Taken by: Nate Barish

Network Algorithms Lecture Notes 9-10-2012

Announcement: Remote students will not need to scribe but still need to do the special HW assignment.

Binary Search on Prefix Length

Goal: What is the longest prefix match given an IP address.

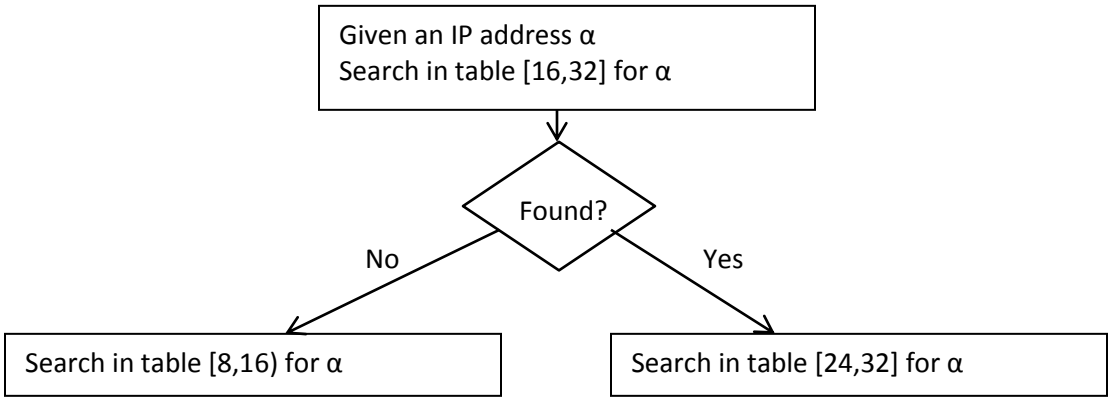
Trivial Way: search 1 bit at a time. Slow: O(n) where n is the number of bits in the ip address.

We want to reduce the complexity of the algorithm from O(n) to O(log(n)).

For 32 bits IP: 32 vs 5 memory accesses

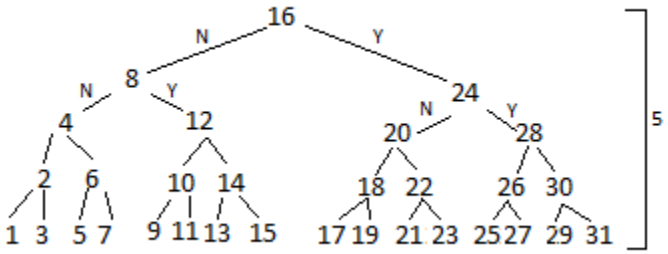
Algorithm

We will do the search in a binary search fashion.



The table magically (for now) tells you if there is a prefix in the table that possibly matches your ip address.

Fully expanded the algorithm looks like this for a 32 bit ip.



Data Structure

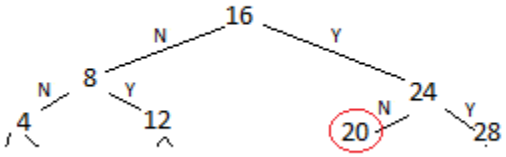
What should we put into table [16,32]?

1. It should contain all prefix rules of length exactly 16. (prefix)
2. It should include the first 16 bits of all prefix rules of length greater than 16. (marker)

If you match a prefix you are guaranteed to have a match of 16 bits (or more).

But if you only match a marker you could end up with a longer match or it may not match anything longer than 16. Meaning the longest prefix length may be lass then 16.

We need to create one of these tables for every node in the graph.



For instance at the node 20 we will create a table [20, 24) containing:

1. All prefix rules exactly 20 bits long.
2. The first 20 bits of all prefix rules of length [20, 24)

What happens when you hit a marker? You are told there might be a longer match but what if you don't find one?

A naive solution would be to back track, but this would increase the O() of our algorithm.

Instead we do pre-computation: For every marker pre-compute the longest prefix match for the marker among the rules of length less than the marker.

So if you match a marker but don't find anything you back and use that rule.

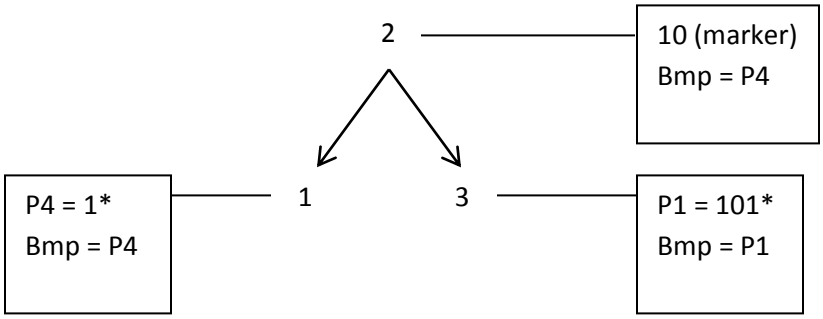
Simple Example

Suppose we have 3 bit ip addresses

Rules:

P4 1*
P1 101*

Table for each of the prefix lengths



Suppose you search for “100”

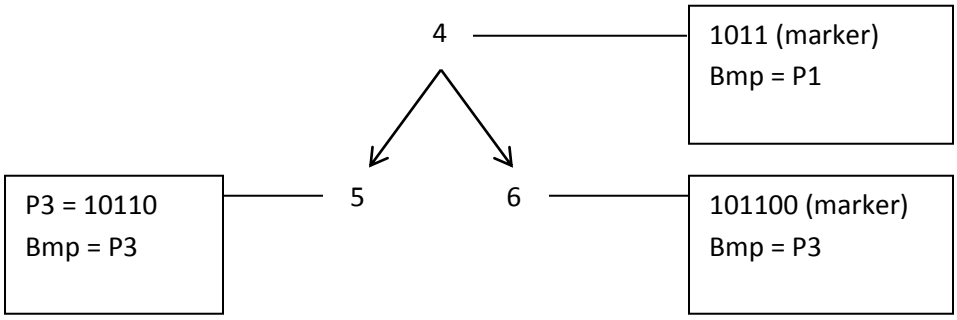
You see 10 in Table 2, but it does not match anything in Table 3 so you fall back to matching P4.

Longer Example

Suppose we have 8 bit ip addresses

Rules:

P1 101*
P2 10110010*
P3 10110*



Details

You will have 33 tables (for a 32 bit ip)

All prefixes in each table will be the same length.

To make tables fast use a hash table with perfect hashing to do the searching.

Speed: $\text{Log}_2(N)$ memory access (where N= number of bits in ip)

Memory Use: memory will blow up faster. But at most you will duplicate each prefix 32 times. In practice you duplicate each one only about 3 times.