김일성종합대학학보
(자연과학)
주체103(2014)년 제60권 제6호

JOURNAL OF KIM IL SUNG UNIVERSITY
(NATURAL SCIENCE)
Vol. 60 No. 6 JUCHE103(2014).

Study on the BURP-Trie Structure using Bitwise Keyword-Index

Kim Tae Myong, Jo Un Chol

Trie structure, the fast retrieval algorithm, which is widely used as for static data structure with no insertion, deletion and update of data, has following advantages: [1－4]

① Time complexity of keyword retrieval is $O(1)$, regardless of the size of keyword set.

② Keyword insertion time is proportional to the size of keyword set.

③ Keyword insertion time is proportional to the length of the keyword and time complexity is $O(1)$.

Double array compression TailTrie, most popular in data structure representing Trie, which has double array structure and Tail array, has following disadvantages:

① Keyword set constructable in Trie is limited.

② Insertion calculation complexity is increased burstly with the size of keyword set.

③ Range search is not so efficient as B+ tree

④ When whole Trie tree is not possible to load onto RAM , disk I/O cost increases for no paging.

On the other hand B+ tree, widely used in database management system, is balanced tree. Searching path is derived using internal nodes and terminal nodes hage data input point. Main properties of B+ tree are as follows.

① Balance of tree is kept in the process of operation (insertion or deletion)

② Each node, except root, is filled at least 50%. But in general file is expanded rather than decreased, so in the process of deletion, adjustment of tree is not necessary. Simple search and deletion is enough.

③ In search of record, path is from root to appropriate terminal.

④ Though there is spatial load for storing index input point in B+ tree, files are aligned and effective insertion and deletion algorithm is possible.

⑤ Equivalent search speed is significantly low compared with Hash tree or Trie.

In this paper, analyzing the advantages and disadvantages of Trie and B+ tree, we proposed an improved TRIE index structure. This structure can use all the possible data type as keyword and support effective keyword compression in order to increase the speed for index construction and search and to effectively manage the large scale databases.

1. Definition of *BURP-Trie* Index Structure

Definition 1 *Slot*: Minimal unit that contains child transition information with internal representative value of keyword is called Slot.

$$Slot = <page\_num,\ node\_num>$$

Table 1 shows the ingredients and description of slots.

Table 1. ingredients, description and size of slot

No.	Element	Description	Size
1	*page_num*	Page number of child node	4B
2	*node_num*	Node number inside the corresponding page (offset in the page)	2B

Definition 2 *Root*: The node from which all the keyword searches start is called Root and written as *R*.

Root is composed of 2 slots.

$$R=<slot(0),\ slot(1)>$$

where $slot[i](i=0,\ 1)$ contains a point to the child node. So with this information, search process may reach to the slot for all the keyword search which has the bit corresponding to the the calculated internal representative value as first bit. In a tree, root is only one.

Definition 3 Internal node: All the nodes used for keyword search, except root, are called internal node, and representeds as *M*. An internal node is composed of internal node header and 3 slots.

$$M=<Hm,\ slot(\#),\ slot(0),\ slot(1)>$$

where Hm=<Parent_slot, Prunded_flag, Prunded_len, Prunded_String>

Table 2 shows the description of each field.

slot(#) is slot unique to internal node and contains transition information for searching keyword ended with bit stream to the corresponding node.

slot(#) has same structure of *slot*(*i*).

Table 2. structure of internal node header

No.	Ingredients	Description	Size	
1	*Parent_slot*	Indicates parent node. In the process of keyword deletion, information for reconstructing tree is included.	6B	
2	*Prunded_flag*	Flag. Indicates whether keyword compression between parent and child node is done.	1bit	1B
3	*Prunded_len*	Compressed keyword length	7bit	
4	*Prunded_String*	Compressed keyword bit string	64*bit*(8B)	

Definition 4 *Leaf node*: Node, containing actual record of <keyword, data> is called leaf, and represented as *L*.

$$L=<K,\ D>$$

where *K* is keyword and *D* is data corresponding to the key *K*.

The length of pair <keyword, data> may be different and variable, so number of records in a node is not fixed. And if the length of <keyword, data> is bigger than the capacity fo a node, new leaf is not added, but auxiliary nodes for storing remained part of <keyword, data> are used, which is called *overflow node*, L_o.

Overflow nodes are concatenated with a leaf node containing former part of the <keyword, data> in sequence. Fig. 1 shows each part of <keyword, data> of a leaf node.

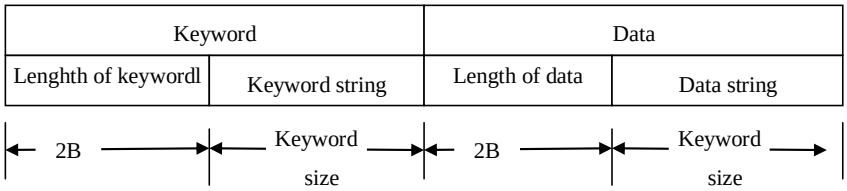

Fig. 1. Structure of <keyword, data> pair.

Considering <keyword, data> pairs in a leaf node, the pairs are aligned with keyword.

And leaf nodes are linked with double linking list, so order of keyword sizes should be maintained among leaf nodes.

Then, as in the case of B+ tree, when considering the order of linking list in the leaf pages, all the records may be stored aligning with the size of the keyword, which supports range search effectively.

Based on the above definitions, *BURP-Trie* is defined as follows.

Definition 5 *BURP-Trie*: When R is root, M definitive set of internal nodes, L definitive set of leaf nodes (overflow node is included), g node transition rule, then suit of (R, M, L, g) is called *BURP-Trie* and is represented as T_P.

$$T_P = (R,\ M,\ L,\ g)$$

Fig. 2 shows a conceptual structure of *BURP-Trie*.

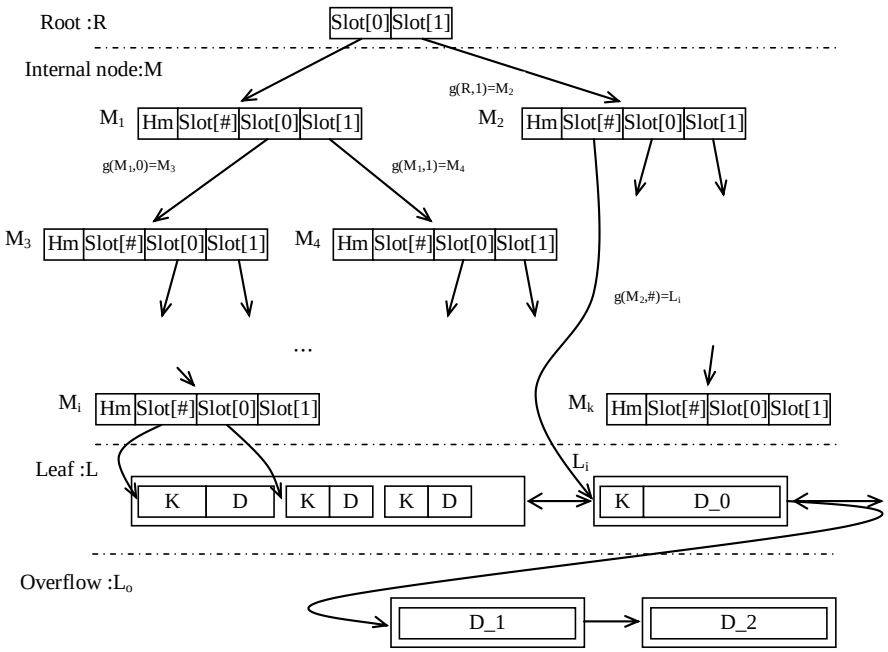

Fig. 2. Conceptual diagram of BURP-Trie

And node transition rule g is defined as follows.

Definition 6 Node transition rule g: If searching keyword is $K=k_1k_2...k_n$ (where k_i is i^{th} bit of keyword), internal representative value is $v_i=val(k_i)$, then status transition function

$$g: \{ v_i \mid i \in \{\#, 0, 1\}\} \times \{R, M\} \Rightarrow \{M, L\}$$

is called node transition rule.

If u_j and u_k are root or internal node and there exist u_j and u_k that satisfies $g(v_i, u_j) = u_k$, then these two nodes are in the relation of parent and child and u_j is parent and u_k is child. g gives page number and node number by searching slot in the node with internal representative value of each bit of keyword K.

2. Characteristics of *BURP-Trie* Index Structure

From the definition of *BURP-Trie*, its characteristics are as follows.

① Having the characteristics of *Trie* index structure, *BURP-Trie* has the same searching speed as *Trie* and superior indexing performance. That is, in searching process no comparison operation is included but only substitution operation, so searching time is same as *Trie*.

② Remaining the space for keyword, in keyword insertion process no tree adjustment is performed in root or internal nodes. But in case of keyword compression, in insertion process of keyword which has the slot as end slot or has same internal representative value upto the node, new node should be assigned. Therefore keyword insertion speed is faster than *TailTrie*.

③ Using root, internal node, leaf, and overflow nodes and paging, advantages of B-tree are maintained.

Capacity of a node is fixed, so static node displacement in page is possible, which requires loading of necessary pages in searching process, so the number of disk I/O may be decreased to the level of B-tree

④ Keyword compression is used, so according to the appearance frequency of bytes composing keyword, much less disk I/O may be required.

⑤ Leaf and overflow nodes are same as B-tree, so disadvantage of Trie which requires returning back along the tree in region search is removed and the same performance of B-tree can be supplied

Conclusion

In this paper new BURP-Trie indexing structure is proposed, which have advantages of both B-tree and Trie and overcome shortcomings.

Proposed indexing structure outperforms B-tree and Trie in index construction and search.

References

[1] M. E. Nebel; A Combinatorical Study"Theoretical Computer Science, 270, 441, 2002.

[2] H. Shang; Trie Methods for Text and Spatial Data on Secondary Storage, Mc-Gill University, 70～95, 2001.

[3] D. E. Zegour; Elsevier Information and Software Technology, 46, 923, 2004.

[4] Minsoo Jung et al.; Information Processing and Management of Elsevier, 38, 221, 2002.

[5] J. Bourdon; Theoretical Informatics and Applications, 35, 163, 2002.