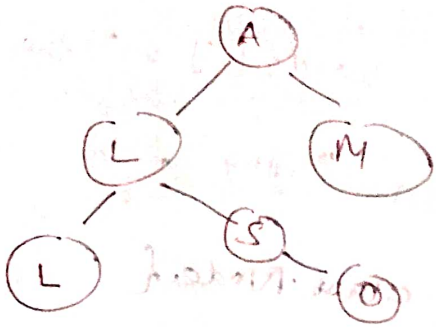
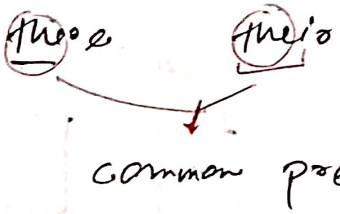
Trie · Data Structure

- Prefix
- digital search
- Retrieval tree



e.g → Prefix – comma -in

Two words:



What is Trie?

words[] = "the", "a", "there", "their", "any".

↓

array.

Time Complexity

BST:

Search

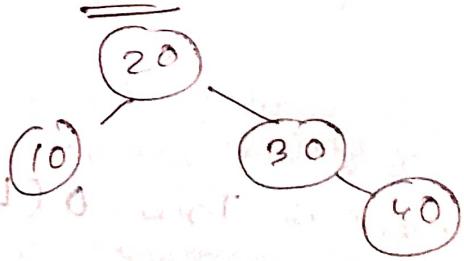
O(h)

↓

height.

O(n).

AVL



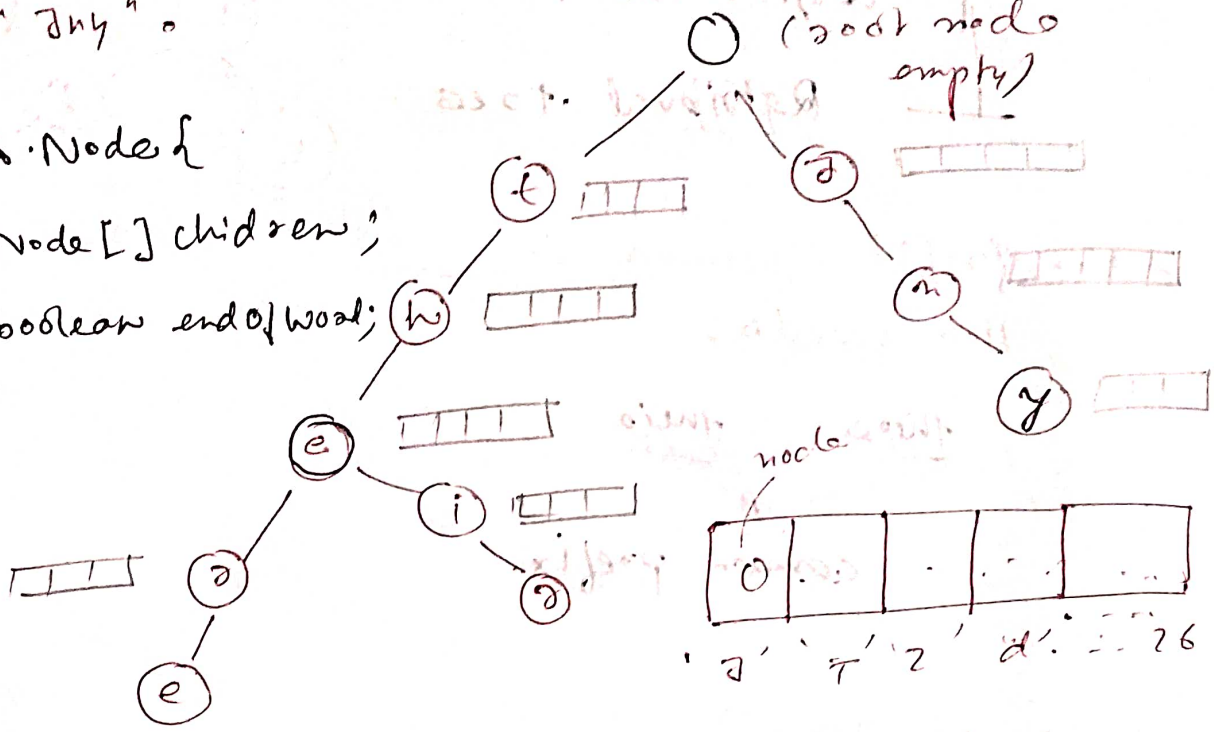
Search : O(log n)

Search → O(L)

In Trie we store letters by letters.

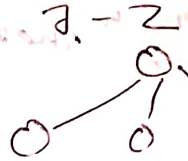
What is Trie ?

words [] = "the", "a", "there", "their", "any".




```
class Node {
    Node[] children;
    boolean endOfWord;
}
```

e.g - a, z



@ : 26 nodes !

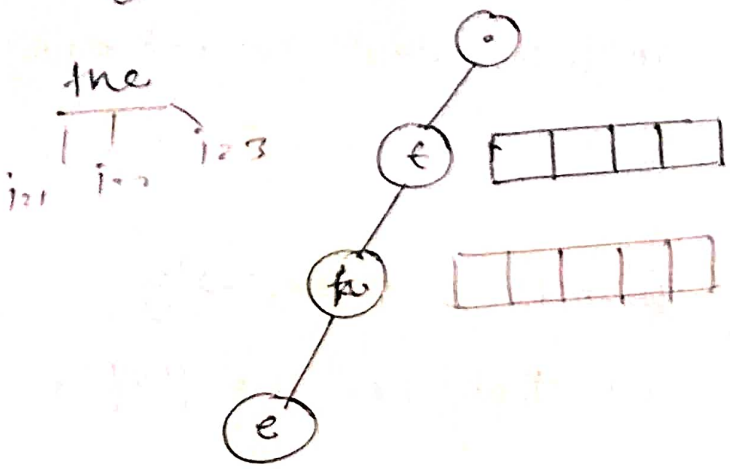
```
static class Node {
    int data;
    Node left;
    Node right;
}
```

Insert in Trie O(L)

words [] = "the", "a", "there", "their", "any".

```
code for (int i = 0 to word.length)
    idx = word.charAt(i) - 'a'
    if (root.children[idx] == null)
        new Node
    root = root.children[idx]
```

Insertion in Trie :



root
for (int i=0; i< to word.length; i++)
{ idx = word.charAt(i) - 'a'.

Search in Trie O(L)

Key = "this" → true

Key = "thoo" → false

Key = "an"

Word Break Problem

words[] = {i, like, sam, samsung, mobile, ice}.

Word Break Problem (Trie data structure)

Approach

words[] = {i, like, sam, samsung, mobile, ice}

key = "like samsung".

Starts with Problem

Create a function boolean startsWith(String prefix) for a trie.

- Returns true if there is a previously inserted string word that has the prefix & false otherwise.

words[] = {"apple", "app", "mango", "man", "woman"}

prefix = "app" output: true

prefix = "moon" output: false.

```
for (int i=0 ; to prefix.length(); i++)
        idx = prefix.charAt(i) - 'a';
if (curr.children[idx] == null)
        return false;
curr = curr.children[idx];
return true;
```