A Retrieval Method for Double Array Structures by Using Byte N-Gram

Masao Fuketa, Kazuhiro Morita, and Jun-Ichi Aoe

***Abstract*—Retrieving keywords requires speed and compactness. A trie is one of the data structure to retrieve keywords, and the double array is one of the implementation methods for the trie. The retrieval algorithm for the double array is fast, and its data structure has compactness. An edge of the trie is represented by a character in previous researches related to the double array, but there are no researches discussing if the edge is represented *n*-gram. Therefore, this paper proposes the data structure and the retrieval algorithm for the double array which represents an edge by *n*-byte. This paper also proposes a method to compress CODE array. From the experimental results comparing with the original double array by using single-byte and multi-byte character sets, the size and the retrieval speed of the proposed method became 62-64% and 1.18-1.3 times, respectively. When the CODE is compressed, the sizes of the proposed method became 41-59%.**

***Index Terms*—Compression, double array, n-gram, trie.**

I. Introduction

In ubiquitous environments such as smart phones and PDAs, the storage capacity is often limited. Retrieving keywords used in many applications requires speed and compactness. A trie is one of data structures to retrieve keywords. In the trie, common prefixes of stored keys are merged and each edge is labeled with a character consisting of keys. Because the trie can retrieve common prefix keywords and predictive keywords, it is used in information retrieval systems [1], natural language processing [2], IP address routing tables [3], and packet filtering [4]. Moreover, the trie is often used as an associative array [5] like a map class in C++ in order to improve the array by hash tables.

A double array is one of the retrieval methods by using the trie. This method uses two arrays called BASE and CHECK, and it has speed and compactness [6], [7]. An edge of the trie is represented by a character in previous researches related to the double array. As for the compression of the double array, there are methods dividing the trie [8], [9], a method removing BASE array [10], but there are no researches discussing if the edge is represented *n*-gram. Therefore, this paper proposes the double array method which represents an edge by *n*-byte. This paper also proposes a method to compress CODE array by using the double array, because CODE array becomes big with *n*'s increasing.

Section II describes the trie and the double array. Section III describes the proposed data structures and retrieval algorithms. Experimental evaluations are given in Section IV. Finally, Section V concludes the proposed algorithm and describes further works.

Manuscript received September 23, 2013; revised November 25, 2013. This work was supported by JSPS KAKENHI Grant Number 24500118.

The authors are with the Department of Information Science and Intelligent, University of Tokushima, Tokushima, Japan (e-mail: fuketa@is.tokushima-u.ac.jp, kam@is.tokushima-u.ac.jp, aoe@is.tokushima-u.ac.jp).

II. Double Array

A trie is a tree structure to store some keys. The pronunciation of "trie" comes from "re<u>trie</u>ve" to distinguish from "tree. In the trie, common prefixes of stored keys are merged and each edge is labeled with a character consisting of keys. Therefore, the trie has the property to retrieve all prefix words. Key retrieval always starts from the root of the trie and traverses nodes by one-by-one character in the key, and does not depend on the number of keys in the trie. From above these features, the trie is used in various fields such as natural language processing, network routing, and so on. Recently, the trie is often used as an associative array like a map class in C++ in order to improve the array by hash tables. The hash approach has some collisions of keys, and the worst-case retrieval time depends on the number of stored keys.

Fig. 1 shows a sample of the trie for key set K={"aaac#", "aab#", "ab#", "abb#", "abba#"} . '#' is added at the end of all keys. The end-mark is used to avoid confusion between keys "ab" and "abba". When "abba" is retrieved, "ab" as the prefix word of "abba" can be retrieved.

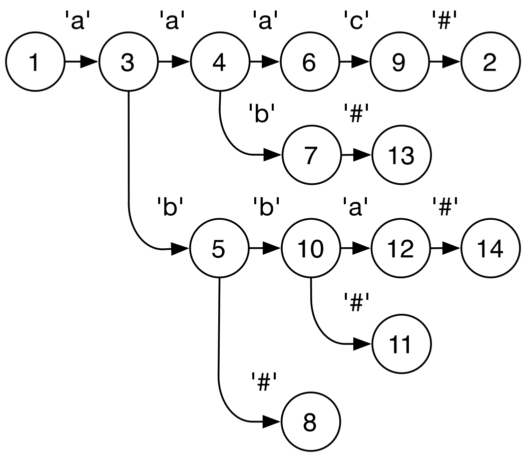


Fig. 1. An example of a trie.

A double array is one of data structures to implement tries. There are some other data structures to implement tries, which are a two-dimensional array, a linked list, and a LOUDS (Level-Order Unary Degree Sequence) [11]. In these three data structures, the two-dimensional array is the fastest and the LOUDS is the slowest in the retrieval speed. In the space, the LOUDS is the smallest and the two-dimensional array is the biggest. There are trade-offs in the time and the space. The double array can keep the high speed of the two-dimensional array and compresses it. That is to say, traversing one character for the double array is O(1).

The double array uses two one-dimensional arrays named BASE and CHECK. In the double array, when an arc labeled by character c traverses from parent node s to child node t, the following equations are satisfied;

t = BASE[s] + CODE[c], CHECK[t] = s, where CODE[c] represents a numerical code of character c. The edge traversal takes O(1) constantly. When the traversal reaches the leaf node s, BASE[s] is set to a negative number which represents a pointer toward records associated with the key. Fig. 2 shows the double array for Fig. 1. This data structure is called an original double array.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
BASE	1	-1	2	4	7	5	12	-3	1	10	-4	13	-2	-5
CHECK		9	1	3	3	4	4	5	6	5	10	10	7	12

	#	a	b	c
CODE	1	2	3	4

Fig. 2. Double array for Fig. 1.

III. Proposal Method

A. Outline

An edge of the trie is represented by a character in previous researches for the double array in order to retrieve prefixes of a given key. Since characters are represented by 1 byte in the ASCII character set, an edge is represented by 1 byte. When an edge is 1 byte, the maximum of the index of CODE array is 256(=2^8). The size is very small.

Fig. 3 shows the trie represented edges as 2 characters for key set K. The edge starting the end-mark is represented by 1 byte. Other end-marks joins a previous character, and the edge is represented by 2 bytes. For example, “b#” is traversed from node 4 to node 7.

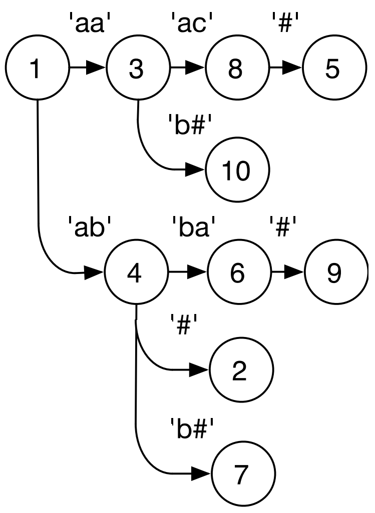


Fig. 3. An example of 2-bytes trie.

In Fig. 1, since the end-mark is traversed from node numbers 5 and 10, “ab” and “abb” can be retrieved during retrieving “abba”. In Fig. 3, the end-mark is traversed from node number 4. In this case, when “abba” is retrieved, only “ab” is retrieved and “abb” can’t be retrieved. In the same manner, if edges of the trie are represented by n-bytes(n>=2), all prefixes of a given key can’t be retrieved. But, when trie is used to retrieve whole keys such as associative array, it is not necessary to retrieve prefixes. Therefore, in this paper, we consider to retrieve whole keys and a method to represent edges as n-byte is proposed. This method is named n-gram double array. When edges are represented n-byte, the size of CODE array becomes big. In the double array, CODE array is used to return values corresponding to characters of edges. But, in the n-gram double array, when the number of characters of edges increases, the size of CODE array becomes big. Moreover, a data structure which compresses CODE array by the double array is proposed.

B. N-Gram Double Array

Fig. 4 shows the 2-gram double array for Fig. 3. CODE array is prepared for characters of all edges. The edges including the end-mark are represented by 1 byte or 2 bytes for 2-byte trie such as Fig. 4. In 3-byte trie, the edges including the end-mark are represented by 1 byte, 2 bytes, or 3 bytes. BASE and CHECK arrays are constructed in the same manner of the original double array.

	1	2	3	4	5	6	7	8	9	10
BASE	1	-3	4	1	-1	8	-4	4	-5	-2
CHECK		4	1	1	8	4	4	3	6	3

	#	aa	ab	ac	ba	b#
CODE	1	2	3	4	5	6

Fig. 4. 2-gram double array for key set K.

An input of retrieval algorithm is $a_1a_2..a_ka_{k+1}$, and a_i represents 1-byte character. k is the length of the input string, and a_{k+1} is ‘#’. $a_i..a_j$ in this algorithm is represented as characters from i-th character to j-th character for the input string a. CODE function returns the value corresponding to n-byte characters. An output is TRUE if the input string is found, and is FAILURE if the input string is not found.

```
[Retrieval algorithm]
(nr-1) $s$ =1;
(nr-2) for($i$ = 1 ; $i$ <= $k$+1-$n$ ; $i$ +=$n$){
(nr-3) $t$ = BASE[$s$]+CODE($a_i..a_{i+n-1}$);
(nr-4) if(CHECK[$t$] != $s$)return FAILURE;
(nr-5) $s$ = $t$;
(nr-6)}
(nr-7) $t$ = BASE[$s$]+CODE($a_i..a_{k+1}$);
(nr-8) if(CHECK[$t$] != $s$ or BASE[$t$]>=0)
return FAILURE;
(nr-9) return TRUE;
```

In the loop from (nr-2) to (nr-6), edges without the end-mark are traversed by each n-byte characters. CODE returns the value for n-byte characters in (nr-3). Edges with the end-mark are traversed, and the traversal is checked the traversal in (*nr*-7) and (*nr*-8). In (*nr*-7), CODE returns the value for the remaining characters of the input key. The length of the characters is from 1-byte to *n*-byte. When the traversal of the end-mark is succeeded, this algorithm returns TRUE in (nr-9).

For example, key "aab#" with *k*=3 is retrieved. First of all, *s* is set to 1 in (*nr*-1). Next, *t* is set to 3(=BASE[1]+CODE("aa")=1+2) in (*nr*-3) and CHECK[3]=1 is satisfied. *s* is set to 3 in (*nr*-5). Because next 2-byte of the input key is "b#" that contains the end-mark, the for loop is finished. In (*nr*-7), *t* is set to 10 (=BASE[3]+CODE("b#")), and CHECK[10]=3 is satisfied. Because the last traversal is succeeded, this algorithm returns TRUE in (*nr*-9).

In this method, although CODE array stored the value for *n*-byte, other parts such as BASE and CHECK arrays are the same as the original double array. Therefore, this structure can be updated dynamically.

C. *The Compression of the CODE Array*

CODE array is used to return values corresponding to characters of edges. But, in the *n*-gram double array, when the number of characters of edges increases, the size of CODE array becomes big. Therefore, CODE array is compressed by the trie and the double array. Fig. 5 shows the trie representation for the CODE array of Fig. 4.

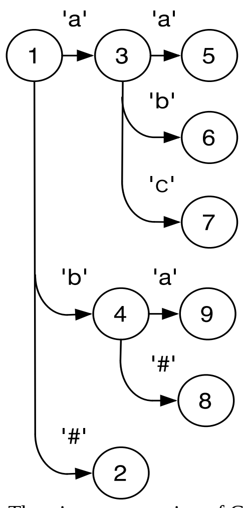
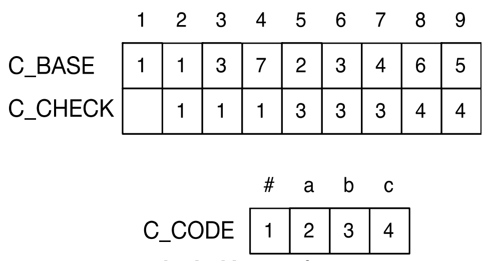


Fig. 5. The trie representation of CODE.

The maximum of the depth for this trie is n for *n*-gram double array. Therefore, the depth of the leaf node for this trie is n, or the edges to the leaf nodes are the end-mark. This trie is constructed by the double array. Fig. 6 shows the double array for CODE.



	1	2	3	4	5	6	7	8	9
C_BASE	1	1	3	7	2	3	4	6	5
C_CHECK		1	1	1	3	3	3	4	4

	#	a	b	c
C_CODE	1	2	3	4

Fig. 6. The double array for CODE.

C_BASE and C_CHECK are BASE array and CHECK array for the double array of CODE, respectively. In this CODE double array, the CODE array that differs from the CODE array of the *n*-gram double array is needed. The array is called C_CODE array. A value for each character is stored to the array.

A CODE retrieval algorithm by using the double array returns the value corresponding to *n*-byte characters $c_1..c_n$. Returned value for the leaf node *s* is stored to C_BASE[s]. For example, CODE value for "*aa*" is 2 in Fig. 4, and "*aa*" is reached to node 5 in Fig. 5. Therefore, 2 is stored to C_BASE[5]. This algorithm returns 2 for string "*aa*".

```
[Retrieval algorithm for CODE double array]
(cr-1) s =1;
(cr-2) for(i =1 ; i <=n ; i++){
(cr-3) t = C_BASE[s]+C_CODE[c_i];
(cr-4) if(C_CHECK[t] != s)return 0;
(cr-5) if(c_i is end-mark)break;
(cr-6) s = t;
(cr-7) }
(cr-8) return C_BASE[t];
```

In the same manner of the double array, retrieval starts from node 1 in (*cr*-1). Because the maximum of the depth for trie is *n*, the loop from (*cr*-1) to (*cr*-7) repeats *n* times. In (*cr*-3) and (*cr*-4), it is checked to traverse from node *s* to node *t* by character c_i. When the traversal is failed, this algorithm returns 0. When the end-mark is traversed, this loop is terminated in (*cr*-5). Finally, this algorithm returns the CODE value in (*cr*-8).

IV. Evaluations

Programs of the proposed method were written in C language. These programs were implemented on the following PC: Intel Xeon 2.4GHz (L2 cache:256K-Byte). The programs compared in terms of the space usage and retrieval speed. 147,478 words of WordNet3.0 and 87,995 words of Japanese WordNet as the key sets were used in experiments. The character set of Japanese WordNet was UTF8.

The double array is represented the original double array. The BASE value and CHECK value are represented as 4 bytes. The value for the CODE array is represented as 4 bytes. Table I and Table II show the space usage and retrieval speed about the *n*-gram double array with the CODE array for 2 key sets.

TABLE I: THE RESULT OF WORDNET FOR N-GRAM DOUBLE ARRAY.

	1-gram	2-gram	3-gram
The number of CODE values	256	65,536	16,777,216
The size of CODE array (byte)	1,024	262,144	67,108,864
The number of nodes for the double array	880,273	513,948	391,254
The size of BASE array (byte)	3,521,092	2,055,792	1,565,016
The size of CHECK array (byte)	3,521,092	2,055,792	1,565,016
Total size (byte)	7,043,208	4,373,278	70,238,896
The rate of the size	1	0.62	9.9
The retrieval time (ms)	27.64	23.40	24.02

TABLE II: THE RESULT OF JAPANESE WORDNET FOR N-GRAM DOUBLE ARRAY.

	1-gram	2-gram	3-gram
The number of CODE values	256	65,536	16,777,216
The size of CODE array (byte)	1,024	262,144	67,108,864
The number of nodes for the double array	532,330	309,460	263,905
The size of BASE array (byte)	2,129,320	1,237,840	1,055,620
The size of CHECK array (byte)	2,129,320	1,237,840	1,055,620
Total size (byte)	4,259,664	2,737,824	69,220,104
The rate of the size	1	0.64	16.25
The retrieval time (ms)	19.45	14.34	12.21

In these Tables, the 1-gram double array is the original double array. The retrieval time is the time that all the registered keys are retrieved. The original double array is a baseline method. The number of CODE values is the maximum of the number represented by *n*-byte. The number of nodes for the double array is reduced with *n*'s increasing. But the size of the CODE array is increased. At the 2-gram double array, the size becomes 62-64% and the retrieval speed is 1.18-1.3 times faster than the original double array.

Table III and Table IV show the results for CODE represented by the double array. The rates of the size are compared with the original double array for Table I and Table II. The CODE represented by the double array is smaller than the CODE array of Table I and Table II. The retrieval speed is slower, because the retrieval of CODE takes much time. But, the sizes become 41-59% compared with the original double array.

TABLE III: THE RESULT OF WORDNET FOR N-GRAM DOUBLE ARRAY WITH CODE REPRESENTED BY THE DOUBLE ARRAY.

	1-gram	2-gram	3-gram	4-gram
The number of the nodes for the CODE double array	43	1,284	12,490	52,421
The size of CODE (byte)	215	6,420	62,450	262,105
The number of nodes for the double array	880,273	513,948	391,254	329,309
The size of BASE array (byte)	3,521,092	2,055,792	1,565,016	1,317,236
The size of CHECK array (byte)	3,521,092	2,055,792	1,565,016	1,317,236
Total size (byte)	7,042,399	4,118,004	3,192,482	2,896,577
The rate of the size	0.99	0.58	0.45	0.41
The retrieval time (ms)	48.35	38.16	35.72	35.65

TABLE IV: THE RESULT OF JAPANESE WORDNET FOR N-GRAM DOUBLE ARRAY WITH CODE REPRESENTED BY THE DOUBLE ARRAY.

	1-gram	2-gram	3-gram	4-gram
The number of the nodes for the CODE double array	146	5,713	7,831	61,956
The size of CODE (byte)	730	28,565	39,155	309,780
The number of nodes for the double array	532,330	309,460	263,905	198,053
The size of BASE array (byte)	2,129,320	1,237,840	1,055,620	792,212
The size of CHECK array (byte)	2,129,320	1,237,840	1,055,620	792,212
Total size (byte)	4,259,370	2,504,245	2,150,395	1,894,204
The rate of the size	0.99	0.59	0.50	0.44
The retrieval time (ms)	32.19	24.05	21.23	22.66

V. CONCLUSION

A new double array has been proposed by representing edges as byte *n*-gram. From experimental observations by using single-byte and multi-byte character sets, at the 2-gram double array, the size and the retrieval speed of the proposed method became 62-64% and 1.18-1.3 times, respectively. When the CODE is represented by the double array, the sizes of the proposed method became 41 -59%.

The future works are to apply a compact double array proposed by Yata [12] and to conduct experiments by using large key sets.

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*, vol. 43, no. 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*, vol. 2, pp. 18–27, 2007.

[4] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *Proc. the ACM SIGCOMM*, Vancouver, British Colombia, Canada, 1998, pp. 191-202.

[5] D. Gusfield, *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*, Cambridge University Press, 1998.

[6] 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.

[7] J. Aoe, K. Morimoto, and T. Sato, "An efficient implementation of trie structures," *Software Practice and Experience*, vol. 22, no. 9, pp. 695–721, 1992.

[8] K. Morimoto, H. Iriguchi, and J. Aoe, "A method for compressing trie structures," *Software Practice and Experience*, vol. 24, no. 3, pp. 265–288, 1994.

[9] M. Fuketa, K. Morita, Y. Sumitomo, S. Kashiji, E.-S. Atlam, and J. Aoe, "A new compression method of double array for compact dictionaries," *International Journal of Computer Mathematics*, vol. 81, no. 8, pp. 943-953, 2004.

[10] M. Fuketa, H. Kitagawa, T. Ogawa, K. Morita, and J. Aoe, "Compression of Double array Structures for Fixed Length Keywords," in *Proc. 2nd International Conference on Networking and Information Technology*, Hong Kong, 2011, pp. 158-165.

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

[12] 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*, vol. 43, no. 1, pp. 237–247, 2007.

Masao Fuketa received B.Sc., M.Sc. and Ph.D. Degrees in information science and Intelligent systems from University of Tokushima, Japan in 1993, 1995 and 1998. He has been a research assistant from 1998 to 2000 in information science and Intelligent Systems from University of Tokushima, Japan. He is currently a research associate in the department of Information Science & Intelligent Systems, Tokushima University, Japan. He is a member in the Information processing Society in Japan and The Association for Natural language processing of Japan. His research interests are in Sentence retrieval from huge text data bases and morphological analysis.

Kazuhiro Morita received B.Sc., M.Sc. and Ph.D. Degrees in information science and Intelligent systems from University of Tokushima, Japan in 1995, 1997 and 2000. Since 2000, he has been a research assistant of the Department of the Department of Information Science & Intelligent Systems, Tokushima University, Japan. His research interests are in Sentence retrieval from huge text data bases, double-array structure and Binary search tree.

Jun-Ichi Aoe received B.Sc. and M.Sc. Degrees in electronic engineering from the University of Tokushima, Japan, in 1974 and 1976, respectively, and the Ph.D. degree in communication engineering from the University of Osaka, Japan. Since 1976, he has been with the University of Tokushima. He is currently a professor in the department of Information Science & Intelligent Systems, Tokushima University, Japan. His research interests are in natural language processing, a shift-search strategy for interleaved LR parsing, robust method for understanding NL interface commands in an intelligent command interpreter, and trie compaction algorithms for large key sets. He was the editor of the computer Algorithm Series of the IEEE computer Society Press. He is a member in the association for computing machinery, the association for the natural language processing of Japan.