DMP-tree: A dynamic M-way prefix tree data structure for strings matching

# Nasser Yazdani; Hossein Mohammadi

## 2010

The data structure is very efficient for time critical applications such as IP lookup where the longest prefix matching a query string must be found in a short time

# Abstract

We propose DMP-tree, a dynamic M-way prefix tree, data structure for the string matching problem in general and prefix matching in particular. DMP-tree has been initially devised for fast and efficiently handling prefix matching which constitutes the building block of some applications in the computer realm and related area. It is assumed there are strings of an alphabet Σ which are ordered. The data strings can have different lengths and some of them can be prefixes of others. Two well known applications of prefix matching are layers 3 and 4 switching in TCP/IP protocols. In layer 3 switching, routers forward an IP packet by checking its destination address and finding the longest matching prefix from a database. In layer 4 switching, the source and destination addresses are used to classify packets for differentiated service and Quality of Services (QoS). DMP-tree is a superset of B-tree. When none of the data strings are a prefix of each other, DMP-tree is the same as B-tree. In DMP-tree, no data string can be in a higher level than another data string which is its prefix. This requires a special procedure for node splitting. Indeed, node splitting differentiates DMP-tree from B-tree. The proposed data structure is simple, well defined, easy to implement in hardware or software and efficient in terms of memory usages and search time compared to other data structures proposed for prefix matching. We have implemented DMP-tree and the experimental results for simulated IP prefixes from the {0, 1} character set show an average search time of LogMN for a large number of N, number of data elements, when the internal node branching factor M is big enough (⩾5).

# Study subjects

**6 independent data sets**

To test the search performance and memory utilization with the IP lookup data, which is the hottest application of our method, some simulated data was produced and a few small modules for bit handling were included in the code. **Six independent data sets 50K, $60K,. . .,100K were produced. were produced**. There were no duplications in the data sets

# Findings

The last approach was to modify functional units of a RISC processor to accelerate DMP-tree based lookup software [28,29]. It led to almost 30% improvement in lookup speed and 14% in code size

# Scholarcy Highlights

* A routing table database consisting of pairs of IP prefixes, or network addresses, and their corresponding hop addresses is searched to find the longest IP prefix matching the packet destination address
* Ref. [30] presents a hardware architecture for an IP Lookup engine based on DMP-tree, It can perform each lookup in just one clock cycle
* We propose DMP-tree, Dynamic M-way Prefix tree, initially for the prefix matching
* Our experimental result with some simulated binary data for the IP lookup application shows when the data set is large, over 50K, and the branching factor in the internal node is big (?5), DMP-tree acts like B-tree in terms of the search performance and memory utilization
* The data structure is very efficient for time critical applications such as IP lookup where the longest prefix matching a query string must be found in a short time

# Scholarcy Summary

## Introduction

And efficiently locating prefixes matching a query string is crucial in some applications.

A router connected to Please cite this article in press as: Yazdani N, Mohammadi H, DMP-tree: A dynamic M-way prefix tree data structure for strings matching, Comput Electr Eng (2008), doi:10.1016/j.compeleceng.2008.04.011 a 10 giga data line with a routing table of 164K IP prefixes has to find the hop address in 50 ns.

This problem is referred as layer 3 switching in the network community.

## Background and basic issues

Tree structures keep the data elements sorted. in order to apply any tree structure to a data set, it is essential to have a mechanism to sort them.

1. If n = m, the two strings have the same length and the values of A and B are compared to each other based on the order of characters in R.

In the English alphabet, assuming the probability of a character to be in the range A –M or N–Z in a text to be roughly 50%, M can be considered as ?.

We have proposed a binary prefix tree for prefix matching in [25,31] which utilizes Definition 1 and the idea of enclosure

It takes a prefix and all data elements enclosed in its space as a data point represented by the prefix.

To build the binary prefix tree, the data elements are sorted and enclosurized first.

Refs. [25,31] proposed two approaches to sort and enclosurize data elements based on bubble and quick sort

## DMP-tree

The worst case search time for the binary prefix tree proposed in [25,31] is O(W) where W is the maximum length of the data elements.

1. No data element can be in a higher level than its enclosure in the index tree structure.

In the binary prefix tree, enclosures are always in the higher levels than their corresponding enclosed elements.

Our experimental results show these properties fade when the branching factor in the internal nodes is large and the data set is big

In this case DMP-tree approaches B-tree in terms of search and memory utilization.

We formally define node splitting, insertion, space division, merging and search mechanisms

Other procedures such as building the tree or deletion are the same as B-tree [4]

## Node splitting

The main problem in node splitting is determining the spilt point. In B, the median element is chosen to split the space.

2. If there is an enclosure which encloses all other data strings in the node, it is selected for splitting the tree node.

In the cases not mentioned above, the data elements can be a combination of disjoint spaces and strings

In this case, an enclosure or a disjoint element which gives the best splitting result is chosen as the split point.

The algorithm assumes the previous data elements in the node are sorted and the place of the new inserted data string has been already found

It keeps a bit for each element which shows whether it is enclosed in a data space or not.

This will not create any problems since that element will be in the data space of the prefix which is enclosure of the element under check.

The worst case is O(N2) when data elements are disjoint

## Space division

Space division is encountered when a data string is replaced by its enclosure in an internal node.

According to the tree definition, all strings in the left subtree are smaller than the root and the elements in the right subtree are bigger.

The new elements which violate sort condition in the tree must be moved to the other side of the element in the root of the subtree

This is how the merge operation is encountered.

As B-tree, Insertion is the most important part of the building process of DMP-tree

It uses Split, SpaceDiv, Merge and NewNode functions to split a node if it is full, divides a space if an element is replaced by its enclosure, and merges two subtrees if necessary.

Inserting an enclosure when it causes space division takes more time since the SpaceDiv procedure has to be called to divide the space according to the new root element.

## Query processing

For the Longest Prefix Matching (LPM) of a query string, [25,31] proposes a method which exhaustively compares all elements in the nodes of the search path.

This is due to the fact that an enclosure and its contained elements may be in the same level.

It is necessary and sufficient to compare Q up to the first data element in the node which is disjoint with P in order to find the Longest Prefix Matching (LPM) of Q.

{/\* tree is a pointer to the root of the index tree and str is the query string.\*/ LPMSearch if tree = NIL,return NULL; i = first element in tree(node).

The All-String-Search procedure correctly finds all strings having a query string as their prefix

## General pattern matching

Ref. [13] tries to solve the problem of finding all strings having prefix P.

The B-tree like index structure keeps pointers to the location of the data strings.

For each node, we have to have M À 1 disk accesses to get the actual value of strings to compare

This makes the search process very costly.

To keep the first few characters of data elements in the tree nodes like [5].

This takes more space, but, It does not need other disk accesses for matching most of the time.

Assume a text h with length N, a DMP-tree built for the suffix set of h has the maximum height dMNÀ1c where M is the maximum branching factor in the internal nodes.

Since there are N elements in the suffix set, and each node, including leaves, can contain M À 1 data element, the maximum height of tree will be dMNÀ1c. h

## Binary alphabet results

DMP-tree is a superset of B-tree and when all data elements are disjoint, it acts exactly the same as B-tree.

We expect the average search and update time to be proportional to LogMN , where N is the number of data elements and M is the branching factor in the internal node.

In some applications such as dictionary checking and IP lookup, when the data set is large and data elements are expected to be random and branching factor in internal nodes is large enough, the DMP-tree memory utilization and search performance approach B-tree.

According to Lemma 3 the initial height of the tree before inserting other prefixes will be dMLÀ1c assuming the number of data elements in leaves and internal node are the same.

Assuming the data set has all the worst case data and they are inserted first, the initial DMP-tree will have the height d3M2ÀÀ18c.

On a heavily loaded SUN SPARC 10 it never took more 16 s to build the index tree

## Results with general Alphabet

We implemented DMP-tree with general alphabet set. The alphabet constituted of 256 possible characters in the ASCII system.

To get these results we have made DMP-tree for three datasets of 50K, 100K and 130K prefixes.

DMP-tree based IP lookup can be a very good, scalable, cheap and fast solution in hardware and it can be treated as a better alternative for TCAM-based methods

Another approach for fast and scalable IP Lookup using DMP-tree, is HASIL [27], HASIL modifies memory unit of a general purpose processor to support two additional instructions.

It led to almost 30% improvement in lookup speed and 14% in code size

## Related work

Ref. [3] is a rich source for pattern and string matching problems and related algorithms.

The general prefix string matching problem in which one is interested in finding the longest prefix of a pattern starting at each position of a text string is addressed in the pattern matching literature

This problem is a general case of the problems we address here and can be solved in linear time by adopting the string matching algorithm of Knuth, Morris and Pratt [15].

The method is mostly concerned with retrieving strings from the secondary memory

It starts with prefix matching and transfers it into the range query.

The main scheme for the prefix matching which is the base of some other methods and has been intensively discussed in the literature is Trie [16].

The binary Trie [20] or the radix-tree [9] is the base of many methods for the IP lookup problem.

An interested reader can refer to [22,23,19,20,10,11,17] for more information

## Conclusion

We propose DMP-tree, Dynamic M-way Prefix tree, initially for the prefix matching. The prefix matching constitutes the base of some applications such as DNA matching, spell checking, online dictionaries, telephone directories and layers 3 and 4.

The well known IP lookup problem in TCP/IP packet forwarding has motivated us to propose the DMP-tree data structures

It has broader applications and may used in problems in which a set of strings from an alphabet R can be prefixes of others as well as in general pattern matching.

Proposed methods have used traditional sorting schemes for strings which are based on the character ordering in the alphabet and lengths of data strings

Those schemes cannot handle prefix matching directly and have to transfer them into range query which makes them very costly.

Our experimental result with some simulated binary data for the IP lookup application shows when the data set is large, over 50K, and the branching factor in the internal node is big (?5), DMP-tree acts like B-tree in terms of the search performance and memory utilization.

We do not have any analytical results for the worst and average case of the tree for the search, or height of the tree or node utilization, even though our experimental results shows it is like B-tree for large number of data and big branching factor in the internal node

# Builds on previous work

The approach identifies the previously-parsed prefixes of a source, creates parsers in the parser states corresponding to the identified prefix and parses the remaining portion of the translation unit. **Finally, we refer the reader to** [12] which utilizes prefix matching in data compression which is crucial in database and data communication applications

# Differs from previous work

Therefore, we only need to retrieve the leftmost and rightmost strings whose prefix is P. **However, based on Manber and Myers** [18] the leftmost string is adjacent to P’s position in the tree

# Contributions

We propose DMP-tree, Dynamic M-way Prefix tree, initially for the prefix matching. The prefix matching constitutes the base of some applications such as DNA matching, spell checking, online dictionaries, telephone directories and layers 3 and 4switching. The well known IP lookup problem in TCP/IP packet forwarding has motivated us to propose the DMP-tree data structures. However, it has broader applications and may used in problems in which a set of strings from an alphabet R can be prefixes of others as well as in general pattern matching. Previously proposed methods have used traditional sorting schemes for strings which are based on the character ordering in the alphabet and lengths of data strings. Unfortunately, those schemes cannot handle prefix matching directly and have to transfer them into range query which makes them very costly. The major problem preventing us from applying the usual indexing schemes such B is the lack of a well known mechanism to sort strings of different lengths. We started our work by a new scheme for comparing and sorting strings of different lengths as a foundation. Then we build our index tree on top of the new sorting and comparison scheme.

# Limitations

The difference between the minimum and maximum height of the tree is large for small numbers of branching factors and decreases as the branching factor increases. Eventually, in some point, the tree is balanced. According to the experiments, the average height of DMP-tree is O(Log {M, N) for large N and big branching factor M in the internal nodes.