Assignment 5

CS 181AG: Network Algorithmics

Due: Oct 4, 2022 10pm PT

This assignment will help you get familiar with unibit tries. It includes a coding component, short-answer questions related to your code, a separate problem, and a reading assignment with questions.

Problem 1: Trie to Return the Correct Interface. In assignment5.py, we create a unibit trie. Let's first go over the components:

1. **Node**: This is a simple trie node. It has fields for the interface name (e.g., P1) (which might be None if the node does not correspond to a prefix), left and right Nodes (which might also be None), and a parent (not used in this assignment)

2. **Trie**: This is the structure itself. It is initialized with a root. For adding and lookup, it simply hands the request over to the root, as we are using a recursive implementation

3. **add**: This is how we add new interface - prefix mappings to the trie. As mentioned above, the trie simply hands over the add request to the root. The root examines the first character of the prefix and decides whether to branch left or right. Let's say hypothetically it sees a 0, so it branches left. If the left node does not exist, it creates it. Then, it removes the first character of the prefix and tells the left node to add what's left of the prefix (using recursion!). As an example, if we were adding P1: 0* to a trie that only contains a root, the root node sees the 0 decides to branch left and it creates a left node. It then tells the left node to add *. The left node sees the special character * and simply adds P1 as its interface name. More details are in the comments.

4. **main:** In main, we first create a trie. When then read in mappings between interface names and prefixes from prefixes.txt. Do not edit prefixes.txt. Finally, we print the output for looking up 5 IP addresses, which you will report (see below).

   Your tasks:

   (a) Implement the lookup function for the Node class. Currently, lookup is implemented for the trie class, which simply hands the request to the root node. The lookup function for the Node class takes an IP address (as a string) and a best matching prefix (also a string, initialized to None). Similar to the add function, the lookup function should recursively search through the trie. As it searches, it must keep track of the best matching prefix so far. It returns the best matching prefix when the search stops, i.e., it sees a 0 and there is no left node or it sees a 1 and there is no right node. When you are done, submit your code to Assignment 5 Programming.

   (b) In main, there are 5 print statements calling the lookup function with IP strings. Please report which interface each IP string is sent to using your code.

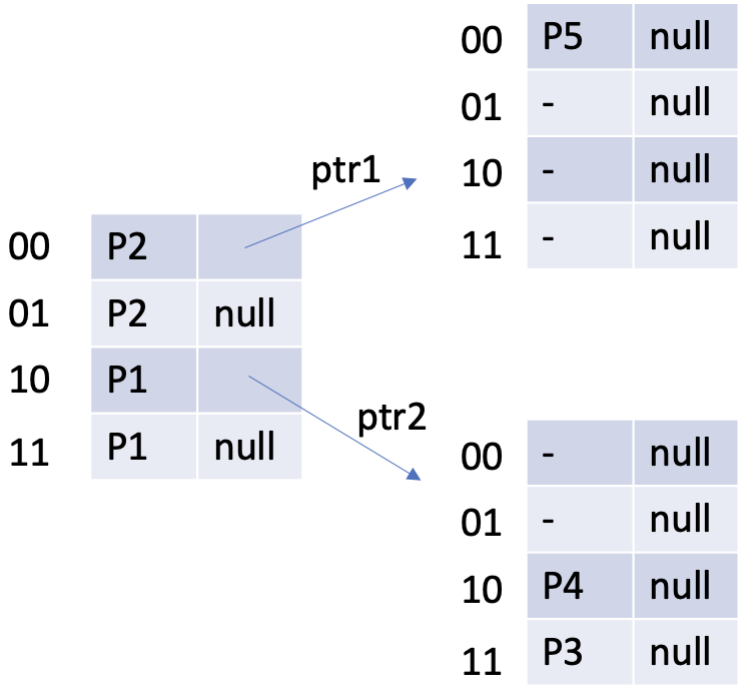
   (c) Answer the following questions: What is the worst-case number of lookups using this trie? What is one thing we can do to lower the worst-case lookup time?

   Hints:

   prefixes.txt contains many prefixes and it will be hard to debug using all the prefixes. I would suggest initially commenting out everything in main except except where the trie is created and instead adding a few small prefixes to the trie. Then sanity check your solution with a few shorter IP strings before uncommenting.

The only part of the code you need to touch is the lookup function in the Node class. If you would like to add other statements for debugging, that's fine.

Problem 2: Lulea Compression. For each of the three nodes shown below, please write the bit array and the value array using the Lulea compression scheme we covered in class.



Answer 2:

This answer will be of the form of an enumeration going from left → right and from up → down.

1. Bit: 1111, Value: [ptr1, P2, ptr2, P1].
2. Bit: 1100, Value: [P5, P2]
3. Bit: 1011, Value: [P1, P4, P3]

Problem 3: Reading. Now that we've learned about the Network Layer (IP), let's read about an alternative to IP that is currently in progress. This technical report explains NDN. Please read through Section 2.2.3 answer the following questions:

1. What is one argument for why NDN is better suited for how the internet works today?
2. What are some challenges in getting widespread use of NDN?

Answer 3:

1. One argument for why NDN is better suited for how the internet works today is because of the fact that we have an infinite set of host names in NDN which alleviates the issue with IPv4 which has exhausted the number of IP addresses and would provide a more significant and lasting answer that IPv6 is currently trying to solve with the finiteness of IP addresses.

2. Some challenges in getting widespread use of NDN is the fact that it is a wholly new alternative to IP that would require increased functionality beyond what IPv4 current offers, as well as finding an appropriate solution for the massive scaling of routing tables as more addresses are added to keep optimal speed as well as finding solutions to the article described issue of DoS attacks on the networks.

Problem 4. How long did this assignment take you?

Answer 4:

This assignment took me ≈ 5 hours.

Problem 5: Extra (optional) reading. When we left off in class, we said the Lulea compression helps the reduce memory usage and achieves reasonably fast lookup, but insertion was quite slow. Section 11.8 of the book describes ways of making insertion faster. There is no work or credit or extra credit associated with reading this - just more information if it interests you.