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
  - ~1 million words in Korean Language
  - In fact, it could be Q queries!
- Assume each word has O(L) length
- Solution 1: Use array
  - Now searching a word/prefix is O(NL), which is very expensive!
    - *O(L) for comparing 2 words. Comparing integers is O(1)*
- Solution 2: Use AVL tree of words
  - Like AVL homework, we can generate all prefixes and add in tree
  - For N words, we have N^2 generated words. So search is O(L * log(N^2) = O(Logn)
- Can we search for an approach that is O(L) only!
- *Similar application: Autocompletion, Spell checker*

m-ary tree

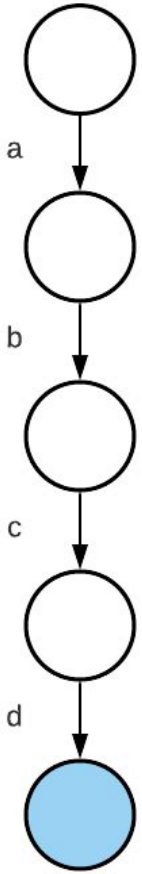
- A binary tree is a tree where a node has up to 2 children
- A ternary tree is a tree where a node has up to 3 children
- An m-ary tree is a tree where a node has up to m children
- Can we use m-ary tree for this task?
- Assume we target only English letters, then m = 26
- But how can we represent set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

Letter Tree (Trie)

- In its simple form, It is an m-ary tree where edge has a letter.
- In a binary tree, each node represent an inserted item
- But in letter tree, a word of L letters, span L nodes!
- Better let’s trace an example
- Assume we want to insert the following words
  - abcd
  - xyz
  - abf
  - xn
  - ab
  - bcd

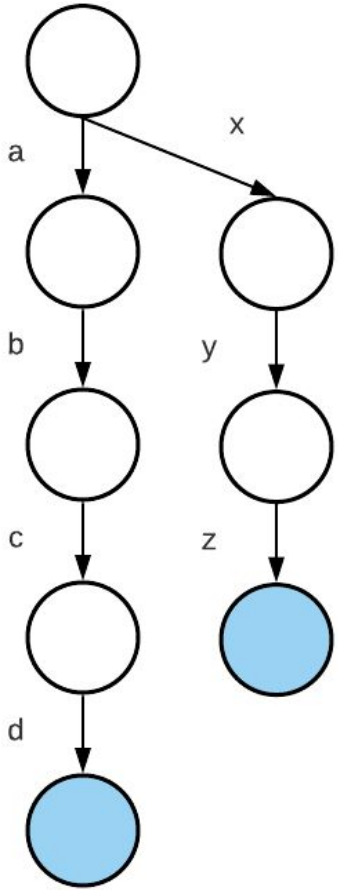
Insert abcd

- Observations
- Abcd is 4 words ⇒ we have 5 nodes
  - Top node is the trie root
- Nodes values are empty!
- Each edge has a letter, total abcd
- Last node is colored (marked as leaf)
- This tree means we have
  - 3 prefixes:a, ab, abc
  - 1 word: abcd



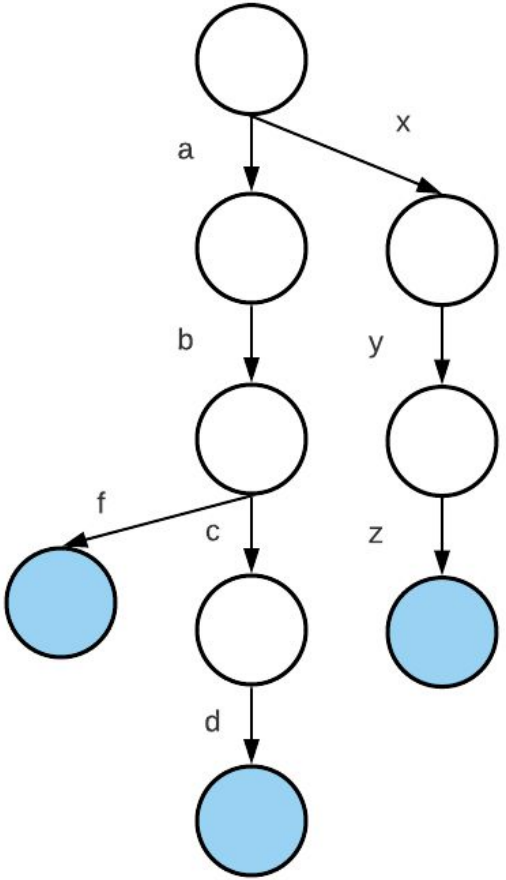
Insert xyz

- With the new word, we see root started new edge (x) then chain for yz
- This tree now has 2 words
  - abcd, xyz
  - + their prefixes



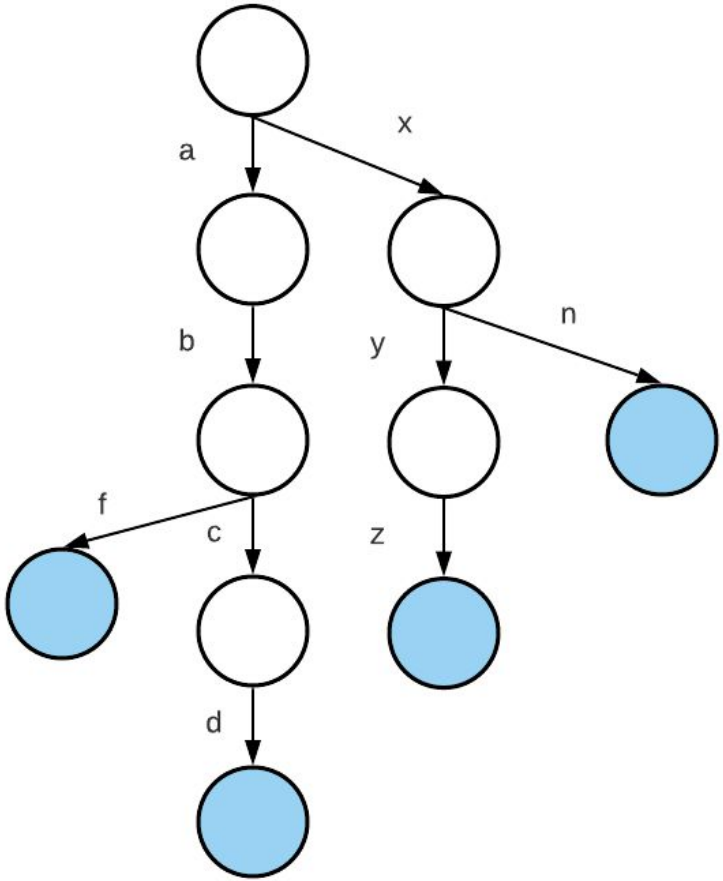
Insert abf

- Now root want to insert abf
- It starts with adding a
  - But it exists already, use it
- Add b from node(a)
  - But it exists already, use it
- Add f from node(b)
  - Not exists
  - Create it
  - Last letter? Mark as full word!



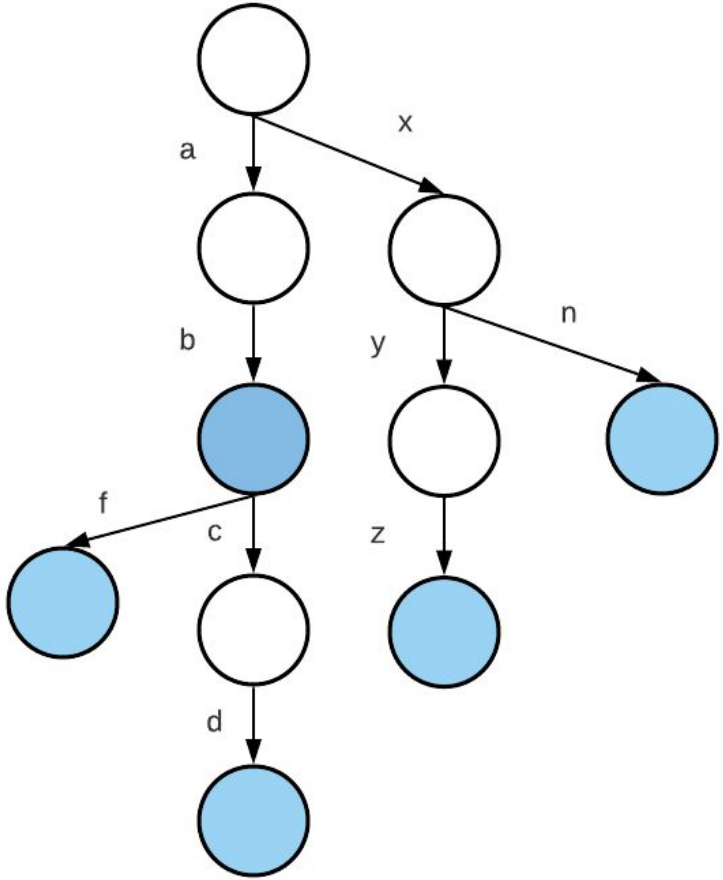
Insert xn

- Follows a similar logic
- Use x, then add n
- Now we have words
  - abcd, abf, xyz, xn
  - + all of their prefixes!



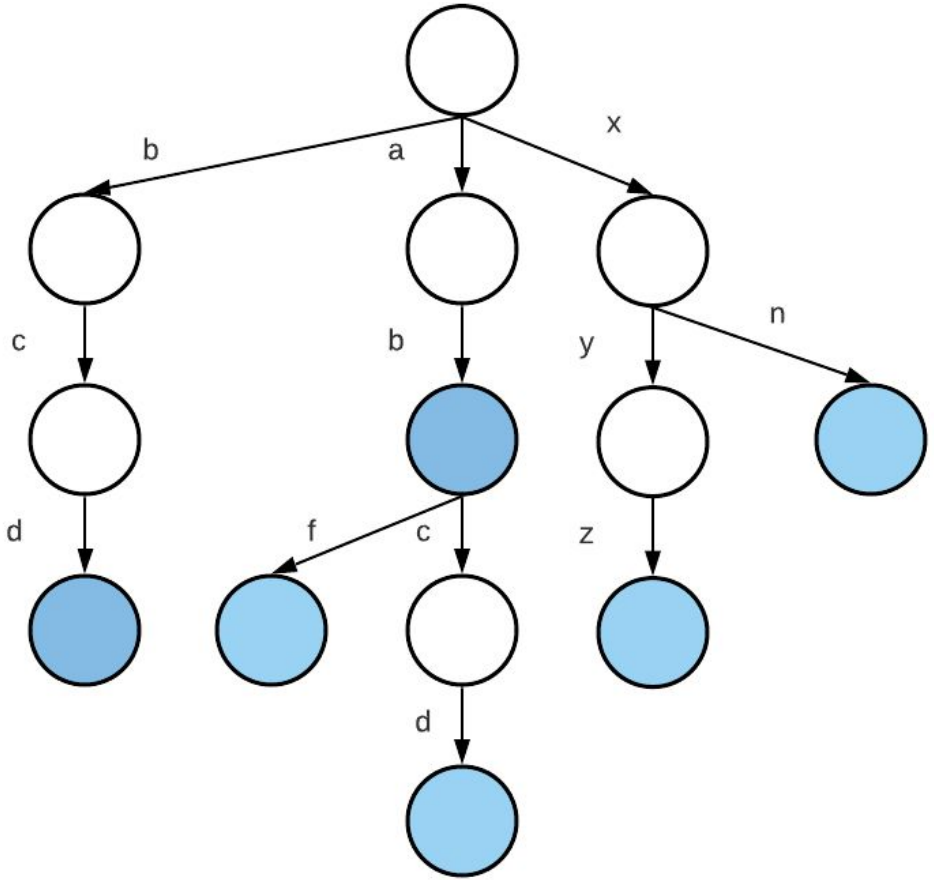
Insert ab

- We find a, then move and also find b
- Previously ab was not marked as full word
- But now we should label it



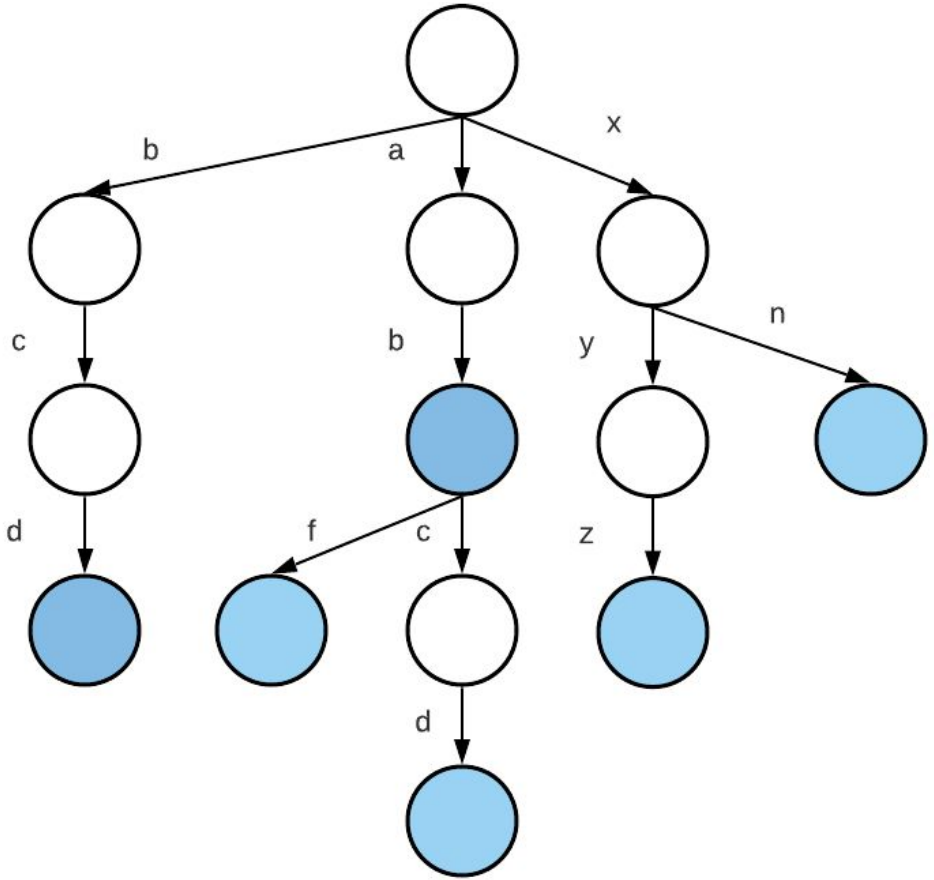
Insert bcd

- From root, we add letter b
- Then c, then d
- Observe: several edges can have same letter
  - Think relative to parent



Searching the tree

- How can we know word abcd exists? Or we have prefix xy?
- Same insertion strategy
- Follow the edges, 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 common in interviews so good to know
- Trie is not efficient. There are other much 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?
  - Actually some implementations are very simple!
  - Think for 15 minutes!

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."