TRIE TREE ALGORITHM WITH MODIFIED KEY SAMPLING FUNCTION

Taghrid Abdul majeed Rashid
Department of Computer, College of Education Ibn Al Haitham, University of Baghdad.
Baghdad- Iraq.

Abstract

The ongoing changes in computer performance are affecting the efficiency of string searching algorithms. The size of main memory in typical computers continues to grow, but memory accesses require increasing numbers of instruction cycles.
One of the most important data structure which is used to improve the string searching is the Trie tree.
The name Trie comes from the word "retrieval". This data structure dose not store the data as specific elements, but rather as a path through the tree.
This research aims to improve the performance of the Trie tree by using a proposed sample function in trie algorithm which improve space utilization and decrease the number of the search levels in the trie tree. i.e. decrease the time of searching because it searches less number of levels than the standard sample function.

Trie tree

. - .

(strings)
(retrieval) (Trie tree)
.(paths)
(Trie tree)
.

Introduction

With the ever-increasing influence of the Word Wide Web (WWW) and the Internet, the importance of data structure and algorithm, engineering principles has significantly increased. Internet algorithmic focuses on topics of algorithm and data structure design and engineering for combinatorial problems whose primary motivation comes from Internet applications. [1]

One of a very important data structure is a **tree.** **A general tree (T)** is a finite set of one or more nodes such that there **is** one designated node R, called the root of T. The remaining nodes in

(T −{R}) are partitioned into n>=0 disjoint subsets T1, T2, …, Tn each of which are children of R.

Trie trees, which come from the word **"retrieval"**, data structure is used to store an associative information. Unlike other data structures, the data are not stored as specific elements, but rather as a path through the tree. This allows optimal searching even for very large databases, although it can be memory-inefficient at times.

A trie tree is commonly used in Dictionaries, Phone directories, Bioinformatics applications, Searching algorithms, Cache-efficient string sort and used to represent the layout of a website consisting of a root index page followed by a hierarchy of subpages. [2]

This paper proposes a modified trie tree algorithm mainly consists of two phases. The first phase concerns with building the trie tree using a proposed key sampling function. The second phase of the algorithm on the other hand, concerns with how to retrieve information from the constructed trie tree.

Performance of the trie tree with the proposed key sampling function is evaluated using different sets of words and compared with the standard trie tree algorithm.

Trie tree

A trie tree is a tree of degree m>=2 (where m is the maximum number of branches in any node of the tree) in which the branching at any level determined not by the entire key but by only a portion of it. The trie contains two types of nodes:

The first type represents a leaf node which contains the string. The second is a branch node which contains 27 link fields (depends on number of alphabet characters).

All characters in the key values are assumed to be one of the 26 letters of the alphabet and the blank.

The standard Algorithm used to build the trie tree is: [3]

- At level 1 all key values are partitioned into 27 disjoint classes depending on their first character, on the j-th level the branching is determined by the j-th character.
- When a subtrie contains only one key value, it is replaced by a node of type string.
- The number of levels in the trie depends on the strategy or key sampling function used to determine the branching at each level.

The standard sampling function:

(sample (x,i)) used to build the trie ,which appropriately sample x, for branching at the ith level in the tries tree is :

SSample (x, i) = i^{th} character of x --- (1)

Given the following set of key values to be represented in an index:

Bluebird, bunting, cardinal, chickadee, godwit, goshawk, gull, oriole, thrasher, thrush and wren.

Figure 1 shows the created trie tree using characters of key value from left to right, one at a time. In Figure 1, branch nodes are represented by rectangles while ovals are used for information nodes.

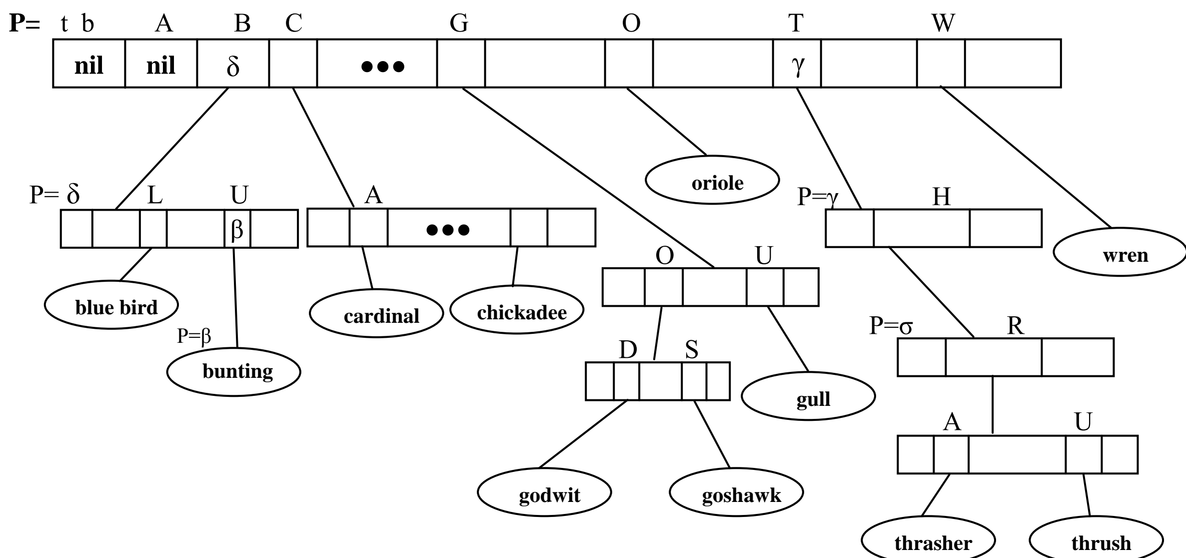

Figure 1: A Trie tree using standard key sampling function

When inserting any new key to the tree, a search operation must be perform before the insertion to find the suitable location for a new word.

Searching a trie for a key value x requires breaking up x into its constituent characters and following the branching patterns determined by these characters.

The algorithm **trie,** [p is a pointer variable used to search for any key] assumes that p=nil is not a branch node and that p^. Key is the key value represented in p if p is an information node.

```
function trie (var t,p: trieptr; x : string) : trieptr;
 {search a trie  for key value x, it is assumed that
   branching on the i^th level is determined by the
   i^th character of the key value. where the root is
   given by pointer variable t}
 var c : char ; I , k :integer;
begin
{assume we can always concatenate at least 1
   trailing blank to x }
     k :=x; concatenate (k,' ');
     i:=1;  p:=t;
     while p is a branch node do
     begin
        c:= i-th character of k;
        p:= p^.link[c];
        i :=  i+ 1;
     end;
     if p = nil or p^.key < > x
        then trie := nil
        else   trie := p
end ;  { of trie }
```

For example search for word (bunting) using function **trie** and trie tree in figure 1, the result is found in location given by pointer variable p, as shown in Figure 2.

But if we search any word not exist in the trie tree in Figure 1 for example word (then).

The value of p is nil which means not found. As shown in Figure 3.

Note that the main advantages of trie tree are

- The time to find a match is related to the length of the match not the number of items in the data structure.
- A significant improvement in speed can be get by using trie tree.
- There is a property that the trie structure is independent of the order in which the keys are inserted, there is a unique trie for any given set of distinct keys.

The main disadvantages of the trie tree are

- The mount of wasted space for unused links.
- They can be applied to strings of elements or elements with an efficiently reversible mapping (injection) to strings.
- They lack the full generality of balanced search trees, which apply to any data type with a total ordering.

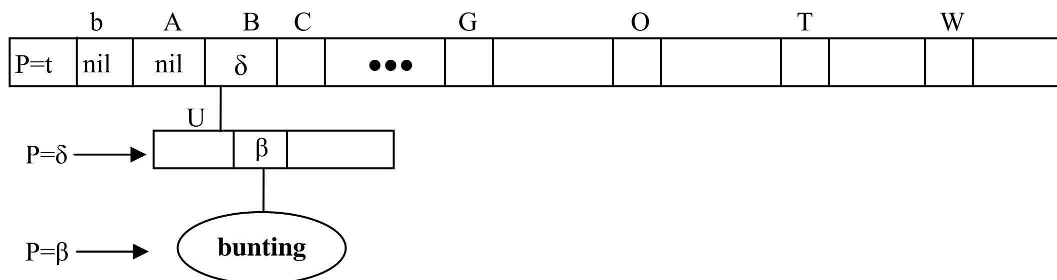

Figure 2 Search for word bunting

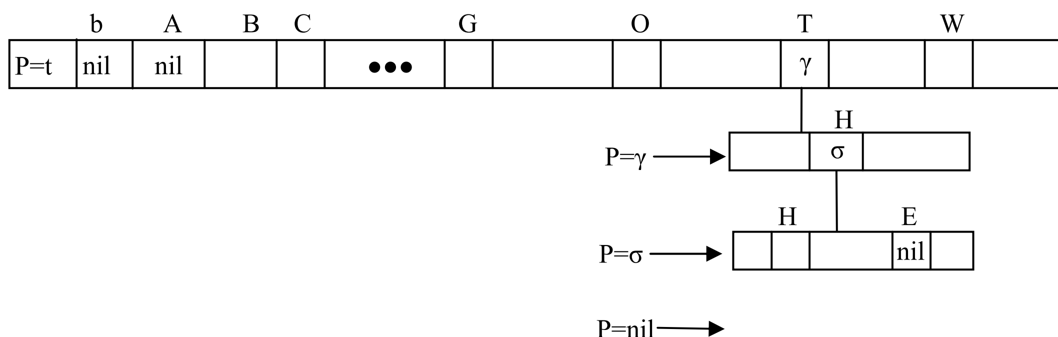

Figure 3: Search for word then

Proposed algorithm

An index structure that is particularly useful when key values are of varying size is the trie. The height of a trie is the length of the longest key in the trie which can be memory-inefficient at times because of the large number of levels.

Using different sample functions one may construct key value sets for which that particular function is best, i.e. it results in a trie with the fewest number of levels.

In this research a new proposed key sampling function is defined to reduce the number of levels in the trie tree which will reduce the overall required memory space to store the tree.

The proposed key sampling function is:

$$\text{PSample}(x, i) = \begin{cases} X_{n-(i-2)/2} & \text{if i is even} \\ X_{i/2+1} & \text{if i is odd} \end{cases} \quad --- (2)$$

Where **i** is the ith character in the string
n is the last character in the string

The trie tree [4], in the proposed algorithm contains two types of nodes:

First type is the external node: represent a **leaf node** which contains two fields:

1. leaf :(string) key
2. is-leaf : (Boolean) flag

Second type is internal node: represent a **branch node** also contains two fields**:**

1. branch: contains 27 link fields to other nodes.
2. is-leaf : Boolean

The three suggested operations performed on the trie are:

1. Initialization: This step concerns with trie tree creation and initialization values:
Initialize a node object which corresponds to the root of the trie by:
1. Filling all 27 link fields with nil value.
2. Filling is-leaf field with false value.

2. Insertion:
1. Find the place at which the string is to be inserted by starting the search from the root node and continue the searching through the characters of the string.
2. If there is no item , just insert the string as a leaf node, as in other types of tree ,and set is-leaf field with true value.
3. If there is something on the leaf node (if the value of is-leaf field is true), then this leaf node becomes a new inner node and build a new subtree (or subtrees) to that inner node depending on the string to be inserted and the string that was in the leaf node. The branch in the tree depending on the characters of the string at level 1, character in position 1 is used, at level 2 character in position N is used, at level 3 character in position 2 is used, at level 4 character in position N-1 is used , and so on until an external node is encountered.
4. Create a new leaf node where the new string is stored.
Figure 4 shows a created trie tree using the proposed sample function for the same keys used in Figure 1.
(Bluebird, bunting, cardinal, chickadee, godwit, goshawk, gull, oriole, thrasher, thrush and wren).

3. Searching:
1. Start from the root node and from the most significant character in the string.
2. Branch in the tree in the same manner used in insertion operation until the leaf node is found (i.e. until the value of is-leaf field is true)
3. Check if the string is in the leaf node or not.

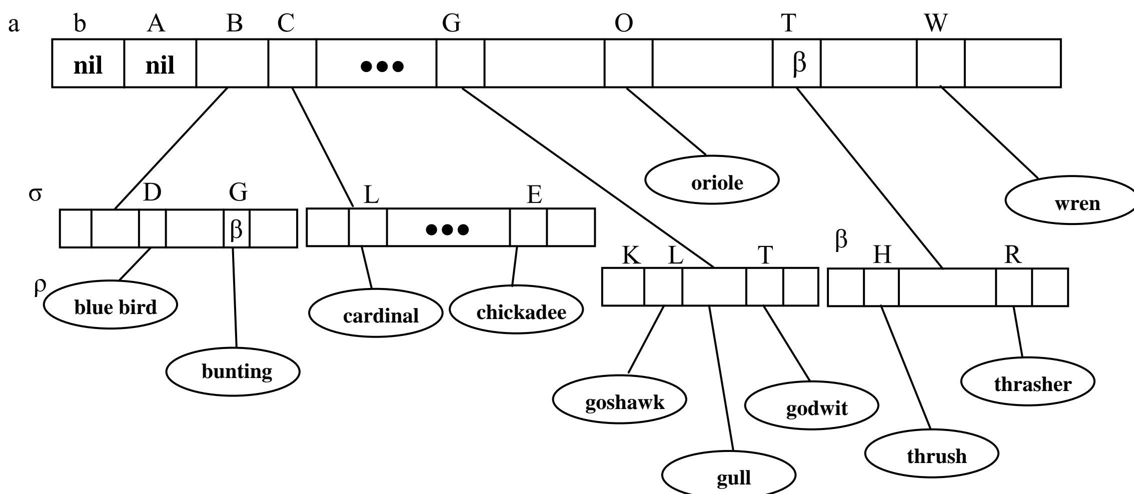

Figure 4: A Trie tree using the proposed key sampling function

Test and results

The suggested algorithm is tested together with the standard algorithm on a different sets of words.
The following tables 1, 2, 3, and 4 show the number of levels for each given word using the standard sample function (SSample) and the proposed sampling function (PSample).

Table1

	Key	No of levels using SSample	No of levels using PSample
1	bluebird	2	2
2	bunting	2	2
3	Cardinal	2	2
4	Chickadee	2	2
5	Godwit	3	2
6	Goshawk	3	2
7	Gull	2	2
8	Oriole	1	1
9	Thrasher	4	2
10	Thrush	4	2
11	Wren	1	1
12	And	2	2
13	apple	5	2
14	Apply	5	2
15	badly	4	2
16	Bat	3	2
17	Some	5	2
18	Abacus	3	2
19	Something	5	2
20	abracadabra	3	3
21	This	3	2
22	Somerest	5	2

Table2

	Key	No of levels using SSample	No of levels using PSample
1	bluebird	2	1
2	bunting	2	1
3	Cardinal	2	2
4	Chickadee	2	2
5	Godwit	3	1
6	Goshawk	3	1
7	Gull	2	2
8	Oriole	1	2
9	Thrasher	4	1
10	Thrush	4	1
11	Wren	1	1

Table3

	Key	No of levels using SSample	No of levels using PSample
1	bluebird	2	2
2	bunting	2	2
3	Cardinal	2	2
4	Chickadee	2	2
5	Godwit	3	2
6	Goshawk	3	2
7	Gull	2	2
8	Oriole	1	1
9	Thrasher	4	2
10	Thrush	4	2
11	Wern	1	1
12	Joe	3	3
13	John	5	2
14	Johnny	5	2
15	Jane	3	3
16	Jack	3	2
17	adams bt	1	1
18	Cooper cc	8	4
19	Cooper pj	8	2
20	Cowans dc	3	4
21	Maguire wh	2	2
22	Mcguire dd	2	2
23	Spanner dw	5	2
24	Span kd	6	3
25	Sefton sd	2	2
26	Span la	6	2
27	Zarda jm	7	2
28	Zarda pw	7	2

Table 4

	Key	No of levels using SSample	No of levels using PSample
1	bluebird	2	3
2	Bunting	2	2
3	Cardinal	2	2
4	Chickadee	2	2
5	Godwit	3	2
6	Goshawk	3	2
7	Gull	2	2
8	Oriole	1	1
9	Thrasher	4	2
10	Thrush	4	2
11	Wren	1	1
12	Some	5	2
13	Something	5	2
14	Abacus	3	2
15	Some rest	5	2
16	This	3	2
17	adams bt	2	2
18	Cooper cc	8	4
19	Cooper pj	8	2
20	Cowans dc	3	4
21	Maguire wh	2	2
22	Meminger dd	2	2
23	Spanner dw	5	2
24	Span kd	6	3
25	Sefton sd	2	3
26	Span la	6	2
27	Zarda jm	7	2
28	Zarda pw	7	2
29	Mcguire al	2	2
30	Abracadabra	3	3
31	And	2	2
32	Apple	5	2
33	Apply	5	2
34	Bad	4	3
35	Badly	4	2
36	Bat	3	2

Conclusion

The **results** in table 5 give a clear evidence that the performance(in term of efficiency) of the trie tree with the proposed key sampling function is better than that of the standard trie tree. With respect to the total number of levels in each trie tree, the proposed key sampling function gains reduction in both computation time (search time for the required word) and memory space . Shortly speaking, as the total number of levels is reduced (via proposed key sampling function), we get more efficient results in term of both time and space.

Table 5: shows the comparison between the results get from the four tables

Table no.	No. of words	Comparison between no. of levels using SSample and PSample					
		Less than	equal	greater than	Less than	equal	greater than
Table 1	22	0	9	13	0%	40.9%	59.1%
Table 2	11	1	4	6	9.09%	36.36%	54.54%
Table 3	28	1	13	14	3.57%	46.43%	50%
Table 4	36	3	11	22	8.33%	30.56%	61.11%

References

1. Goodrich, M.T. and Tamassia, R. **2001.** *Algorithm Engineering*. John Wiley and Sons, New York, pp.129-133.
2. Heinz, S. and Zobel, J. **2002** *Practical data structures for managing small sets of strings.* In M. Oudshoorn, editor Proc. Australasian computer science Conf., pages 75-84, Melbourne, Australia.
3. Ellis Horowitz. Sartaj Sahni. **1994,** *Fundamentals of data structures in Pascal.* Copyright computer science press, Inc, pp.512-520.
4. Heinz, S.; Zobel, J. and Williams, H. E.. **2002.** Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information systems*, **20**(2):192-223