Dynamic Hashing Algorithm for Retrieval Using Hangeul Name on Navigation System

Jung-Hwa Lee, *Member, KIMICS*

***Abstract*— Recently, a name retrieval function is widely used on navigation systems. In this paper, we propose the new dynamic hashing algorithm for a name retrieval function on it. The proposed dynamic hashing algorithm by constructing an index using the variance information of character is the better than existing methods in terms of storage capacity and retrieval speed. The algorithm proposed in this paper can be useful on systems that have limited resources as well as navigation systems.**

***Index Terms*—dynamic hashing, retrieval system, hangeul search, navigation**

I. INTRODUCTION

RECENTLY the more efficient method to find the target data was needed in a retrieval system dealing with increasing amounts of data[1,2]. For quick data retrieval, the most common method is that create index on the search key. Especially, as a navigation system is a widely used, a data retrieval system using hangeul name is also widely used on navigation systems. A hangeul name data has the many linguistic characteristics, so we can improve a performance of retrieval system if consider these properties[3,4].

Many studies have already been done how to create an index. B^+-tree, Trie and etc are used of creating index generally[5,6,7]. However, these methods are not easy to use on system that has limited resources like navigation system because the index size itself is very large.

We can use hashing instead of indexing also. Hashing algorithm is faster than the index to provide search capabilities. But much larger address space than the amount of hashing data is needed, so hashing algorithm difficult to use in many cases, despite the advantage of faster search speed[8].

In this paper, we propose the dynamic hashing algorithm to hangeul name search that are available in environment has limited resources such as navigation systems.

We find out a target hash bucket using index in dynamic hashing. Dynamic hashing is needed less size of index than index method like B^+-tree or trie, etc and retrieval speed is faster than that, so this algorithm can be useful in navigation system[9].

The proposed dynamic hashing algorithm by constructing an index using the variance information of character is the better than existing methods in terms of storage capacity and retrieval speed.

Manuscript received May 23, 2011; revised June 1, 2011; accepted June 7, 2011

Jung-Hwa Lee is with the Department of Computer software engineering, Dongeui University, Busan, 614-714, Korea (Tel:+82-51-890-1729, Fax:+82-51-890-1724, Email:junghwa@deu.ac.kr)

II. RELATED WORK

Dynamic hashing has the advantages and benefits of an index and hashing mentioned in the introduction, it consists of indices and buckets that data are stored in.

Especially, Dynamic hashing is more useful in case of application that have to deal with large amount of data[10,11].

Fig. 1 shows a file structure of general dynamic hashing.

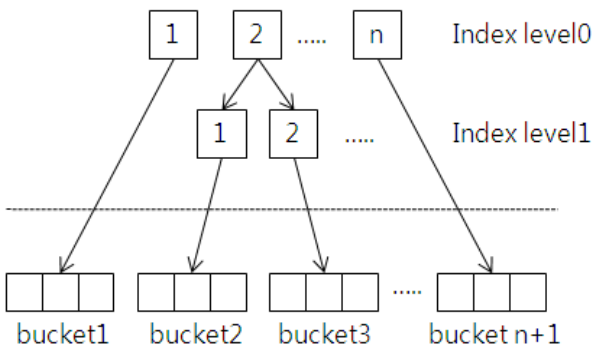

Fig. 1. File structure of general dynamic hashing.

Dynamic hashing only can load a limited amount of data in each bucket, so in case of a number of synonym exceeds a number of data can be stored in bucket, it has to be split dynamically. Also, a pointer to the bucket should be adjusted so that each split bucket can be picked up.

Generally, we make index using the value of hash function in dynamic hashing, but we can use the relative index of hangeul character instead of the value of hash function in hangeul dynamic hashing.

Therefore, if we want to find word '문화', we have to find syllable-initial character 'ㅁ' in first syllable at index level 0. And if there is a pointer to bucket contains target word, we can find the word at that bucket using common search algorithm like linear search or binary search. Else if there is a pointer to next level index, also

find syllable-peak character 'ㅜ' at index level 1, because already the bucket containing word starting with syllable-initial character 'ㅁ' are split.

III. DESIGN OF THE DYNAMIC HASHING ALGORITHM USING VARIANCE INFORMATION BY LOCATION OF NAME DATA.

A. Variance information by location of name data.

Hangeul syllable(like '공', '과') is composed of syllable-initial character ('ㄱ'), syllable-peak character ('ㅗ'), and syllable-final character ('ㅇ') or syllable-initial character ('ㄱ') and syllable-peak character ('ㅘ'). These are complete hangeul syllables. Each character in complete hangeul syllable has a different frequency distribution depending on the location[12]. In case of index using relative index value of hangeul character for dynamic hashing, the distribution of index character frequency is the more uniform, the more data can be loaded evenly in the bucket, in other words, can reduce a overflow. So, in this paper, we use the variance information by location of name data for growing up the performance of storing data. Finally, the proposed method improves a performance of dynamic hashing.

The hangeul name data has very characteristic properties. That is, almost name data composed of two hangeul syllables as shown in Fig. 2. In this case, usually appears at a rate of 98% among name data.

한국 + 초등학교, 동양 + 주식회사, ...

Fig. 2. Structure of hangeul name data.

So, we use the location specific frequency information of characters in first syllable and second syllable considering of this characteristic properties.

Fig. 3 and Fig.4 shows the frequency of syllable-initial, syllable-peak, syllable-final character in first syllable. In Fig.3 and Fig.4, the index on x-axis means a character index. For example, the index value of syllable initial character 'ㄱ' is 1, and the index value of character 'ㄴ' is 2(TABLE 1).

TABLE 1
INDEX OF SYLLABLE INIT-PEAK-FINAL CHARACTER

INDEX	SYLL-INIT	SYLL-PEAK	SYLL-FINAL
1	ㄱ	ㅏ	FILL
2	ㄲ	ㅐ	ㄱ
3	ㄴ		
:	:	:	:
17	ㅌ	ㅟ	ㅁ
18	ㅍ	ㅠ	ㅂ
19	ㅎ	ㅡ	ㅂㅅ
20		ㅢ	ㅅ
21		ㅣ	ㅆ
22			ㅇ
23			ㅈ
24			ㅊ
25			ㅋ
26			ㅌ
27			ㅍ
28			ㅎ

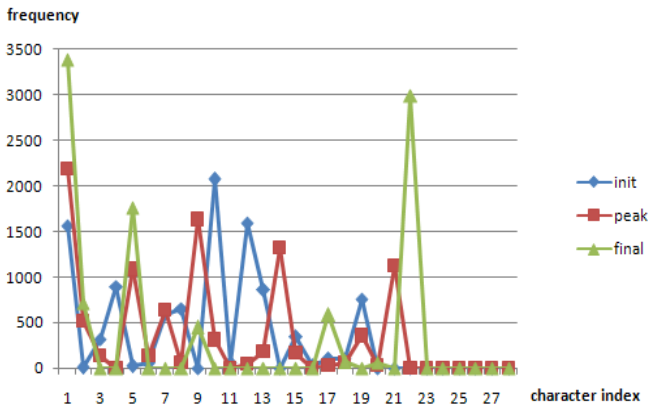

Fig. 3. Frequency of syllable-initial, syllable-peak, syllable-final character in first syllable.

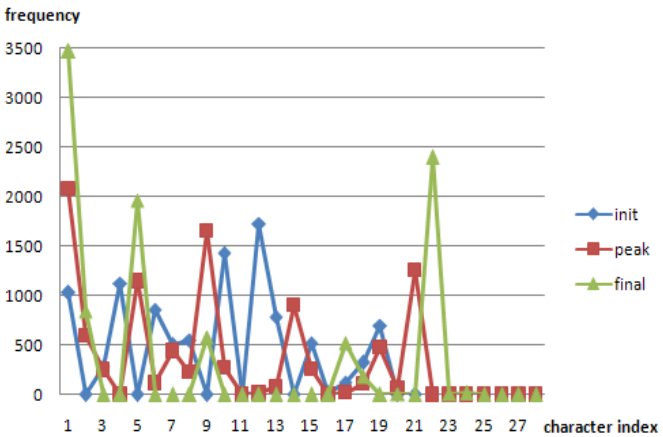

Fig. 4. Frequency of syllable-initial, syllable-peak, syllable-final character in second syllable.

As you see above, the difference of frequency of syllable-final character is the most in the first syllable and the difference of frequency of syllable-initial character is the least, relatively. That is to say, the first syllable's

syllable-initial character has more uniform distribution of character frequency than syllable-final character in the first syllable. Also, the difference of frequency of syllable-initial character is the least in the second syllable.

Therefore, we can prevent an index from increasing level through using index character that has uniform distribution preferentially, source data is also stored in the buckets uniformly.

Various statistical values can be used for measurement of uniformity. In this paper, we use the variance value of character. A variance is the sum of difference average between each value. If the average of x let E(x) then a variance defined as:

$$V(x) = E(x^2) - E(x)^2 \qquad (1)$$

Overall, the variance value of character frequency is small means show an even distribution on the basis of average.

Fig. 5 shows variance value of syllable-initial, peak, final character of two syllables.

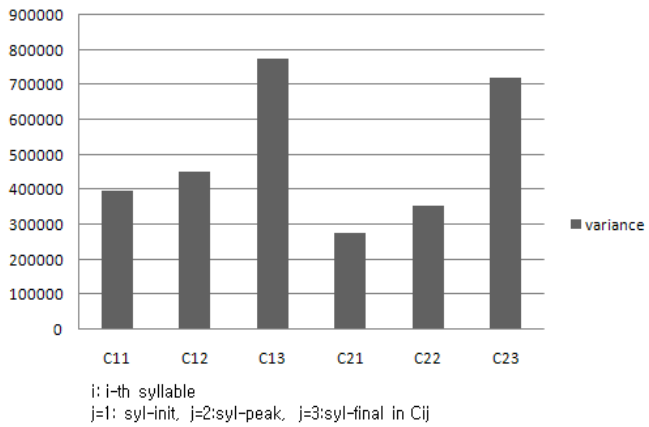

Fig. 5. variance of syllable-initial, peak, final character of two syllables.

B. Hangeul dynamic hashing using variance of character

As mentioned earlier, the distribution of characters that has smaller variance value are uniform, so if we select that character as an index for dynamic hashing, a name data can be stored in bucket uniformly. Therefore, we select characters are used as index according to the ascending order of value of variance in our proposed method as Fig. 6.

C_{21} - > C_{22} -> C_{11} -> C_{12} -> C_{23} ->C_{13} ....

Fig. 6. The order of selected character for index.

Algorithm 1 shows the algorithm of constructing the proposed dynamic hashing structure.

Algorithm 1 : the algorithm of constructing The proposed dynamic hashing structure.

Sw : source word
S_n : series of characters (Fig. 6)
Bk : bucket
I_n : level n of index
Method:

```
1: begin
2:   if In[Sn]->Bk is not full
3:     Store Sw in I[Sn]->Bk
4:   else if In[Sn]->Bk is full
5:     create new index level In+1using Sn+1
6:     split Bk to two new buckets
7:     adjust In+1[Sn+1] to point new buckets
8:     re-hashing Sw in Bk to new buckets
                                  using Si+1
9 :    Store Sw in In+1[Sn+1]->Bk
10:  end if
11: end
```

The structure of proposed hangeul dynamic hashing in accordance with algorithm 1 is shown in Fig. 7.

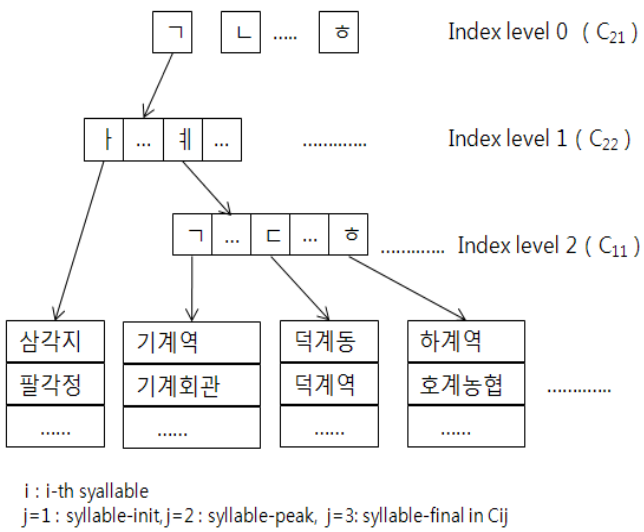

Fig. 7. The structure of proposed hangeul dynamic hashing.

The procedure of searching name data under the proposed method is as follows. In case of searching name data, '하계역' , first, we find syllable-initial character 'ㄱ' of second syllable '계' in the index level 0 and also find syllable-peak character 'ㅖ' of second syllable '계' in the index level 1, second and finally, syllable-initial character 'ㅎ' of first syllable '하' in the index level 3. And then we can go to the bucket is stored the target word '하계역' using bucket pointer on index. Finally, we look up a target word on the bucket using search algorithm.

Algorithm 2 shows the algorithm of searching target word under the proposed method.

Algorithm 2 : the algorithm of searching word

Tw : terget word
S_n : series of characters (Fig. 6)
Bk : bucket
I_n : level n of index
Method:

```
1: begin
2:   do while
3:     if I_n[S_n] has not I_n+1
4:       search Tw in I_n[S_i]->BK
5:       break
6:     else if I_n[S_n] has I_n+1
7:       n <- n + 1
8:       go to next level of index
9:     end if
10:  end while
11: end
```

IV. PERFORMANCE STUDY

A. Performance of dynamic hashing

The total search time for finding target word is the sum of the traverse time of index tree and the search time in data bucket in dynamic hashing. The follows equation shows the total search time in dynamic hashing.

$$Ts_t = Is_t + Bs_t \quad (2)$$

Ts_t is the total search time, Is_t is the search time of index and Bs_t is the search time in bucket.

We can use the binary search if the source words are sorted in bucket, but if these are not, we have to use the linear search to find target word in bucket. The number of source word in bucket are small, there is little difference in performance in previous two cases. Therefore, In order to reduce the total search time in dynamic hashing, to improving the performance of the index is very important above all.

We have two perspectives about performance of index. The one is how can improve the performance of searching index another one is how can store whole data using smaller number of buckets. These two perspectives are not independent but are closely related to each other. If the amount of data stored in the bucket is greater than the bucket size, the overflow will occurs. And then the overflow bucket is split to two buckets and index level grows up for pointing new bucket. So, if data can be stored in the distributed buckets properly, we can prevent the increase of the index level.

Therefore, in this paper, we propose new hangeul dynamic hashing algorithm using variance information by location to determine the order of the index. The proposed algorithm has smaller index level than the conventional method and also has smaller number of buckets.

B. Evaluations

In this section we perform an evaluation of proposed method in this paper.

The performance of dynamic hashing can be measured two kinds of evaluations. The one is the average level of index and the other is the number of buckets needed of storing the data.

The data used in this experiment were extracted from navigation system's name data.

Fig. 8 shows the average level of index to store name data in accordance with the number of data is 10,000, 20,000, 30,000 and 40,000. In Fig. 8, Simple method is the method described in II-A. In this method, the index is made in accordance with the simple order of character like C_{11}->C_{12}->C_{13}⋯ .

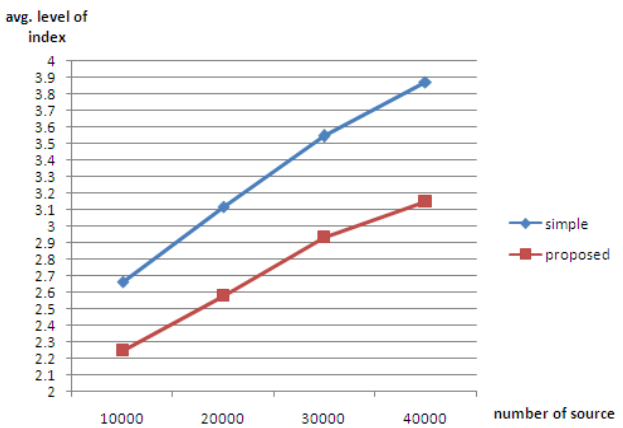

Fig. 8. The average level of index in accordance with the number of data.

As shown in the Fig. 8, the needed index level is 2.25 to store 10,000 name data in the proposed method but, it is 2.66 in the ordinary method. In case of 40,000 name data, the proposed method needs 3.15 index level, on the other hand the ordinary method needs 3.87. So we can find that the average level of needed index using the proposed method is less than that using the ordinary method. If the number of stored data is bigger than the difference of average index level between two methods is bigger. When you consider that more than 100,000 name data are used in navigation system generally, we can know the difference of them is bigger.

Fig. 9 shows the number of buckets is needed of storing name data in two methods.

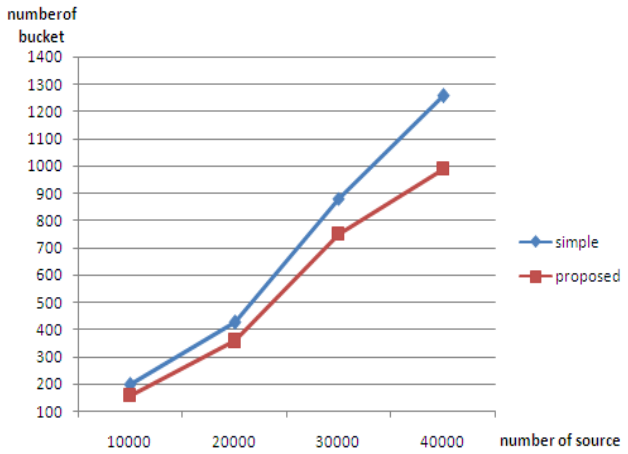

Fig. 9. the comparison of needed buckets.

As shown in the Fig. 9, we can find that needed buckets are smaller the proposed method than the simple method. Also, if the number of stored data is bigger than the difference of number of needed buckets between two methods is bigger. So, the proposed method is more efficient than the ordinary method in conclusion.

Fig. 10 shows the size of index is needed. As shown in Fig.10, the size of index is smaller the proposed method than the simple method, also.

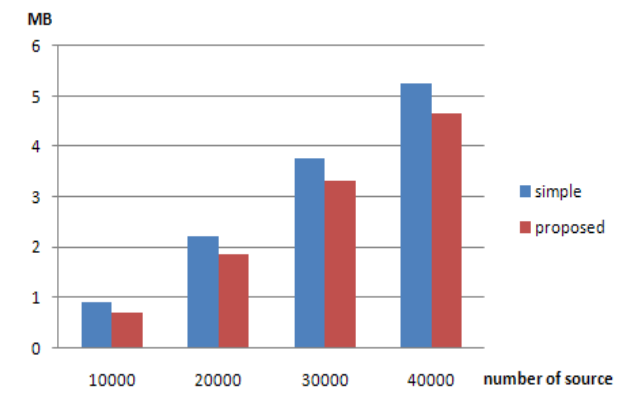

Fig. 10. the comparison of index size.

V. CONCLUSION

In this paper, we proposed the dynamic hashing algorithm using variance of character. And, we could find that our algorithm is more efficient than ordinary algorithm through experiments. The proposed method can use efficiently in the system has limited resource like navigation systems or mobile systems because the size of index is smaller than ordinary index mechanism, and also the number of needed buckets are fewer than simple dynamic hashing algorithm.

The proposed algorithm also can use for other retrieval systems using person name or address, etc. So, in the future, we plan to do experiment other retrieval system are applied our dynamic hashing algorithm.

ACKNOWLEDGMENT

This work was supported by Dong-eui University Foundation Grant (2009).

REFERENCES

[1] Taijin Yoon, Hwan-Gue Cho and Wookeun Chung, "A Phoneme-based Approximate String Searching System for Restricted Korean Character Input Environments", Journal of KISS, vol. 37, No, 10, pp.788-801, 2010.
[2] Gollapudi, S. and Panigrahy, R. ,"A Dictionary for Approximate String Search and Longest Prefix Search", CIKM International Conference, Vol.15, pp.768-775, 2006.
[3] Paolo Ferragina and Roberto Grossi. "The String B-Tree: a new data structure for string search in external memory and its applications," Journal of the ACM, vol. 46(2), pp.236-280, 1999
[4] Kyeonghwan Kim, "High-Speed Korean Address Searching System for Efficient Delivery Point Code Generation," The KIPS Transaction, Vol.8, No.3, pp.273-284, 2001
[5] Hiecheol Kim, Jeonghun Shin, Yong-Doo Lee, "Construction of Korean Dictionary Using Multi-dimensional Binary Tree", Conf. of KISS, Vol. 25, No.1(B), pp. 452 ~ 454, 1998.
[6] Paolo Ferragina and Roberto Grossi. "The String B-Tree: a new data structure for string search in external memory and its applications", Journal of the ACM, vol. 46(2), pp.236-280, 1999.
[7] Cheol-Su Kim, Woojeong Bae, Yong-seok Lee, Jun-ichi Aoe, "Construction of Korean Electronic Dictionary Using Double-array Trei Structure", Journal of KISS(B), Vol. 23 ,No. 1, pp. 85-94, 1996.
[8] Ronnblom, J., "High-error approximate dictionary search using estimate hash comparisons", Software:Practice and Experience, Vol.37 No.10, pp.1047-1059, 2007.
[9] Martine Dietzfelbinger, Marine Huhne, "A dictionary implementation based on dynamic perfect hashing", Journal Journal of Experimental Algorithmics (JEA), Vol.12, June, 2008.
[10] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan, " Dynamic Perfect Hashing: Upper and Lower Bounds", SIAM J. Comput.. Vol. 23, pp. 738-761, 2007
[11] ShinWoo Kim, YongKyu Lee, " An implementation and evaluation of large-scale dynamic hashing directories", Journal of Korea Multimedia Society, Vol.8, No.7, pp.924-942, 2005
[12] Jung-hwa Lee, "Character based Hangeul search using Location specific Character Frequency", International Journal of KIMICS, Vol. 7, No. 3, pp.345-350, 2009+

Jung-Hwa Lee received his B.S. and the Ph.D. degrees at the Department of Computer Science from Pusan National University, Korea, in 1995 and 2001, respectively. He is a professor at the Department of Computer Software engineering, Dongeui University in Korea. His research interests include database, Hangeul information processing, and semantic web, etc