Computers and Electrical Engineering 36 (2010) 818–834

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

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

Nasser Yazdani*, Hossein Mohammadi

Router Laboratory, ECE Department, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Available online 9 July 2008

Keywords:
String matching
Prefix matching
Multi indexing
DMP-tree
IP Lookup

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 Log_M^N for a large number of N, number of data elements, when the internal node branching factor M is big enough ($\geqslant 5$).

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Quickly and efficiently locating prefixes matching a query string is crucial in some applications. The prior assumption in these applications is that there are strings of an alphabet Σ which are ordered. The strings can have different lengths and be prefixes of each other. The following lists a few real world applications as examples for which the efficient prefix matching is very critical:

1. Routers forward an IP packet by finding the next hop on the path towards the final destination. To perform this, a routing table database consisting of pairs of IP prefixes, or network addresses, and their corresponding next hop addresses is searched to find the longest IP prefix matching the packet destination address. Finding the next hop becomes harder as the increasing number of hosts on the Internet increases the size of routing tables. On the other hand, as communication line speeds increase, the time to process and forward a packet gets smaller. For instance, a router connected to

* Corresponding author.
E-mail addresses: yazdani@ut.ac.ir (N. Yazdani), hosm@ece.ut.ac.ir (H. Mohammadi).

0045-7906/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
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 next hop address in 50 ns. This problem is referred as layer 3 switching in the network community. The alphabet in this application is very limited, only {0,1}.

2. Internet Service Providers (ISPs) like to provide different services to different customers. Some organizations filter packets from the outside world by installing firewalls to deny access to unauthorized sources. Supporting these functionalities requires packet filtering or packet classification mechanisms in layer 4 of TCP/IP protocols. Forwarding engines must be able to identify the context of packets and classify them based on their source and destination addresses, protocols, etc., at wire speed. Routers handle this by keeping a set of rules which applies to ranges of network addresses in a database. Thus, we have to deal with the prefix matching problem in two dimensional space; i.e., for the source and destination addresses of a packet.
3. String processing has been an active research area from the dawn of computer science. Availability of extremely large amounts of textual data such as medical documents, libraries, archives, marketing information, etc., have motivated and influenced this study. The produced results and algorithms have found application in many diverse fields ranging from dynamic dictionary matching [1] and spell checking to molecular biology [13] and DNA matching. In all of these applications there are a set of patterns which can be dynamic. Besides insertion and deletion, a user is interested in finding all occurrences of a query string in the data set. The general substring search can be translated into the prefix matching problem as will be explained latter.
4. Other applications include directory lookups for telephones and social security numbers [24]. Ref. [2] relates the prefix matching problem to computer speech recognition by compactly encoding pronunciation prefix trees. A method to improve the parsing process of source codes which uses prefix matching has been introduced in [8]. 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. The proposed approach there parses the input stream of data symbols into prefixes and data segments and uses the previously longest matching prefixes to compress the data.

The prefix matching search has been performed by the Trie structure [16]. A Trie is essentially an M-way tree where a branch in each node corresponds to a letter or character of alphabet Σ. A string is represented by a path from the root to a leaf node. The Trie structure has been modified and applied to most of the applications discussed above [24,21,22,19,10] The main problem with Trie is that it is inflexible; i.e. the number of branches corresponds to the number of characters, and keeps some extra nodes as place holders. Furthermore, in general, the search time is proportional to the length of the input strings. Others like Ferragina and Grossi [13] have tried to apply B-tree to the prefix matching problem. They have transferred the prefix matching query into the range query. It seems this approach does not work well in time critical applications such as IP lookup.

We propose DMP-tree as a new indexing and searching scheme for prefix matching. DMP-tree is a dynamic M-way tree which is built bottom up like B-tree. When data elements are disjoint, none of them is a prefix of other, the DMP-tree behaves exactly like B-tree. This is a nice property of our method. This is why we believe B-tree is a special case of DMP-tree. Our method can efficiently handle a wide range of prefix matching queries. We later explain how DMP-tree can be applied to the general pattern matching problem. However, in first step, besides the longest prefix match queries, we are also interested in exact match, smallest prefix match, all prefix match and listing all strings having prefix of a given pattern queries. Our solutions are efficient in memory usage and search time while scaling well to a larger data size or higher dimensions.

The rest of the paper is organized as follows. Section 2 explains the background of the work and addresses the basic issues. Section 3 discusses the DMP-tree structure. Application of DMP-tree to the general pattern matching query comes in Section 4. Section 5 discusses about general pattern matching. Section 6, contains our experimental results. We discuss related work in Section 6. Finally, Section 7 concludes the paper.

2. Background and basic issues

Tree structures keep the data elements sorted. Thus, in order to apply any tree structure to a data set, it is essential to have a mechanism to sort them. The traditional sort mechanism based on the value and lengths of strings [18] does not efficiently handle prefix matching queries and has to transfer them into range queries [13] of different lengths when the strings are prefixes of each others. The following definition gives a simple approach to compare and sort strings of different lengths. It is worth noting that we assume the characters in the alphabet are ordered.

Definition 1. Assume there are two strings $A = a_1a_2 \ldots a_n$ and $B = b_1b_2 \ldots b_m$, where a_i and b_j are characters of alphabet Σ and there is a character $\perp$ which belongs to. Σ Then

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 Σ.

Download English Version:

https://daneshyari.com/en/article/454154

Download Persian Version:

https://daneshyari.com/article/454154

Daneshyari.com