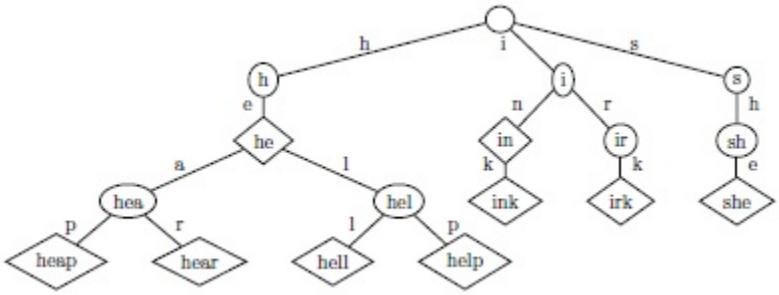
CSCI 104L Lecture 25: Tries

Tries



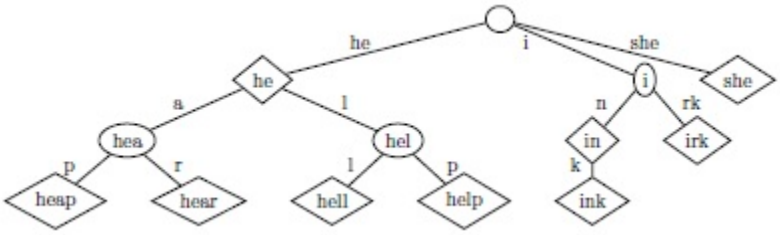
Suppose we have a binary search tree with n nodes, where each key is a string with k characters.

Question 1. For large k, would it be accurate to say that find takes $O(\log n)$ time? What would be a more accurate runtime analysis?

Trie, from re**trie**val are intended specifically for string keys which are very long.

This significantly cuts down on comparison on each level, as you're only looking at one character.

You can imagine there being a lot of wasted space in a Trie. You can improve on this by making a Compressed Trie.



Each node in a compressed trie has at least one of the following properties:

- It is the root node.
- It is a word node.
- It has at least two children.

Question 2. Construct a compressed trie with the following words: ten, tent, then, tense, tens, tenth

Network Routing: Incoming packets have a destination IP address, such as 132.125.73.60. Quickly determine the output port to forward the request through.

The IP address is actually written in binary, so the above IP address is 10000100.01111101.01001001.00111100

Question 3. What would be a good data structure to solve the Network Routing problem?

Suffix Trees

Tries can be thought of as prefix trees: given a prefix of a string, you can quickly identify the possible ways to finish it. Thus it is often used for auto-complete.

What if we want something more powerful: the possibility to match a substring, rather than just a prefix?

A suffix tree of a word W is a compressed trie of all possible suffixes of W.