Project:
Prefix tables and longest match prefix rule

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* that maps 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 the set of all addresses. For example, the fourth entry in the prefix table states that all data-packets with a destination address whose first two bits are 10 may forwarded to next-hop 4.

A prefix q is more *specific* than a prefix p if $p \neq q$ and p coincides with the first bits of q. For example, prefix 1001 is more specific than prefix 10, whereas prefixes 00 and 1001 are not comparable in terms of specificity. The *longest prefix match rule* specifies that an address is matched to the longest prefix that contains the address.

Prefix	Next-hop
ϵ	1
0	2
00	3
10	4
010	5
1001	6
1101	7

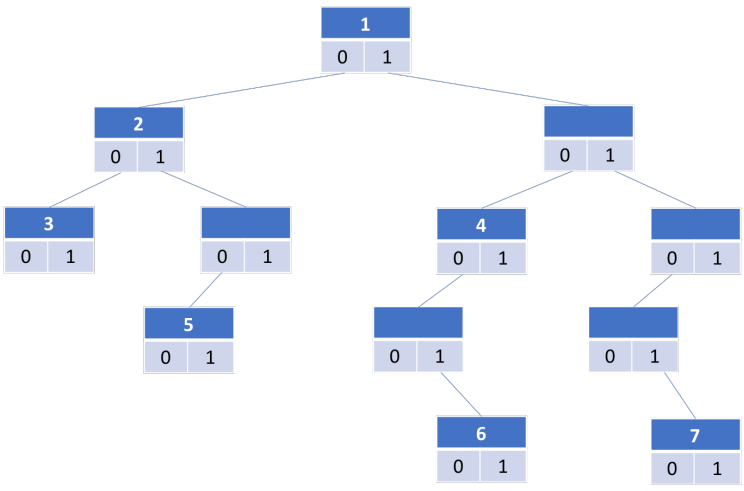

Figura 1. A prefix table and the corresponding prefix tree.

For example, in the prefix table of Figure 1, a data-packet with destination address starting with 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 the empty 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 as a data 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. For example, node 00 in the prefix tree of Figure 1 is associated to next-hop 2, whereas node 01 is not associate to a next-hop.

A prefix table held at a node changes dynamically through execution of a routing protocol running among all nodes of the network. The changes in a prefix table consist in adding a new entry and deleting an existing entry. These changes must be reflected in the prefix tree representing the table, but occur at speeds that are orders of magnitude slower than that of address look up. Hopefully, the routing protocol produces prefix tables that are consistent in the sense that data-packets are guided from origins to destinations along selected paths, but such a routing problem is outside the scope of this project.

III. Equivalent prefix trees

The size of the prefix tables maintained at core routers of the Internet, measured by the number of entries in the table, is very large indeed, on the order of hundreds of thousands. Therefore, it is reasonable to ask whether it is possible to compress a prefix tree while maintaining the same forwarding behavior. Two prefix trees are *equivalent* if the address look up of every prefix produces the same next-hop in both trees.

A prefix tree is a *compression* of another if they are equivalent and the former has a smaller number of entries. *Filtering* allows compression of a prefix tree. Given prefixes p and pp' pointing to the same next-hop and such that there is no other prefix whose specificity is in between p and pp', prefix pp' can be removed from the tree without altering forwarding behavior. *Aggregation* also allows compression of a prefix tree. Given a set of prefixes $pp'_1 \dots pp'_n$ all with the same next-hop and such that the union of the addresses represented by them makes up the addresses represented by p, all those prefixes can be substituted by the sinlge prefix p. Given a prefix tree, it is even possible to compress that tree *optimally*, thus producing an equivalent tree with minimum number of entries. Such a compression is typically not used in practice (why?). As an example, all three prefix trees depicted in Figure 2 are equivalent and the one on the right is optimal.

Prefix	Next-hop
ϵ	1
0	2
00	3
10	4
11	5
010	5
1001	4

Prefix	Next-hop
ϵ	1
0	2
00	3
10	4
11	5
010	5

Prefix	Next-hop
ϵ	5
00	3
10	4
011	2

Figura 2. Equivalent prefix trees.

IV. Your assignment

What you have to do. The short answer is that you have to investigate the algorithmic aspects of prefix trees, including address look up, insertion, deletion, and compression, implement the corresponding algorithms, and write a report with a summary of your findings. In the long answer, 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 an entry in the table; (iv) to delete an entry from the table; (v) to compress the table. The algorithm to compress a table does not need to be optimal. You may assume that addresses are strings of 16 bits. In order to develop your program, you will devise algorithms for the following functions:

- *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;
- *CompressTree*, that receives as input the prefix tree of a prefix table and returns another prefix tree representing a prefix table with a smaller number of entries.

What do you have to deliver, how, and when. You have to deliver a report of no more than three pages with your algorithms explained very clearly and with a short discussion of your findings. For the non-trivial algorithms you should use pseudo-code. 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 to 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 the syntax is not detailed and typically variables are not declared—the names of variables should suggest their meanings. For the most difficult algorithms, you may should include an example of execution. The short discussion can address the complexity of the algorithms, alternative algorithms, and aspects that you want to highlight. Remember that your writing is your way of communicating your ideas to others so that your work can be properly understood and publicized. So, It is of the utmost importance that you write a clear and concise report with easy-to-understand explanations. And make it an interesting read!

You also have to deliver the code of your program duly documented. The report and the code should be sent in a .zip file to my email address with subject p1.<group number>.zip where <group number> is your group number (only .zip files will be accepted). You will also have to deliver a printed version of the report. The deadline for the .zip file is October 18, 2019, 23:59. The printed report should be delivered at the class on October 21.

How I will evaluate your assignment. I will evaluate the quality of your report, test the code for correctness, and have a discussion with you at the end of the semester. However, I hope to provide some feedback shortly after your delivery.