A Compression Method Using Link-Trie Structure for Natural Language Dictionaries

Masaki Oono, Masako Fuketa, Yutaka Inada, Yo Murakami and Jun-ichi Aoe

Department of Information Science & Intelligent Systems
Faculty of Engineering, University of Tokushima
2-1 Minami-Josanjima, Tosukushima, 770-0806, JAPAN
E-mail: oono@ccr.tokushima-u.ac.jp

Abstract - A trie structure is a key search method that is frequently used in the field of Natural Language Systems and Database Systems. Morita presented an efficient data structure for collocation information using trie structure. This structure stores two basic words into a trie and defines link information by a link function. This paper presents how to apply Morita's method into general key-set and compress trie. Presented method divides a key into several parts and defines link information between keys. From simulation results, it turned out that presented method is 30% smaller than Conventional method in space capacity.

I. INTRODUCTION

Recently, more and more mobile applications such as cellular phone and PDA (Personal Digital Assistance) are being developed. In these fields, space capacity is often limited. Therefore, it is important to use a fast and compact structure in developing an application in these fields.

A trie structure[1][2][3] is a key search method that can retrieve information fast regardless of the size of dictionary. Trie also can easily detect a prefix and a mismatch point of registered keys. Therefore, trie is frequently used in the field of natural language processing such as morphologic analysis[4] and bibliographic search[5]. If more compact structure is presented without losing its fastness, it will be useful in several fields. Morita presented an efficient structure for storing collocation information using a trie structure[7]. This structure stores two basic words into a trie and defines collocation information by a link function.

This paper presents how to apply Morita's method into general key-set and compress trie. Presented method divides a key into several parts and defines link information between keys. From simulation results, it turned out that presented method is 30% smaller than old method in space capacity.

II. TRIE STRUCTURE

The trie structure is a tree structure which compresses common prefixes of keys[6]. Key is searched by the unit of character. Therefore if transition of a character is done in the time of O(1), you can retrieve information fast regardless of the size of dictionary. "#", which is not used in key, is added at the end of key as end-sign in trie structure to correspond key to trie-node one for one. Transition from node s to node t by character 'a' is defined as g(s, 'a')=t. If no transition by character 'a' is made, it's defined as g(s, 'a')=fail. Function g is also used for character string X. In that case, it's defined as g(s, X)=t. Unless otherwise mentioned, 'a', 'b', 'c', ... are used as characters. X, Y and Z are used as character strings. Also X#, Y#, Z# are used as character strings which have end-signs. An example of trie structure for key set K={"back", "bad", "be", "beef"} is shown in Fig. 1.

III. HOW TO MANAGE COLLOCATIONAL INFORMATION USING TRIE STRUCTURE

A. How to Manage Link Information

This section explains about link-trie structure. Link-trie structure is a structure which manages collocation information efficiently. Suppose collocation information is XY, then collocation information in this section means relational information α between basic word X and Y. And it is represented as (X, Y, α).

First, key X, Y is stored in a trie respectively. Secondly, link is defined in leaf node X to define collocation information between these keys. Therefore this trie is called a link-trie. Suppose such s that g(1, X#)=s, then leaf node s corresponds to X one for one. Therefore record information about key X can be stored at corresponding record of node s. Hence, suppose such node s that f(s)∋t and such node t that g(1, Y#)=t and there is a link from s toward t in link-trie, then collocation information α between X and Y is stored as an element of record information REC(s, t).

Link information is shown in Table I when collocation information is registered in link-trie for collocation key-set K2={"Japan", "Nation", α }, ("Japan", "People", α ), ("China", "Nation", α ), ("Tokushima", "University", α ), ("Tokushima", "City", α ), ("Tokushima", "China", α )}.

Link information defined by function f(s) is all obtainable by criterion X as shown in Table I. This means you can search not only (X, Y, α ) but also can search all Y and collocation information whose relation is defined. This is the characteristics of link-trie structure which compresses common prefix of keys.
that has basic key and its collocation information at the same time.

B. Double-Array

This section explains about data structure of the link-trie. Link-trie consists of trie part which stores key information, function \( f(s) \) which stores link information from leaf node \( s \) and \( \text{REC}(s,t) \) which stores collocation information.

Double-Array structure is compact as well as fast. Double-Array consists of two arrays which are called \( \text{BASE} \) and \( \text{CHECK} \). \( \text{BASE} \) defines offset to a transition target. \( \text{CHECK} \) defines a node number from which a node transits. Suppose inner code of character \( 'a' \) is defined as \( \text{N}(a') \), then transition to node \( t \) from node \( s \) by character \( 'a' \) is defined as follows:

\[
t = \text{BASE}[s] + \text{N}(a') \tag{1}
\]

Just like this, transition to node \( t \) from node \( s \) by character \( 'a' \) in Double-Array is defined by the sum between \( \text{BASE}[s] \) and the inner code of character \( 'a' \) and also by storing \( s \) in \( \text{CHECK}[t] \).

A transition in Double-Array takes time of \( O(1) \). Therefore, key search in Double-Array is quite fast because it is only dependent on the key length. If node \( n \) is a leaf node of trie, it is distinguished from other keys by defining \( \text{BASE}[n] = 0 \). By doing this, it can be decided whether key search is successful or not. Additionally, because each index of Double-Array corresponds to each node one for one, this paper explains without discriminating them.

Double-Array for Fig. 1 is shown in Fig. 2. Suppose that inner code of \( '#' \) is 1 and character \( 'a' \) is \( 2 \) is \( 2 \) respectively. Transition to node 4 from initial node 1 by character \( 'b' \) is confirmed valid by \( \text{BASE}(1) + \text{N}(b') = 4 \), \( \text{CHECK}[4] = 1 \), which complies with equation (1). Additionally, node 2, 3, 7 and 9 is confirmed to be the end of a key because it has a minus value in \( \text{BASE} \) respectively.

Following shows how to search a key "bad" in Double-array shown at Fig. 2. First, transition destination of \( s = 1 \) is obtained as 4 by \( t = \text{BASE}[1] + \text{N}(b') = 4 \). Transition to node 4 from node 1 by character \( 'a' \) is confirmed by \( \text{CHECK}[4] = 1 = s \). In the same way, transition by character \( 'a', 'd' \) and \( '#' \) are confirmed. At last by equation \( \text{BASE}[7] = -2 < 0 \), node 3 is confirmed a leaf node, which means search was successful.

C. Link Function

Link function is implemented by Double-Array \( \text{D}(f(s)) \) also to search quite fast as follows. Suppose inner codes \( w \) of node \( t \) which is an element of \( f(s) \) and inner codes \( v \) such \( \alpha \) that \( \text{REC}(s,t) \supseteq \alpha \), then \( \text{D}(f(s)) \) which has \( vw \) as an arc is defined. Additionally each length of \( v, w \) is fixed to determine solely \( v \) and \( w \) from \( vw \). Leaf node of trie corresponds to key one for one because each length of \( v, w \) is fixed. Therefore there is no need to make end sign "#" in link-trie.

IV. A COMPRESSION METHOD FOR DOUBLE-ARRAY USING LINK INFORMATION

This section presents how to apply the method which is explained in section III to general key set and compress trie. This method consists of key division and link function between divided keys.

A. Key Division Method

The basic idea is that a word consisting of several basic words is divided into basic words. For example, a word "football" can be considered to consist of "foot" and "ball". There are many words like this example and a method which divides a key into basic words is necessary.

The key division method is that a key will be divided when brother node is made while making a trie. To be made a brother node means that two keys share nodes until brother node. In this point, shared nodes most likely make a basic word. Therefore, this method is appropriate.
TABLE II.
The Simulation Results for 100,000 Keys.

<table>
<thead>
<tr>
<th>Key set</th>
<th>English</th>
<th>Japanese</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average length (byte)</td>
<td>12.21</td>
<td>12.4</td>
</tr>
<tr>
<td>Longest length (byte)</td>
<td>61</td>
<td>70</td>
</tr>
<tr>
<td>Space capacity (MByte)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conventional method</td>
<td>5.49</td>
<td>5.44</td>
</tr>
<tr>
<td>New method</td>
<td>4.20</td>
<td>3.71</td>
</tr>
<tr>
<td>Trie part</td>
<td>2.01</td>
<td>1.72</td>
</tr>
<tr>
<td>Link function part</td>
<td>2.18</td>
<td>1.99</td>
</tr>
<tr>
<td>Search time (ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conventional method</td>
<td>1.5</td>
<td>1.6</td>
</tr>
<tr>
<td>New method</td>
<td>2.9</td>
<td>2.8</td>
</tr>
<tr>
<td>Insertion time (ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conventional method</td>
<td>9.1</td>
<td>9.6</td>
</tr>
<tr>
<td>New method</td>
<td>9.7</td>
<td>10.0</td>
</tr>
<tr>
<td>Deletion time (ms)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conventional method</td>
<td>1.5</td>
<td>1.6</td>
</tr>
<tr>
<td>New method</td>
<td>2.4</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Fig. 4. The simulation results for space capacity.

V. EVALUATION

A. THEORETICAL OBSERVATION

Worst case of transition in trie by Double-Array takes time of O(1) and worst case of key search depends on the length of a key. It's quite fast regardless of number of keys. Time to search link information depends on Hash function. Suppose the number of link coming from leaf node s of basic word is n and hash function returns node number s. Then worst search time becomes O(n).

Trie part is implemented by Double-array. Its space capacity is proportional to number of nodes, e and it is represented by O(e). It is proved to be sufficiently compact. Space capacity of link information is dependent on the number of link. The more keys are inserted, the less nodes are added to trie. But link information increases in proportion to the number of link. Therefore space capacity is represented by O(n) if total number of link is represented by n.

B. EXPERIMENTAL OBSERVATION

Presented method is implemented by about one thousand lines in C language and it is run on DELL OPTIPLEX GX870 (CPU: Pentium 4 [3.0GHz]). Comparison simulation is conducted to confirm efficacy of presented method. Simulation used Japanese key set which are extracted from Mainichi Newspaper in year 2004 and WordNet dictionary[8]. Japanese letters are represented in 2 bytes. Therefore in this simulation, successive two transitions are defined from gins, node number t where link ends, additional information \( \alpha \). Chaining method is used as a way to implement Hash method.

Link function between "ba" and "ck" from Fig. 3 are made as follows. Hash value is made from Hash function by 2, which is leaf-node number of "ba" and 10, which is leaf-node number of "ck" and additional information \( \alpha \). Suppose Hash(s, t, \alpha) = 3, then node which includes 2, 10, \alpha are added to hash table[3rd].

But many one-length keys would be made if a key were divided every time a brother node is made. Hence, threshold value is set to prevent this. Key will be divided only when two keys share more nodes than threshold value.

Fig. 3 shows trie which applies key division method for \( K=\{"back", "bad", "be", "beef"\}, \) which is used in Fig. 1. Threshold value is set as 2 bytes. Suppose "bad" is inserted after "back" is inserted. "bad" has 'b' and 'a' in common with "back" but don't have 'd' in common. Therefore a brother node is made and "bad" are divided into "ba" and "d", "back" are divided into "ba" and "ck".

B. LINK FUNCTION METHOD

The method explained in section III adopted Double-Array structure as link function. But adopting Double-Array structure have problems in defining link function in new method. Only one link function is made to define one collocation information in collocation link. On the other hand, new method defines more than one link because key is divided into several parts. Undefined key can be retrieved because of this. Therefore additional information is also added to prevent this. Link information consists of node numbers and additional information which are stated above. Therefore defining many links cause space capacity to balloon. Additionally because key division method doesn't guarantee perfect division, inefficient link can be made and it is inappropriate to adopt Double-Array as link function structure. Given this factor, new method adopts Hash method. Hash value is made from Hash function by node number s where link begins, node number t where link ends, additional information \( \alpha \). Chaining method is used as a way to implement Hash method.

Link function between "ba" and "ck" from Fig. 3 are made as follows. Hash value is made from Hash function by 2, which is leaf-node number of "ba" and 10, which is leaf-node number of "ck" and additional information \( \alpha \). Suppose Hash(s, t, \alpha) = 3, then node which includes 2, 10, \alpha are added to hash table[3rd].
one Japanese letter because transition unit in trie is 1 byte. Table II shows the simulation results. It shows that presented trie is about 30 percent smaller than old trie in space capacity. The cause is that once basic key is inserted in trie, there is no need to add the key but define link. Insert time by presented method is about the same as conventional method. Search time by presented method takes more time than conventional method. Fig. 4 shows changes in space capacity by 100,000 keys from Mainichi Newspaper. When 400,000 keys are inserted, presented method is about 68 percent of conventional method in space capacity. This shows efficacy of presented method.

VI. CONCLUSION

A new method applying Morita's method into general key-set has been proposed and implemented. And it turned out that new method is efficient in space capacity. In future study, it will be considered how to develop faster link structure.

REFERENCES