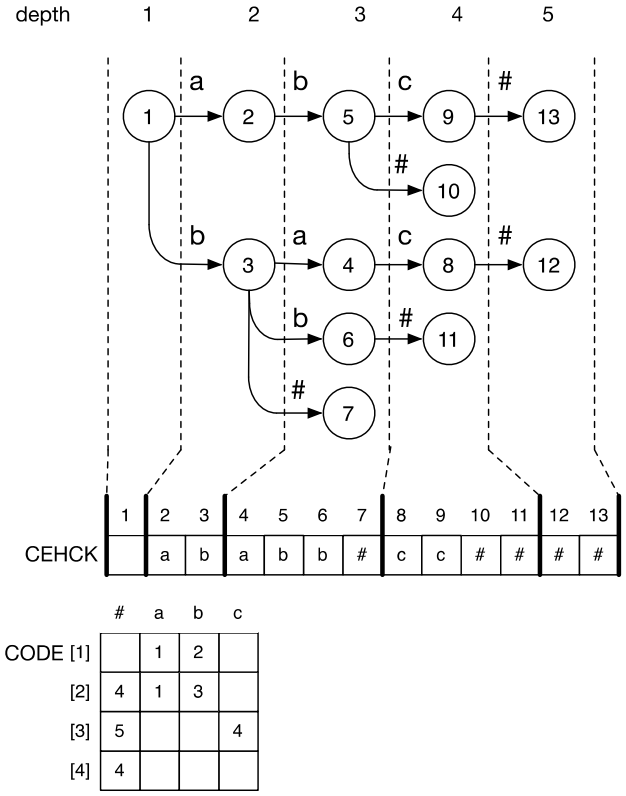

Figure 5. A single array with multi CODE for K

always unique because of s=BASE[s], it is impossible to trace the same child state number by t=s+CODE[k][c]. Therefore character c can be stored to CHECK array. Finally, the following equations are determined;

$$t = s + \mathrm{CODE}[k][c]\ (s\in \mathrm{D}_k,\ t\in \mathrm{D}_{k+1},\ 1\leqq \mathrm{k}\leqq n+1) \qquad (4)$$
$$\mathrm{CHECK}[t] = c$$

BASE array is removed from double array and CODE array become two-dimensional array. This structure is named as Single Array with Multi CODE(SAMC). Fig. 5 shows SAMC for keyset K.

3.2 The retrieval algorithm

An input of retrieval algorithm is KEY = $a_1,a_2,\ldots, a_n,a_{n+1}$='#' (length = n). An output is true if KEY is found, and is false if KEY is not found. The difference between the presented method and original double array is how to check the range of the child state number t. In double array, the retrieval algorithm checks if t is not bigger than the maximum state number. The presented method checks if t is contained in the state numbers of next depth in (R-4). The function max(D_k) returns a maximum state number in D_k. Other parts of this algorithm are the same as double array.

[Retrieval Algorithm]

(R- 1) s=1;
(R- 2) for(k=1;k<=n+1;k++){
(R- 3) t=s+CODE[k][a_k];
(R- 4) if(max(D_k)<t≦max(D_{k+1}) && CHECK[t] == a_k){
(R- 5) if(k == n+1)return true;
(R- 6) s=t;
(R- 7) }

```
(R- 8)   else{
(R- 9)      return false;
(R-10) }
(R-11)}
```

4. Evaluations

4.1 Theoretical Observations

Table 1 shows the time complexity of retrieval. Let $|\Sigma|$ be the number of characters. The retrieval algorithm of the presented method is very much similar to that of the original double array. The time complexities for retrieval for both methods are the same. Let k be the length of the input key, the time complexity of retrieval for both methods is $O(k)$. Matrix form and CDA is $O(k)$ by the same manner. In LOUDS, the time complexity is $O(k\,|\Sigma|)$, because all characters may be checked for each depth like list form. Therefore, LOUDS is slower than other methods.

Table 2 shows the sizes of dictionaries. Let m be the maximum length of keywords in the dictionary and $|D|$ be the number of the state numbers. The size of LOUDS is calculated based on the implementation of the tx library[1] .In the ODA, the BASE value and CHECK value are represented as 4 bytes. In the CDA, the BASE value and CHECK value are represented as 4 bytes and 1 byte, respectively. CODE values are represented as 1 byte for ODA and CDA. In the presented method, CHECK value is represented as 1byte. CODE value is represented as 4 bytes, because CODE values are bigger than the number of states for each depth of trie. In matrix form, state numbers are represented as 4 bytes. The size of matrix form is biggest. Because $4\,m\,|\Sigma|$ is very small comparing with

TABLE I. TIME COMPLEXITIES OF RETRIEVAL

Trie	Time Complexity
matrix	k
ODA	k
CDA	k
SAMC	k
LOUDS	$k\|\Sigma\|$

TABLE II. SIZE OF DICTIONARIES

Trie	Size of dictionaries
matrix	$4\|\Sigma\|\|D\|$
ODA	$8\|D\|+\|\Sigma\|$
CDA	$5\|D\|+\|\Sigma\|$
SAMC	$\|D\|+4m\|\Sigma\|$
LOUDS	$1.34\|D\|$ (=11/32$\|D\|+\|D\|$)

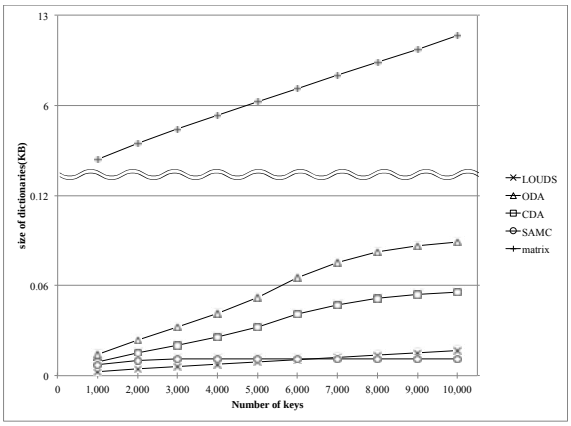

Figure 6. The dictionary sizes for 4-digit numbers

[1] http://www-tsujii.is.s.u-tokyo.ac.jp/~hillbig/tx.htm

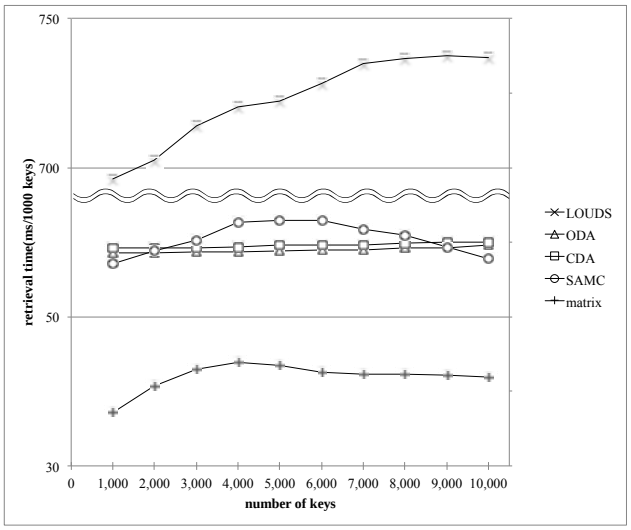

Figure 7. The retrieval Time for 4-digit numbers

|D| for most of keysets, the size of the presented method is the smallest, but these sizes are in case of max(D) = |D| . In double array and the presented method, there are many empty states, and the sizes of including empty states are evaluated by the following experimental results. However in presented method, as child state numbers are specified by parent state numbers, the size of CHECK array for depth k+1 is mostly bigger than that for depth k. In case of keyset with variable length (maximum length *n*) when *k*(<=*n*) is small, $|D_{k+1}|$ is bigger than $|D_k|$. But from some points, $|D_k|$ is bigger than $|D_{k+1}|$. Therefore, when keys are variable length, the size of dictionary is big for increasing unused states. When keys are fixed length, $|D_{k+1}|$ is always bigger than $|D_k|$. So experiments are done using keyset with fixed length.

4.2 Experimental Observations

Programs of ODA, CDA and presented method were written in C language. Tx library was used as the program

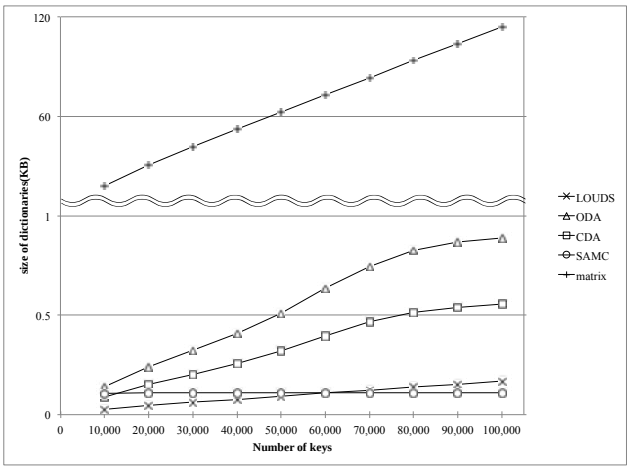

Figure 8. The dictionary sizes for 5-digit numbers

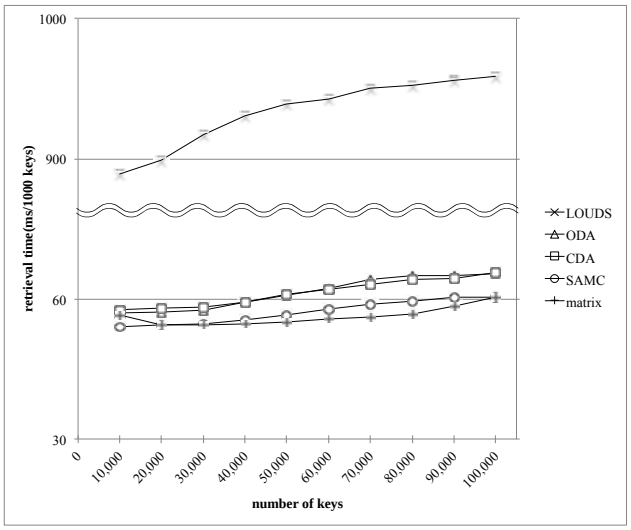

Figure 9. The retrieval Time for 5-digit numbers

of LOUDS. These programs were implemented on the following PC: Intel Xeon 2.4GHz(L2 cache:256K-Byte). It compared about the dictionary size and retrieval speed.

The experiments used 10,000 keys with fixed length from 0000 to 9999. Randomized 1,000 to 10,000 keys were used as keysets.

Fig. 6 shows the sizes of dictionaries. The size of matrix form is much bigger than other methods. When the number of keys is big, the presented method is the most compact. Furthermore the size of the presented method is mostly more compact than ODA and CDA.

Fig. 7 shows the retrieval time. All the registered keys are retrieved, and the average retrieval time per 1,000 keys is evaluated. The LOUDS is clearly slower than other methods. Because the retrieval algorithm of the presented method is as fast as that of the double array, the retrieval speed of the presented method is similar to other double array methods (ODA and CDA).

Next experiments used five-digit numbers with fixed length from 00000 to 99999. Fig. 8 and Fig. 9 show the dictionary sizes and the retrieval time, respectively.

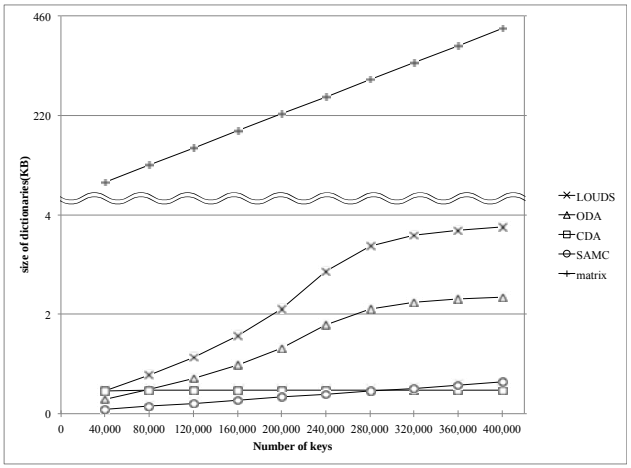

Figure 10. The dictionary sizes for 4-lowercase alphabets

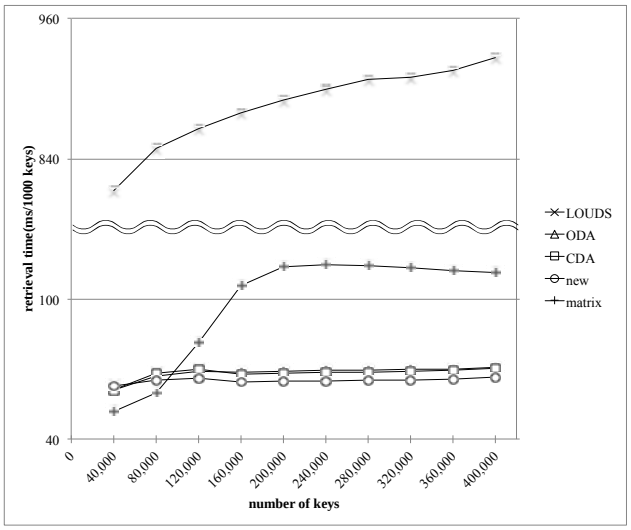

Figure 11. The retrieval Time for 4-lowercase alphabets

Moreover, experiments conducted using four-lowercase alphabets from "aaaa" to "zzzz". The number of keys in this keysets is 456,976(=26^4). Fig. 10 and Fig. 11 show the dictionary sizes and the retrieval time, respectively. From these results, even if the length of keys is bigger or the kinds of characters increase, results were the same as the most cases.

From the above observation, the presented method has the same retrieval speed as ODA and CDA. In addition, when a large set of keywords with fixed length is used, the size of the presented method is more compact than that of LOUDS as succinct data structure.

5. Conclusion

A new compression method of the double array has been presented by removing the BASE array. From theoretical and experimental observations, it is verified that the presented method is the most effective in double array methods. The size of the presented method is more compact than that of LOUDS as succinct data structure for a large set of keywords with fixed length especially.

A new method of the double array has been presented for keyset with fixed length. Further work will focus on compacting the dictionary size for keysets with variable length by keeping the fast retrieval speed.

6. References

[1] M. D. Brain and A. L. Tharp, "Using tries to eliminate pattern collisions in perfect hashing," IEEE Transactions on Knowledge and Data Engineering, vol. 6, no. 2, pp. 239–247, 1994.

[2] R. Baeza-Yates and G. Gonnet, "Fast text searching for regular expressions or automaton searching on tries," Journal of the ACM, 43(6), pp. 915–936, 1996.

[3] J. Fu, O. Hagsand, and G. Karlsson, "Improving and analyzing LC-Trie performance for IP-address lookup," Journal of Networks, 2, pp. 18–27, 2007.

[4] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," Proc. of the ACM SIGCOMM'98, pp.191-202, Vancouver, British Colombia, Canada.

[5] J. Aoe, "An efficient digital search algorithm by using a double-array structure," IEEE Transactions on Software Engineering, SE-15(9), pp. 1066–1077, 1989.

[6] S. Yata, M. Oono, K. Morita, M. Fuketa, T. Sumitomo, and J. Aoe, "A compact static double-array keeping character codes Source," Information Processing and Management, 43(1), pp. 237–247, 2007.

[7] G. Jacobson, "Space-efficient static trees and graphs," In Proc. 30th FOCS, pp. 549-554, 1989.