Comparisons of Efficient Implementations for DAWG

Masao Fuketa, Kazuhiro Morita, and Jun-ichi Aoe

***Abstract*—Key retrieval is very important in various applications. A trie and DAWG are data structures for key retrieval. The double array is one of methods to construct a trie and has both high speed and compactness. In this paper, a data structure of DAWG by the double array using BASE and CHECK is compared with that of DAWG by the double array using CHECK and NEXT, and the retrieval speed and the space usage are theoretically observed. When DAWG and DFA by the double array are constructed, it turns out that it is important to consider indexes for CHECK and NEXT arrays as edge numbers.**

***Index Terms*—Automaton, DAWG, double array, triple array.**

I. Introduction

Key retrieval is used in various applications [1]. A trie is one of data structures to retrieve keys and merge common prefixes of keys [2]. Moreover, Directed Acyclic Word Gragh (DAWG) is a data structure to reduce the number of trie states [3]. DAWG merges common parts of keys. As the trie and DAWG are kinds of Deterministic Finite Automaton (DFA), they can be traditionally represented by a matrix form (transition table) and a linked list.

The triple array is a data structure to construct DFA [4]. This method uses three one-dimensional arrays called BASE, CHECK, and NEXT in order to compress the matrix form. It has high speed because keys of length k can be retrieved by O(k). There is also a method called the double array [5], which compresses the triple array. This method deletes a NEXT array from the triple array and consists of BASE and CHECK. The double array is used in various applications and fields because of its high speed [6], [7]. However, because one state has only one parent state, the original double array can construct a trie but cannot construct DAWG and DFA. Moreover, the compact double array was proposed as a method to compress the double array [8]. This method reduces the space usage by storing traversed characters in CHECK. A method to construct DAWG by the features of the compact double array was proposed [9]. This method has higher speed and less space usage than other methods such as the matrix form, linked list and TST [10]. Furthermore, a method to construct DFA with CHECK and NEXT by deleting a BASE array from the triple array was proposed [11].

In this paper, a data structure of DAWG by the double array using BASE and CHECK(BC DAWG) is compared with that of DAWG by the double array using CHECK and NEXT(CN DAWG), and the retrieval speed and the space usage are theoretically observed. A construction algorithm of CN DAWG is proposed. Moreover, features of DAWG by the double array are discussed.

Manuscript received August 4, 2014; revised November 16, 2014.

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

II. Double Array and DAWG

A. Trie

A trie is a tree structure used for key retrieval in the field of natural language processing, and is a kind of DFA. Fig. 1 shows examples of the trie in key set K = {“aaa”, “aba”, “bbc”, “cbc”, “cc”}. Double circles show terminal states. The trie merges common prefixes of keys. Retrieval always starts from a root state (for example, state number 1 in Fig. 1), and traverses states by one-by-one character in the key. The trie is traditionally represented as a matrix form (transition table) and a linked list. Fig. 2 shows the matrix form for Fig. 1. A vertical axis shows state number s, a horizontal axis shows traversed character c, and matrix [s], [c] shows a state number for the destination of traversal. This matrix form is very sparse.

B. Triple Array

There is a method called the triple array to implement DFA. This method is a data structure which the matrix form is compressed. The triple array uses three one-dimensional arrays called BASE, CHECK, and NEXT. In the triple array, the following three equations are satisfied in the case of traversal from state s to state t in character c;

$$\begin{aligned} e &= BASE[s] + CODE(c) \\ CHECK[e] &= s \\ t &= NEXT[e] \end{aligned} \tag{1}$$

where CODE is numerical values for character c and returns different numbers for each character. An index for the destination of traversal is calculated in the first equation, the traversal is checked if it is correct in the second equation, and NEXT is referred and the next state number is obtained in the third equation.

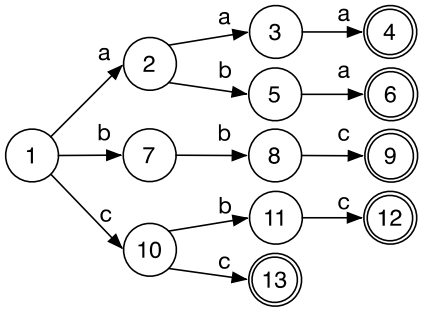


Fig. 1. Trie.

DOI: 10.7763/IJCTE.2016.V8.1018

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

Fig. 2. Matrix form.

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

	a	b	c
CODE	1	2	3

Fig. 3. Triple array.

As traversal is calculated by (1), the key with length k can be retrieved by O(k). Furthermore, it can also construct the trie, because the triple array has a data structure to construct DFA. Fig. 3 shows the triple array in key set K.

C. Double Array

There is a method called the double array, which reduces the space usage of the triple array. The double array reduces NEXT of the triple array by $t = e$ = NEXT[e]. The following two equations are satisfied in the case of traversal from state s to state t in character c;

$$\begin{aligned} t &= BASE[s] + CODE(c) \\ CHECK[t] &= s \end{aligned} \tag{2}$$

As NEXT is deleted, the space usage of the double array is small and its retrieval speed is improved. In the same manner of the triple array, the key with length k can be retrieved by O(k). Moreover, the compact double array was proposed by Yata. This method is a data structure to compact the size of CHECK by storing traversed characters in CHECK. In the compact double array, the following two equations are satisfied for traversal;

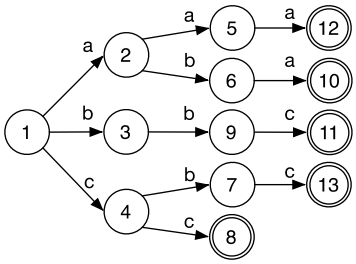
$$\begin{aligned} t &= BASE[s] + CODE(c) \\ CHECK[t] &= c \end{aligned} \tag{3}$$

However, in different states s and s', if BASE[s] and BASE[s'] are the same, these are traversed to the same state t. In the compact double array, all BASE values need to be constructed as different values. Fig. 4 shows the compact double array in key set K. CODE uses the same value as Fig. 3. State numbers in Fig. 4 are different from state numbers in Fig. 1 because the values of two arrays are determined to satisfy (3). This double array is defined as the BC double array.

As the parent state of state t is only state s, these double arrays cannot construct DFA.

D. DAWG

DAWG is possible to have less number of states than a trie because it merges all common substrings. Fig. 5 shows DAWG for key set K. In the same manner of a trie, retrieval of DAWG starts from the root state. As DAWG is a kind of DFA, it is possible to be represented as a matrix form and a linked list. In DAWG, terminal states cannot keep records for each key because some keys reach to the same terminal states.



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

Fig. 4. Compact double array.

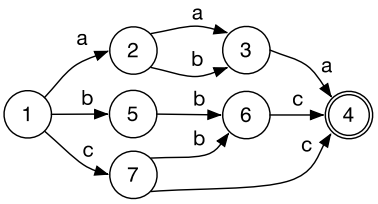


Fig. 5. DAWG.

III. DAWG Using Compact Double Array

Yata proposed a method to construct DAWG by the compact double array. In the double array, indexes of BASE and CHECK are represented as the state number.

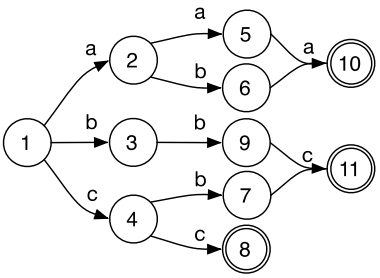


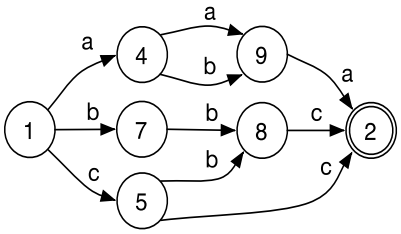
Fig. 6. Custom designed DAWG.

From (3), only character c can traverse to state t. Therefore, different characters cannot traverse the same state; for example, edges of characters 'a' and 'b' from state 2 to state 3 in Fig. 5 cannot be represented by the BC double array. However, it can traverse from states s and s' to state t by character c if the same values are stored in BASE for states s and s'($s \neq s$'), because the compact double array stores traversed characters in CHECK. In other words, because of BASE[s] = BASE[s'], the values of BASE[s]+CODE(c) and BASE[s']+CODE(c) are the same value t and are possible to satisfy CHECK[t]=c. DAWG in Fig. 5 modified by the above-mentioned features is shown in Fig. 6. State 3 in Fig. 5 corresponds to states 5 and 6 in Fig. 6 and traverses from states 5 and 6 to state 10 by the same character 'a'. Also, Fig.

7 shows the double array form of DAWG in Fig.6. Hereafter, this DAWG is defined as BC DAWG. Although the number of states in Fig. 6 seems to be more than the number in Fig. 5, the number is considered not to increase because of the same state as state 5 and 6 in Fig. 6. DAWG needs to be changed in order to construct BC DAWG as the changes from Fig. 5 to Fig. 6.

	1	2	3	4	5	6	7	8	9	10	11
BASE	1	4	7	5	9	9	8		8		
CHECK		a	b	c	a	b	b	c	b	a	c

Fig. 7. BC DAWG.



	1	2	3	4	5	6	7	8	9	10	11
CHECK		a	b	c	a	b	b	c	b	a	c
NEXT	1	4	7	5	9	9	8	2	8	2	2

Fig. 8. CN DAWG.

IV. DAWG Using CN Double Array

A. Data Structures

Maeda proposed a data structure to represent DFA by the double array [11]. The paper by Maeda explained that BASE was deleted by storing BASE values in NEXT. This means BASE is deleted by s=BASE[s] in the same manner of Fuketa's method [12]. Therefore, (1) in the triple array is changed as follows;

$$\begin{aligned} e &= s + CODE(c) \\ CHECK[e] &= s \\ t &= NEXT[e] \end{aligned} \tag{4}$$

As traversal to the same state is possible by storing the same value in NEXT, DFA can be constructed by the double array with CHECK and NEXT. In this double array, CHECK[e] stores state number s for the source of traversal, and NEXT[e] stores state number t for the destination of traversal. Moreover, CODE values for traversed character c from s to t can be calculated by CODE(c)=e-CHECK[e] with considering (4).

Different states don't become the same values because values stored in NEXT are the state numbers. Traversed character c can be stored in CHECK[e] in the same manner of the compact double array. In this double array, traversal is satisfied with the following equation;

$$\begin{aligned} e &= s + CODE(c) \\ CHECK[e] &= c \\ t &= NEXT[e] \end{aligned} \tag{5}$$

The double array to implement DFA is defined as the CN double array. As DAWG is a kind of DFA, DAWG is represented by the CN double array.

This DAWG is defined as the CN DAWG in order to distinguish from the method proposed by Yata. Fig. 8 shows the CN DAWG built for key set K. In the state diagram shown in Fig. 5, state numbers are different from Fig. 8, but the shape of the figure is the same. In other words, it is possible to construct DAWG without construction of custom designed DAWG.

B. Construction Algorithm

In this subsection, a construction algorithm of CN DAWG is described. After DAWG is constructed by the matrix form, it is converted into the CN double array. In the algorithm, the following variables and functions are used;

- ROOT STATE NUMBER: A root state number in a matrix form
- PUSH (S_SET,s): To add state s to set S_SET.
- POP (S_SET): To return one value from set S_SET.
- GET_LABELS(s): To return all traversed characters from state s.
- N [s]: An array to store a state number of CN DAWG corresponding to state s in the matrix form.
- X_CHECK (C_SET): A function returns a minimum value of states s which s+CODE(c) (c in C_SET) are all unused indexes. Simultaneously, index=s+CODE(c) is set as used indexes.

```
(C- 1): STATE_SET = ∅;
(C- 2): PUSH(STATE_SET,ROOT_STATE_NUMBER);
(C- 3): NEXT[1]=X_CHECK(GET_LABELS(ROOT_STATE_NUMBER));
(C- 4): N[ROOT_STATE_NUMBER] = NEXT[1];
(C- 5): while(STATE_SET ≠ ∅){
(C- 6):     s = POP(STATE_SET);
(C- 7):     C_SET = GET_LABELS(s);
(C- 8):     for c in C_SET{
(C- 9):         t = matrix[s][c];
(C-10):         if(N[t] is not defined){
(C-11):             PUSH(STATE_SET, t)
(C-12):         }
(C-13):         e = N[s] + CODE(c);
(C-14):         CHECK[e] = c;
(C-15):         if(N[t] is defined){
(C-16):             NEXT[e] = N[t];
(C-17):         }
(C-18):         else{
(C-19):             NEXT[e] = X_CHECK(GET_LABELS(t));
(C-20):             N[t] = NEXT[e];
(C-21):         }
(C-22):     }
(C-23): }
```

Fig. 9. Construction algorithm.

Fig. 9 shows a construction algorithm. This algorithm can construct both DAWG and DFA. In (C-3), a root state of CN DAWG is set to the NEXT array. The loop of (C-5) repeats for all states in DAWG. The loop from (C-8) to (C-22) sets the CHECK and NEXT values for all child states of state number s. In (C-10), the same state are not processed twice. In (C-15)-(C-21), if t is processed for the first time, a new state number is searched by X_CHECK. If not, values which have already been set are used.

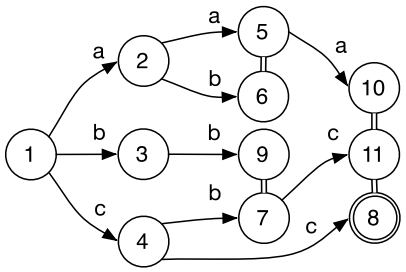
V. Related Work

Aoe proposed a method to construct DFA using the double

array by BASE and CHECK[13]. In the double array, as only one parent state is defined, a data structure which represents different states as the same states is necessary in order to traverse the same state by different characters. In this method, different states are once traversed by different characters, and then the traversal from one side to the other side is stored in BASE. The traversal to the same state is represented as BASE[*s*]<0, because BASE usually stores positive values. For example, in the custom designed DAWG of Fig. 6, states 5 and 6 are the same states. By defining BASE[6] =-5 for states 5 and 6, it shows that these states are the same states. Therefore, the traversal from state *s* to state *t* by character *c* in the double array is conducted by the following conditions;

1) BASE[*s*]≧0
   t = BASE[*s*]+CODE(*c*)
   CHECK[*t*] = *s*
2) BASE[*s*]<0
   s' = -BASE[*s*]
   t = BASE[*s*']+CODE(*c*)
   CHECK[*t*] = *s*'

Fig. 10 shows this double array representation of key set K. Fig. 10 shows that states (5 and 6; 7 and 9; 8, 10, and 11) are the same states. If positive values stored in BASE are unique, this double array can change into the compact double array. As this data structure needs to judge positives or negatives for each traversal, the retrieval speed of this data structure is slower than that of the BC double array and CN double array.



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

Fig. 10. DAWG for Aoe's method.

VI. Observations

BC DAWG merges traversal of the same characters by the feature to store traversed characters in CHECK of the BC double array. Therefore, the original DAWG is one state, but BC DAWG can be more than two states. For example, state 4 in Fig. 5 becomes states 8, 10, and 11 in Fig. 6. However, as retrieval can be conducted by *k* times of traversal if length of a retrieval key is *k*, time complexity becomes O(*k*) and is very fast. In the case of large key sets, as BASE and CHECK respectively requires 4 bytes and 1 byte when the number of all states is *n*, the space usage becomes 5*n* bytes. In this method, DFA which traverses from state *s* to state *s* cannot be constructed.

Because CN DAWG uses the CN double array which can deal with DFA, the same state diagram can be represented as the original DAWG. Its retrieval speed is very fast, and its time complexity for a key of length *k* is O(*k*). When the custom designed DAWG for BC DAWG in Fig. 6 is compared with the state diagram in Fig. 8, the state number of Fig. 8 is less, but the number of indexes in arrays is the same in BC DAWG of Fig. 7 and CN DAWG of Fig. 8. As the number of indexes in arrays except for the first index matches the number of edges in CN DAWG, indexes of CHECK and NEXT arrays are considered as the edge numbers. For example, character 'a' (=CHECK[2]) traverses from state 1 to state 4 （=NEXT[2]） through edge number 2. Variable *e* in (5) represents the edge number. In BC DAWG, the number of all states is the same as the number of indegrees of all states, when the number of edges with the same character (for example, traversal from states 5 and 6 to state 10 in Fig. 6) is counted as 1. In other words, *n* is considered as the number of edges in BC DAWG as well as CN DAWG. In CN DAWG, as CHECK and NEXT respectively requires 1 byte and 4 bytes when the number of edges is *n*, the space usage becomes 5*n* bytes. The space usage of BC DAWG is the same as that of CN DAWG. This means BC DAWG and CN DAWG have the same speed and space usage.

Moreover, when BASE in BC DAWG is compared with NEXT in CN DAWG, it finds out that they are exactly the same values except for values of NEXT to terminal states. However, BC DAWG needs to create the custom designed DAWG. The reason is indexes of the double array are considered as state numbers because there is a limitation which one state has only one parent state in the double array. BC DAWG cannot construct DFA because of the limitation. Therefore, it is important to consider indexes of the double array as edge numbers in order to construct DAWG and DFA in the double array.

VII. Conclusion

This paper has compared a data structure of DAWG by the double array using BASE and CHECK arrays with that of DAWG by the double array using CHECK and NEXT arrays, and has mentioned that the retrieval speed and the space usage are the same. Moreover, when DAWG and DFA by the double array are constructed, it turns out that it is important to consider indexes for CHECK and NEXT arrays as edge numbers.

A future work is to propose a method to conduct a dynamic construction by CN DAWG.

References

[1] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333-340, 1975.
[2] D. E. Knuth, "The art of computer programming," *Sorting and Searching*, vol. 3, Addison-Wesley, 1972.
[3] A. V. Aho, J. D. Ullman, and J. E. Hopcroft, *Data Structures and Algorithms*, Addison Wesley, 1983.
[4] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1985.
[5] J. Aoe, "An Efficient digital search algorithm by using a double-array structure," *IEEE Transactions on Software Engineering*, vol. 15, no. 9, pp. 1066-1077, 1989.
[6] Z. Yuan, B. Yang, X. Ren, and Y. Xue, "TFD: a multi-pattern matching algorithm for large-scale URL filtering," in *Proc. Intl. Conf. Computing, Networking and Communications, Communications and Information Security*, 2013, pp. 359-363.

[7] C. Zheng, Q. Zheng, Z. Zhou, and F. Tian, "A method for large cross-language lexicon management based on collaborative work of hash family and double-array trie," in *Proc. 14th International Conference on Computer Supported Cooperative Work in Design*, New York, 2010, pp. 658-663.
[8] S. Yata, M. Oono, K. Morita, M. Fuketa, T. Sumitomo, and J. Aoe, "A compact static double-array keeping character codes," *Information Processing and Management*, vol. 43, no. 1, pp. 237-247, 2007.
[9] S. Yata, K. Morita, M. Fuketa, and J. Aoe, "Fast string matching with space-efficient word graphs," *Innovations in Information Technology*, pp. 79-83, 2008.
[10] J. L. Bentley and R. Sedgewick, "Fast algorithms for sorting and searching strings," in *Proc. Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 360-369.
[11] A. Maeda and K. Mizushima, "A compressed-array representation of automata and its application to programming language," in *Proc. the 49th Programming Symposium on Information Processing Society of Japan*, 2008, pp. 49-54.
[12] M. Fuketa, H. Kitagawa, T. Ogawa, K. Morita, and J. Aoe, "Compression of double array structures for fixed length keywords," *Information Processing & management*, vol. 50, no. 5, pp. 796-806, 2014.
[13] J. Aoe, "An efficient implementation of finite state machines using double-array structures," *The IEICE Transactions on Information and Systems*, vol. J70-D, pp. 653-662, 1987.

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, respectively. He had been a research assistant from 1998 to 2000 in information science and intelligent systems, University of Tokushima, Japan. He is currently an associate professor in the Department of Information Science and Intelligent Systems, University of Tokushima, Japan. He is a member of the information processing society in Japan and the association for natural language processing of Japan. His research interests are information retrieval and natural language processing.

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, respectively. Since 2006, he has been a research associate in the Department of Information Science and Intelligent systems, University of Tokushima, Japan. His research interests are sentence retrieval from huge text databases, double array structures 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 received the Ph.D. degree in communication engineering from the University of Osaka, Japan, in 1980. Since 1976, he has been with the University of Tokushima. He is currently a professor in the Department of Information Science and Intelligent Systems, University of Tokushima, Japan. His research interests are natural language processing, a shift-search strategy for interleaved LR parsing, a 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 of the association for computing machinery and the association for the natural language processing of Japan.