CS-7260 Scribe Notes: IP Lookup

By Xiang Fu

September 5, 2012

1. Previous

IP Lookup: Trie$\begin{cases} 1-bit: too\ many\ memory\ accesses \\ multi-bit: more\ storage \end{cases}$

2. Multi-bit Trie

Given a set of rules, give an upper limit on the # of memory accesses (at most k times).

How to minimize the storage cost?

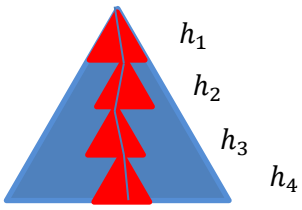
2.1 Fixed-Stride Tries

As a lookup would take at most k times memory accesses, so for the trie, we cut it at most $k-1$ times to get k levels.

We denote the height of level i to the above segment level as h_i.

For the level i, $forest(i)$ denotes leaves in level i, except those stumps. A stump is a leaf with no child.

For $forest(i)$, each leaf is a table of size 2^{h_i}. And for a long prefix, we need to blow up the whole tree through levels.

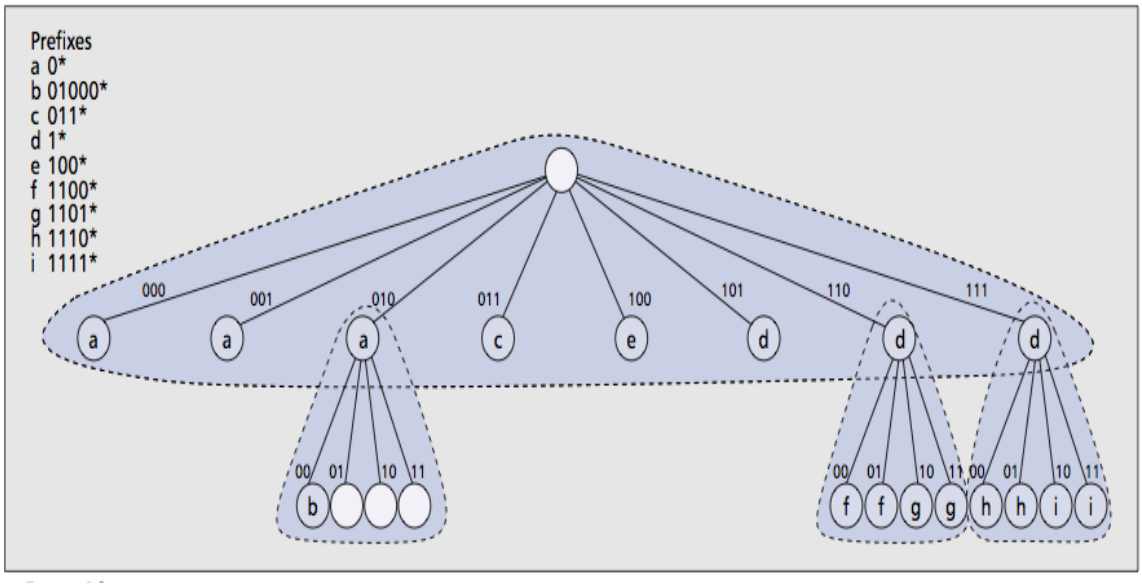


Below is an example.

The graph below indicates a fixed stride trie with $h_1 = 3, h_2 = 2$.

Leaves 010*,110*,111* are in the $forest(3)$. 000*,001*,011*,100*,101* are stumps.

The storage cost for $level\ 3$ is $2^3 = 8$. The cost for $level\ 5$ is $2^2 \times |forest(3)| = 4 \times 3 = 12$.



Fixed-Stride Trie Tree

So the problem can be figured out:

Objective:

$$\min(storage\ cost(trie|h_1, h_2, \dots, h_k))$$

Subject to:

IP is full 32-bit address. So

$$h_1 + h_2 + \cdots + h_k = 32$$

Method:

Dynamic programming:

l denotes the current level we are not working at, m denotes the number of strides left. As we are not at the level l, so the height j of the next stride is in the range $[1,32 - l]$. For each leaf in $forest(l)$, it has a table to contain children with size 2^j. So the total cost is the children size of $forest(j)$ + the cost of the rest trie. Then it can be done recursively.

$$Opt(l,m) = \begin{cases} \min\limits_{1 \le j \le 32-l} \left(2^j \times |forest(l)| + Opt(l+j, m-1)\right), & m > 1 \\ 2^{32-l} \times |forest(l)|, & m = 1 \end{cases}$$

The optimized solution is $Opt(0,k)$.

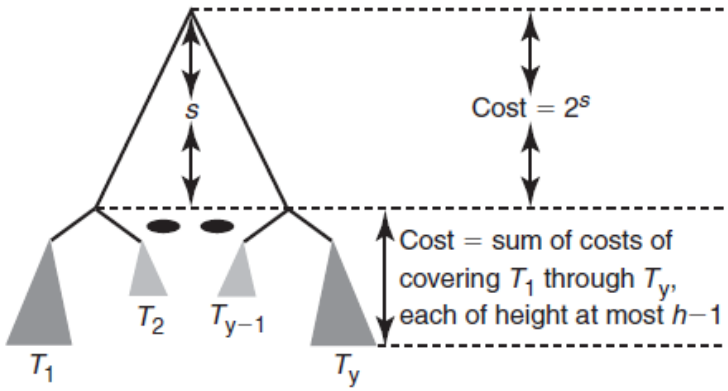
2.2 Variable-Stride Trie

If the size of subtrees are very different, fixed-stride trie is not optimized.

For the above fixed trie, the leftmost subtree only contains one leaf. But we still need to compare two bits for that node. It's obviously that we can optimize the tree by using one bit comparison.

In a variable-stride trie, the number of bits examined by each node can vary, even for the nodes at the same level.

We can also solve this problem via dynamic programming. For the root node, we need to concern the children nodes separately.



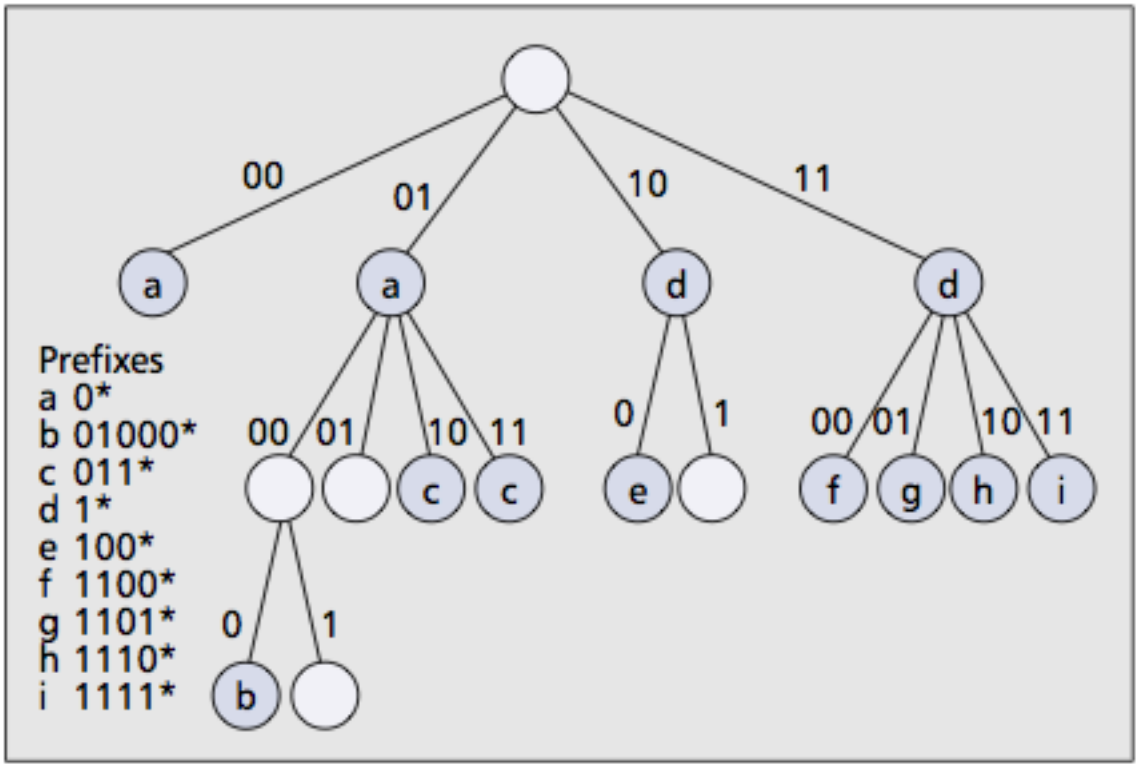
Cost of a Variable-Stride Trie Tree

r denotes the current root of the subtree, a particular node in the tree. m denotes the # of strides left. $forest(r,j)$ denotes the forest starts from root r, at level j.

$$Opt(r,m) = \min_{1 \le j \le 32-level(r)} \begin{cases} 2^j + \sum_{t \in forest(r,j)} Opt(t, m-1), & m > 1 \\ 2^{32-level(r)}, & m = 1 \end{cases}$$

The optimized solution is $Opt(\epsilon, k)$.

Below is the optimized solution of previous example for variable-stride trie.



Variable-Stride Trie Tree