Data Structures

Letter Tree (Trie)

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

A text task

- Given a dictionary of N words, check if a word/prefix exists in it
  - There are 1 million+ words in the Korean language
  - In fact, it could be Q queries!
- Assume each word has O(L) length
- Solution 1: use an array
  - However, searching for a word/prefix is O(NL), which is very expensive!
    - *O(L) for comparing 2 words. Comparing integers is O(1)*
- Solution 2: Use an AVL tree of words
  - As in the AVL homework, we can generate all prefixes, adding them to the tree
  - For N words, we generate N^2 words. So search is O(L * log(N^2)) = O(Logn)
- Can we find an approach that is no worse than O(L)?
- *Similar applications: Autocompletion, Spell checker*

m-ary tree

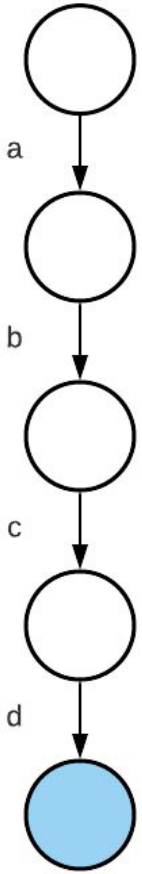
- A binary tree is a tree where a node has up to 2 children
- A ternary tree is a tree where a node has up to 3 children
- An m-ary tree is a tree where a node has up to m children
- Can we use an m-ary tree for this task?
- Assume we target only English letters; so m = 26
- How can we represent a set of English words using an m-ary tree?

Letter Tree (Trie)

- In the simplest form, a trie is an m-ary tree where each edge has a letter
- In a binary tree, each node represents an inserted item
- In a letter tree, a word of L letters spans L nodes!
- Let's trace an example to help visualize this:
- Assume we want to insert the (*fictional new*) following words:
  - abcd
  - xyz
  - abf
  - xn
  - ab
  - bcd

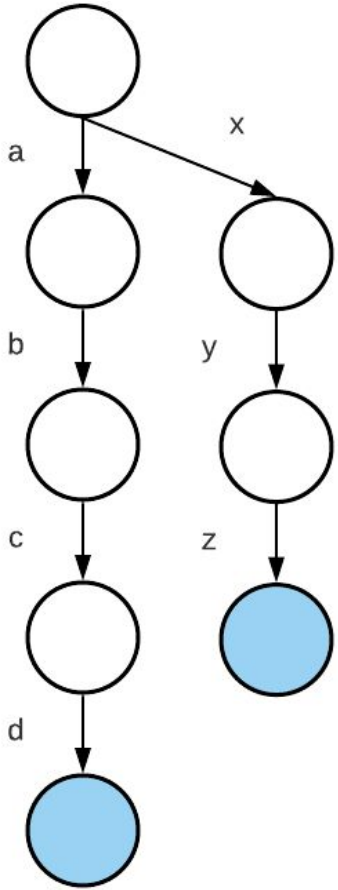
Insert abcd

- Observations
- abcd contains 4 letters ⇒ we have 5 nodes
  - The top node is the trie's root
- Node values are empty!
- Each edge has a letter, giving us a total of abcd
- The last node is colored (marked as leaf)
- This tree (or trie!) indicates that we have:
  - 3 prefixes:a, ab, abc
  - 1 word: abcd



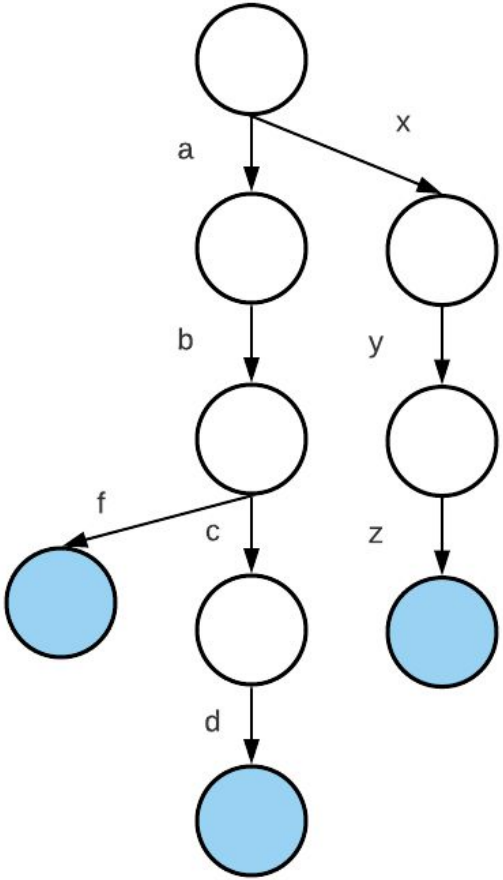
Insert xyz

- This is an entirely new word, with a new collection of letters
- We can see that the root produces a new edge (x), and then starts a chain for yz
- This tree now has 2 full words:
  - abcd, xyz
  - + their prefixes



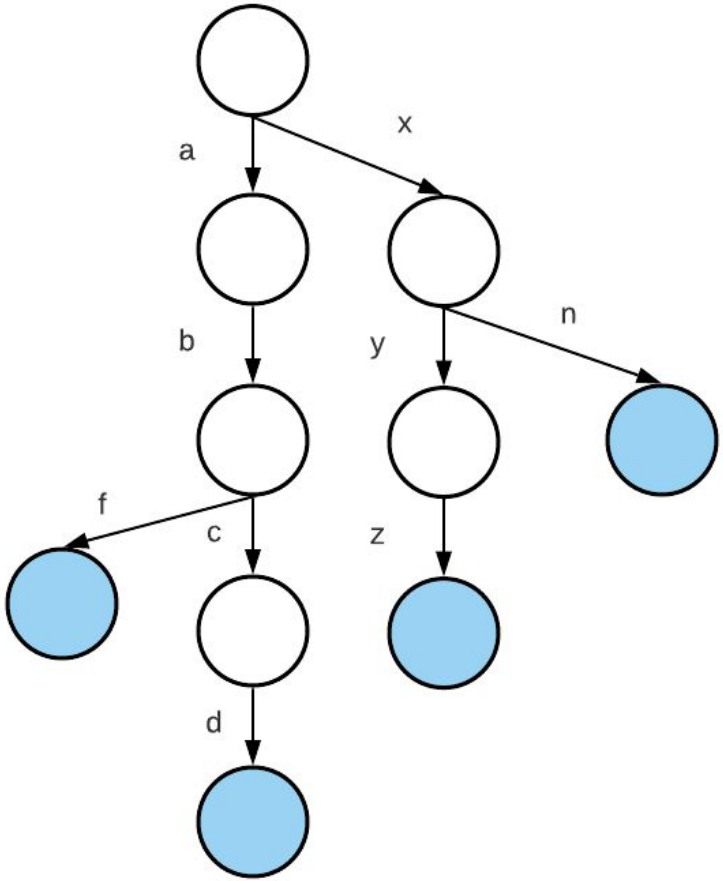
Insert abf

- Now, we want to insert **abf** from the root
- It starts with adding a
  - Since 'a' already exists in the letter tree, use it!
- Add b from node(a)
  - Again, 'b' already exists from node 'a', so let's use this as well!
- Add f from node(b)
  - 'f' doesn't exist in the tree. More importantly, it doesn't exist from 'b'!
  - So, create the new 'f' node
  - Is this the last letter in the word? Mark it as a full word!



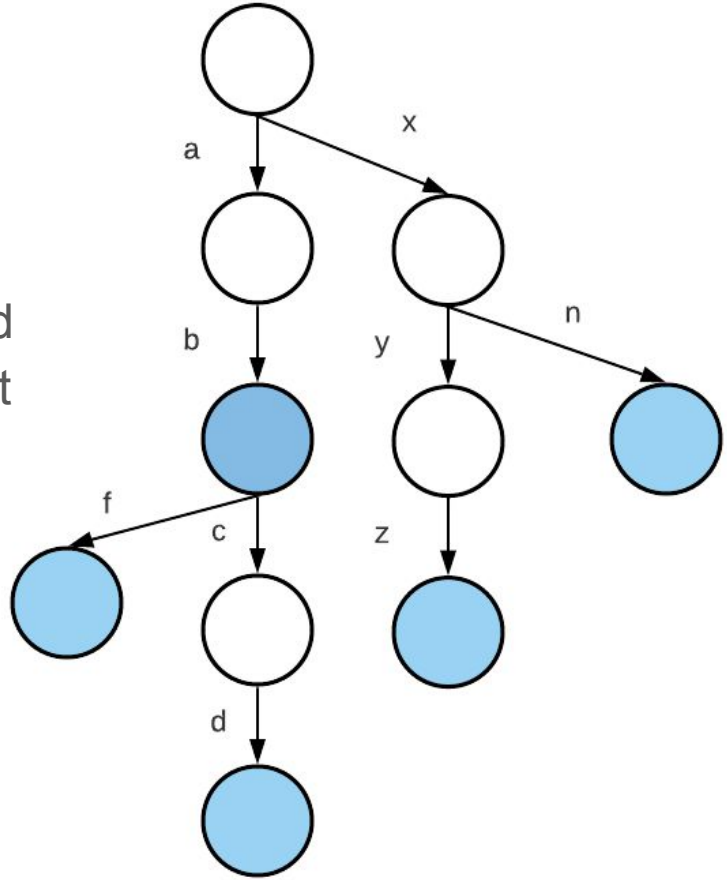
Insert xn

- This follows similar logic to before
- Use the 'x', then add 'n'
- We've now added 4 words:
  - abcd, abf, xyz, xn
  - + all of their prefixes!



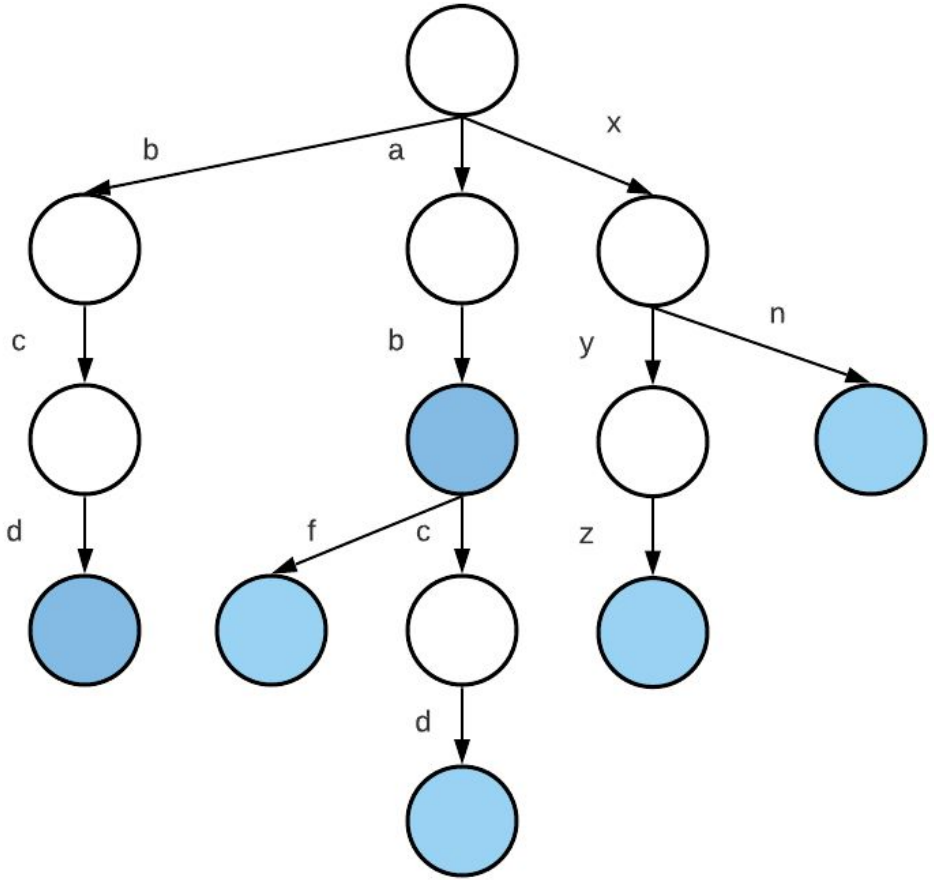
Insert ab

- After finding 'a' from the root, we move and also find 'b'
- Previously ab was not marked as a full word
- But now we should label it, and indicate that the 'b' is also a leaf node in this case



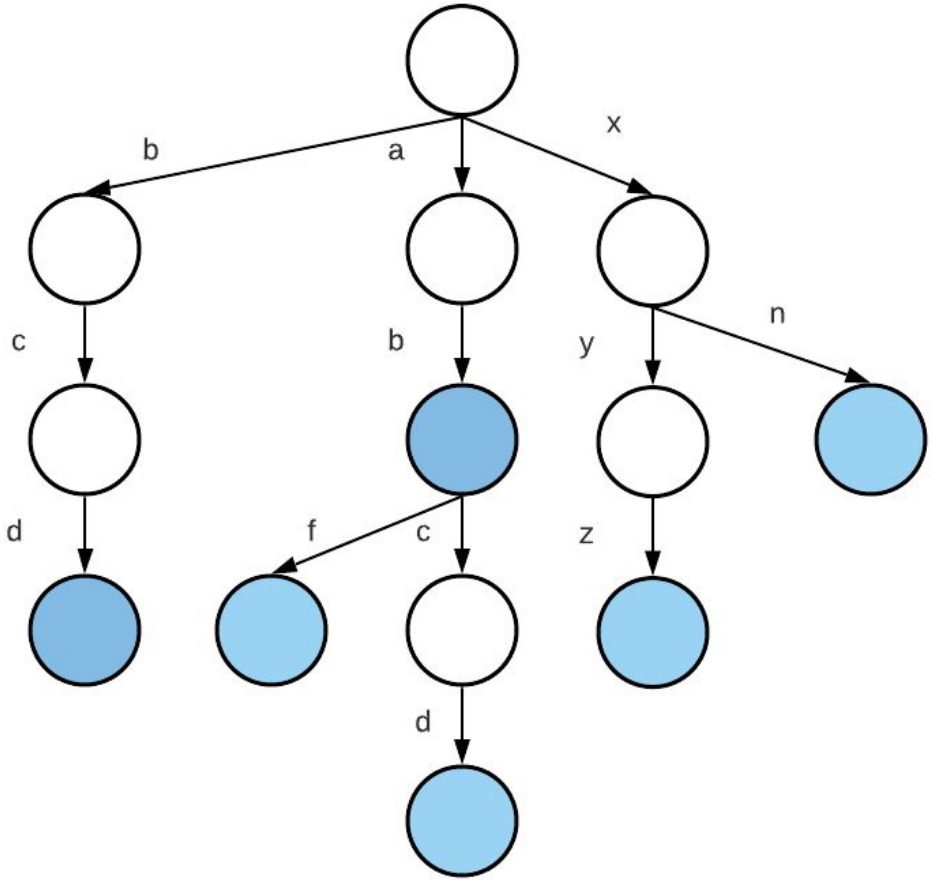
Insert bcd

- From the root, we start by looking for 'b'.
  - It isn't found!
  - So, add letter 'b'
- Then c, then d
- Observe: several edges can have the same letter
  - Think of this as similar to how child nodes share the same parent node, as seen in previous 'tree' sections



Searching the tree

- How can we know the word abcd exists? Or that we have prefix xy?
- It's similar to the insertion strategy
- Follow the edges for as long as you can keep going
  - If can’t but still target word is done, then not even found
  - If found, then either full word or not (just prefix)



Trie and interviews

- Trie is a common topic in interviews so good to know
- Tries are not efficient. There are other much more complex data structures
  - Suffix tree and suffix array [good to know at least the names for an interview]
  - They are in the **intersection** of data structures and string manipulation algorithms
- Your turn: How can we implement this data structure?
  - Some implementations are actually very simple!
  - Think for 15 minutes!

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."