Tries Data Structure

Tries are special and useful data structure that are based on prefix of a string.

It is used to represent the Retrieval of data thus the name Trie.
Trie is an efficient information retrieval data structure.

Properties Us

1. Trie is tree.
2. Stores a set of string.
3. Every node consists of atmost 26 children. (a, ... z)
4. Every node (except root) will store a letter in alphabet.
5. the children of a node are alphabetically stored.

e.g S = {bear, bell, bid, bull, buy, sell, stock, stop}

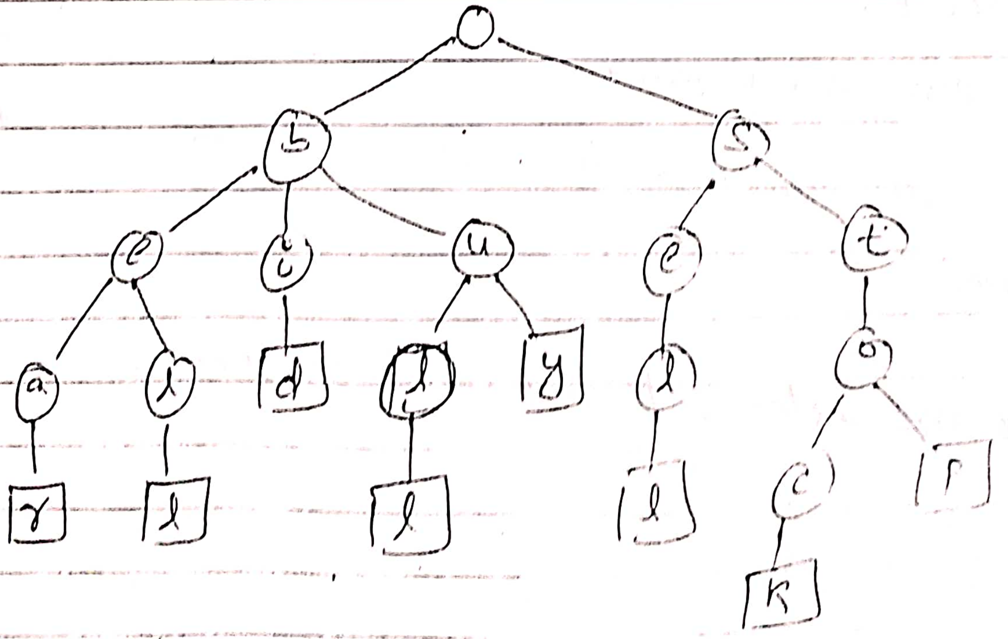


fig trie

Trie is an efficient information retrieval data structure.
Using Trie, search complexities can be brought to optimal
limit (Key length). If we store key in Binary Search Tree,
a well balanced BST will need time proportional to
m * log n where m is maximum string length and
N is the number of keys in tree.

Using Trie, we can search the key in O(m) time.

However the penalty is on Trie storage requirements.

<u>Why Trie</u>

① With Trie we can insert, find strings in O(L) time
where L is the length of a single word. This is obviously
faster than BST. This is faster than Hashing because
of the way it is implemented. We do not need to
compute any hash function, No collision handling is required

② We can easily print all words in alphabetical order

③ We can efficiently do prefix search (or auto complete) with
trie

Trie are faster but requires huge memory.

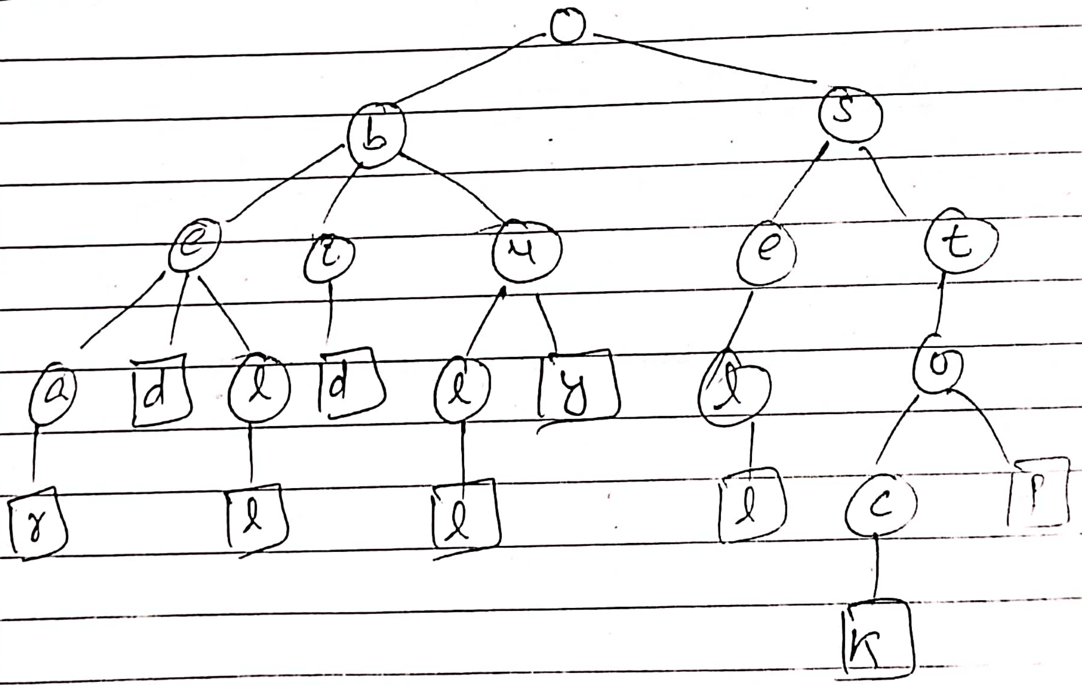
Running time for operations

A standard trie uses $O(n)$ space.

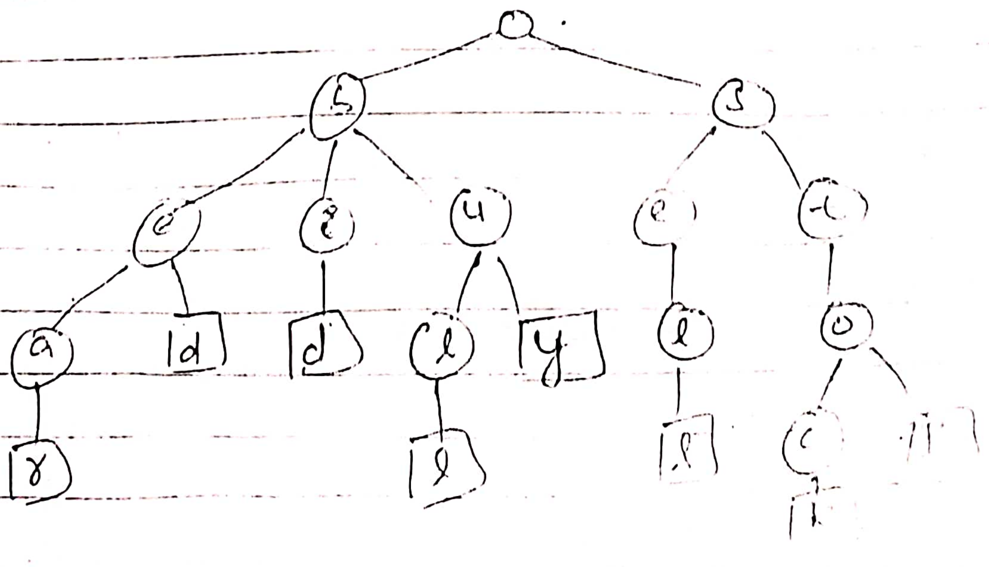
Operations (find, insert, remove) take time $O(dm)$ each where

n = total size of strings in S.
m = size of string involved in operation
d = alphabet size.

Insert "bed"



Remove "bell" : start from backward



b
s
e
i
u
e
t
a
d
d
l
y
l
o
r
l
l
c
p
k