First Mini-Project:
Prefix Trees and Longest Prefix Matching

João Luís Sobrinho

I. Introduction

Every destination in the Internet has an *address*, which is binary string of fixed length. In IPv4, addresses have 32 bits, and in IPv6, they have 128 bits. Every data-packet sent into the Internet carries with it its destination address. Every router in the Internet maintains a *forwarding table*. A router matches the destination address of every incoming data-packet against its forwarding table to determine the neighbor router that gets the data-packet one-hop closer to the destination, subsequently forwarding the data-packet to that router. A forwarding table does not contain an entry for every possible address, because of the large number of such addresses. Instead, a forwarding table is a (binary) *prefix table*, mapping prefixes to next-hops. A *prefix* is a string of bits of length not longer than that of the addresses, representing all addresses whose first bits coincide with the prefix. For example, prefix 10 represents all addresses whose first two bits are 10. A *next-hop* is an identifier that singles out a neighbor router. Figure 1 (left) shows a prefix table, where ϵ is the empty prefix that represents all addresses. For example, the prefix table implies that data-packets with a destination address whose first two bits are 10 can be forwarded to next-hop 4.

A prefix table may contain pairs of prefixes such that the shorter prefix of the pair coincides with the first bits of the longer prefix. The *longest prefix match rule* specifies that an address is matched to the longest prefix that contains the address. For example, in the prefix table of Figure 1, a data-packet with address starting with

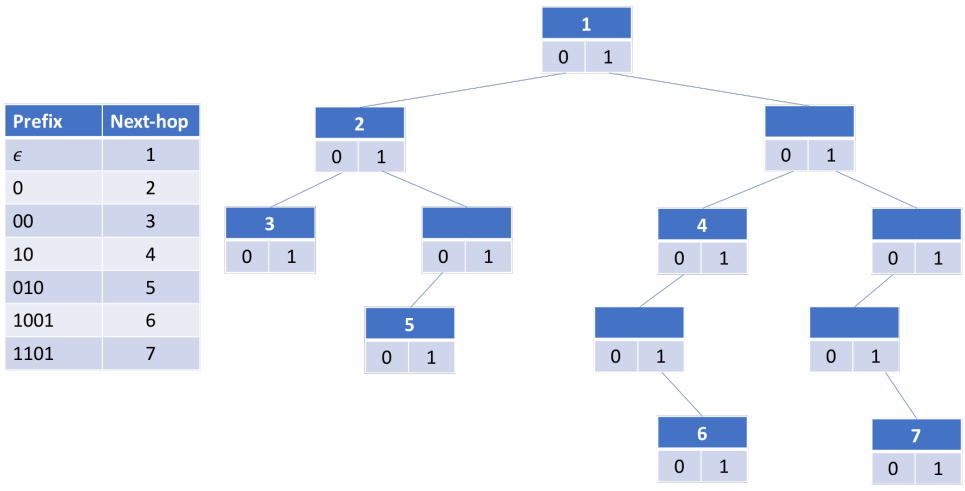

Figura 1. A prefix table and the corresponding prefix tree.

00101000 will match prefix 00 and will be forwarded to next-hop 3; a data-packet with destination address starting with 01110100 will match prefix 0 and will be forwarded to next-hop 2; and a data-packet with destination address starting with 11001010 will match prefix ϵ and will be forwarded to next-hop 1.

II. Prefix trees

The prefix table is accessed to look up addresses and return next-hops according to the longest prefix match rule. The look up must be executed at the very large speeds at which data-packets arrive at a router. It is very important to represent a prefix table in a structure that is parsimonious and potentiates quick address look ups. The typical representation of a prefix table is as a (binary) *prefix tree*. In such a tree, each node corresponds to a binary string. A node is associated to a next-hop if its binary string is a prefix in the prefix table. The node has a 0-child if there is a prefix whose first bits are those of the binary string of the node appended with a 0, and similarly for a 1-child. Figure 1 shows a prefix table and the corresponding prefix tree. Note that, although every node corresponds to a binary string, that string is not actually stored at the node; rather it can be computed by traversing the tree from the root to the node.

The maintenance of a prefix tree is performed by a routing process running in the router, at a speed that is slower than that at which data-packets arrive. It consists of an insert operation, which adds a prefix and corresponding next-hop to the prefix tree, and a deletion operation, which deletes a prefix from the prefix tree.

III. Two-bit prefix trees

The speed at which an address is looked up in a router depends on the number of accesses to its memory. In the worst case, every bit of an address must be compared against a bit of a prefix, so that the number of memory accesses equals the length of the addresses. The number of memory accesses can be cut by half by comparing an address against a prefix two bits at a time. However, this presupposes that all prefixes in a prefix table have even length. In order to cut by half the number of memory accesses, an original prefix tree must be expanded to an equivalent prefix tree where all prefixes are of even length. Two prefix trees are *equivalent* if they yield the same next-hop for the same address. A prefix table where every prefix has even length can be represented by a two-bit prefix tree, where each node may have up to four children corresponding to the pairs of bits 00, 01, 10, 11. The depth of the two-bit prefix tree is half that of the original prefix tree. Figure 2 shows a prefix table where every prefix has even length and the corresponding two-bit prefix tree. The prefix table of Figure 2 is equivalent to that of Figure 1.

IV. Your assignment

What you have to do. You have to develop a program that reads a prefix table from a text file and then interacts with a user, who can ask: (i) for the table to be printed to screen; (ii) to look up the next-hop of an address; (iii) to insert a prefix and the associated next-hop in the table; and (iv) to delete a prefix from the table. You may assume that addresses are strings of 16 bits. In order to develop your program, you must devise algorithms for the following functions:

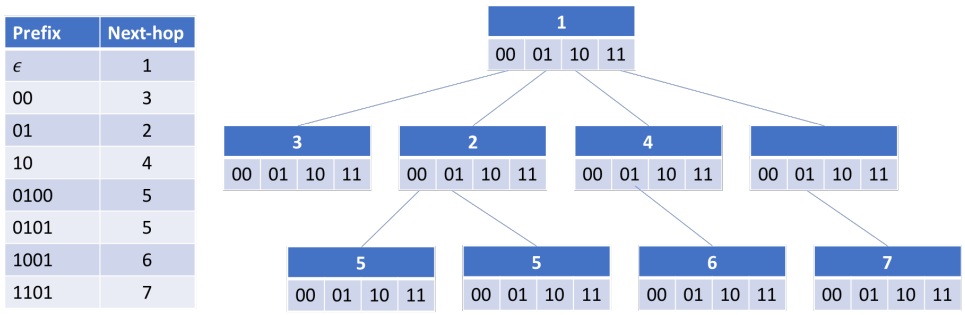

Figura 2. A prefix table where every prefix has even length and the corresponding two-bit prefix tree. The two-bit prefix tree is equivalent to that of the previous figure.

- *PrefixTree*, that reads a prefix table from a text file and returns a prefix tree representation of that table;
- *PrintTable*, that receives as input a prefix tree and prints to screen the corresponding prefix table;
- *LookUp*, that receives as input a prefix tree and an address and returns the next-hop for that address;
- *InsertPrefix*, that receives as input a prefix tree, a prefix and the associated next-hop, and returns a prefix tree with the prefix included;
- *DeletePrefix*, that receives as input a prefix tree and a prefix and returns a prefix tree with the prefix withdrawn.

For extra points, you should also devise algorithms for the following functions:

- *BinaryToTwoBit*, that receives as input a prefix tree and returns an equivalent two-bit prefix tree;
- *PrintTableEven*, that receives as input a two-bit prefix table and prints to screen the corresponding prefix table of even length prefixes.

What do you have to deliver, how, and when. You have to deliver the code of your program, duly commented, and a report of no more than three pages highlighting the pseudo-codes of your algorithms (at least the non-trivial ones) and a short discussion. The pseudo-code of an algorithm is a high-level description whose main purpose is to be easily understood by a human being, rather than be compiled to machine code. It may have the selection and iteration constructs of imperative programming languages, and calls to sub-routines, including recursive calls, but it has no detailed syntax and no declarations of variables (the names of variables should suggest their meanings). The short discussion can address the complexity of the algorithms, alternative algorithms, or aspects that you found particularly interesting. The code and the report should be sent in a .zip file to my email address with subject p1.<group number>.zip where <group number> is your group number. You will also have to deliver a printed version of the report. The deadline for the .zip file is October 13, 2017, 23:59. The printed report should be delivered at the class on October 16.

How I will evaluate your assignment. I will test your programs for correctness. I will evaluate the quality of your algorithms from your report. It is of the utmost importance that you write a clear report with neat, easy to understand, pseudo-codes. Remember that your writing is your way of communicating your ideas to others so that your work can be properly understood and publicized. I hope to have a discussion with you about your report shortly after delivery.