Trie 1

Content

Spell Checker

playgrind

scout

class

claay

N words collection of words max size is L

↙ List of String ↘ Hashset of String

abc → Creating a hash will take O(L) TC where L is length of word.

→ Searching for a word in Hashset of size N
O(L) will be TC to create a hash

→ O(L)

→ Searching for a word in a list of string {N}
word length = L

O(N * L)
↓
TC to compare a single word in dictionary of words.

Auto complete

$$\begin{tikzcd} \text{pla} \arrow[r] & \text{place} \\ & \text{play} \\ & \text{plate} \end{tikzcd}$$

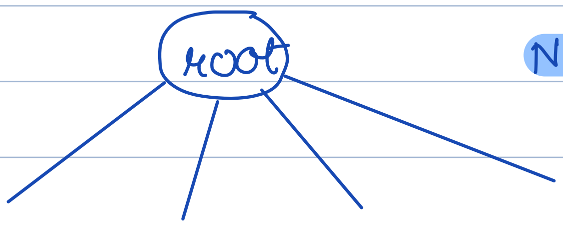
$$\begin{tikzcd} & \text{N words collection of words max size is L} \arrow[dl] \arrow[dr] & \\ \text{List of String} & & \text{Hashset of String} \end{tikzcd}$$

$\longrightarrow$ Sort the list of string TC : O(NL)

. . . . place plate play . . . .

Trie is a heirarchical DS thats optimised to store list of words and perform search operations

$\searrow$ Prefix Tree



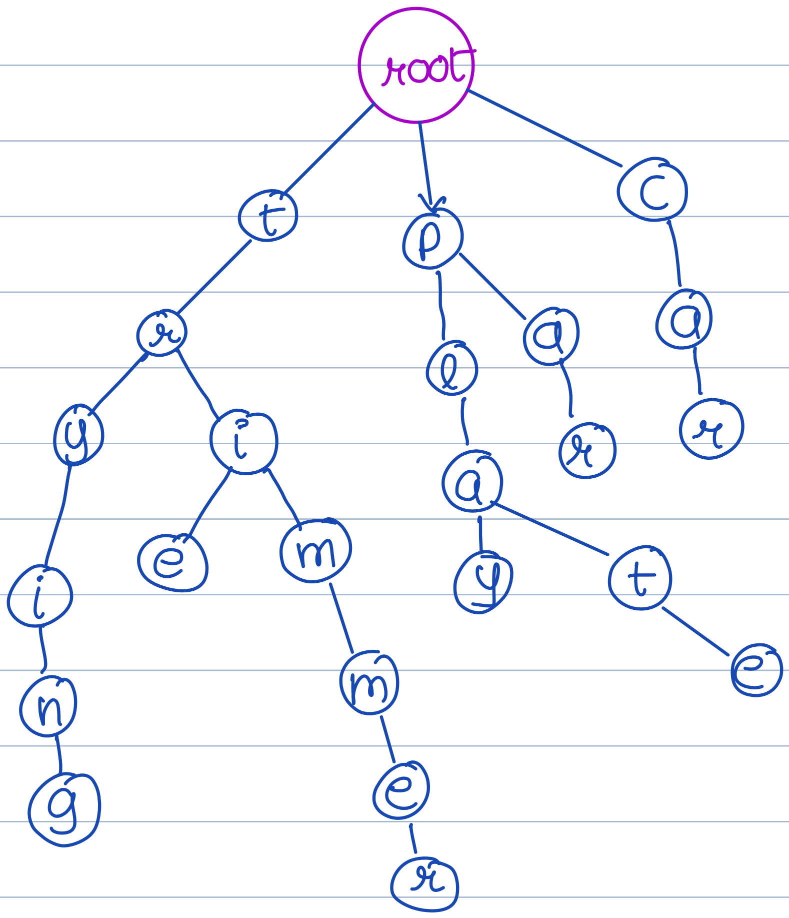
N - ary tree

Insert a word in a trie

try trim trie play trying

dict

plate car par trimmer pla



```
class TrieNode {
    char data
    TrieNode[] children

    TrieNode() {
        data = '#'
        chidren = new TrieNode[26]
    }
}
```

Search a word in Trie

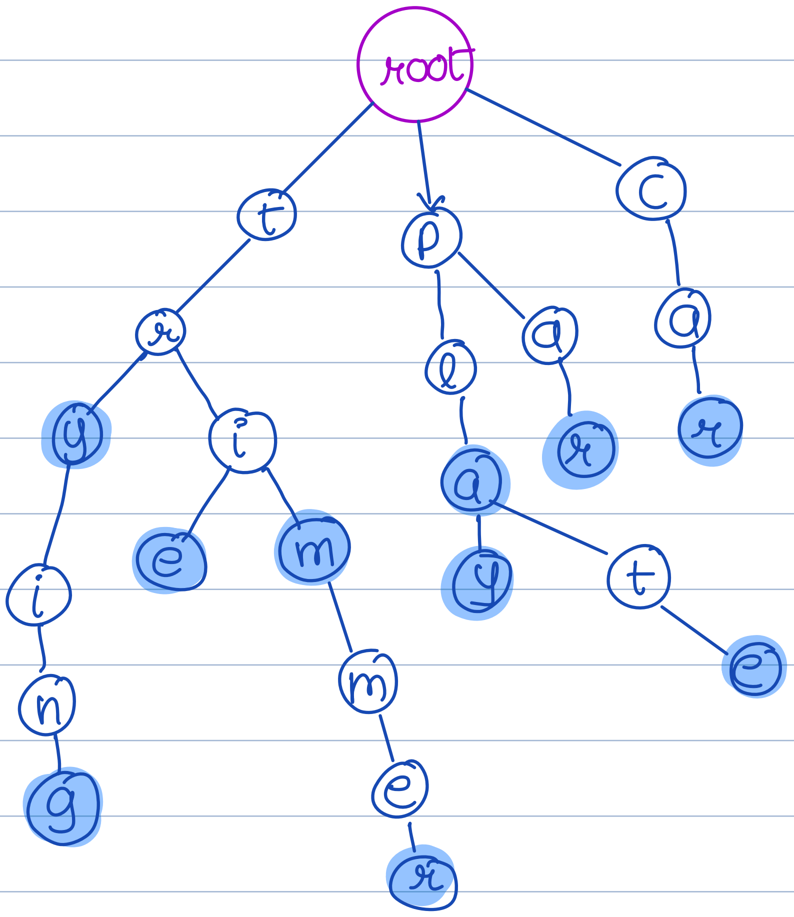
search (trie) $\longrightarrow$ T

search (tri) $\longrightarrow$ F

search (try) $\longrightarrow$ T

search (pla) $\longrightarrow$ T

try	trim	trie	play	trying
plate	car	par	trimmer	pla

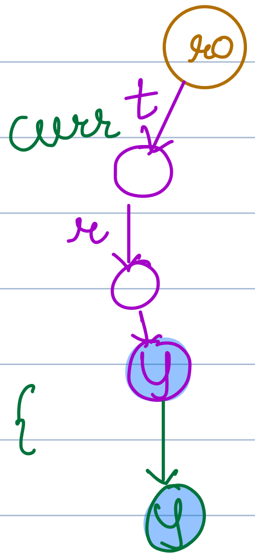


```
class TrieNode {
|       TrieNode[] children
|       bool eow
|
|       TrieNode () {
|       |    chidren  =  new  TrieNode[26]
|       |    eow  =  false
|       }
}
```

pseudocode Insert

insert (try)
insert (tryy)

```
void insert ( TrieNode root , String word ) {
    curr = root

    for( i=0 ; i<N ; i++) {
        Char ch = word.charAt(i)
        idx = ch - 'a'

        if ( curr.children [idx] == null ) {
            child = new TrieNode ()
            curr.children [idx] = child
        }

        curr = curr.children [idx]
    }

    curr.eow = true
}
```



TC : to insert a word of length L in a trie
$O(L)$

SC: $O(L)$

NOTE: Insert all dictionary words inside trie before searching.

Algo steps {searching}

$$\begin{tikzcd} \arrow[r] & \text{Go char by char and see if child exists in the trie} \arrow[dl, "\text{yes}"'] \arrow[dr] & \\ \text{proceed} & & \text{false} \end{tikzcd}$$

→ At the last node return eow

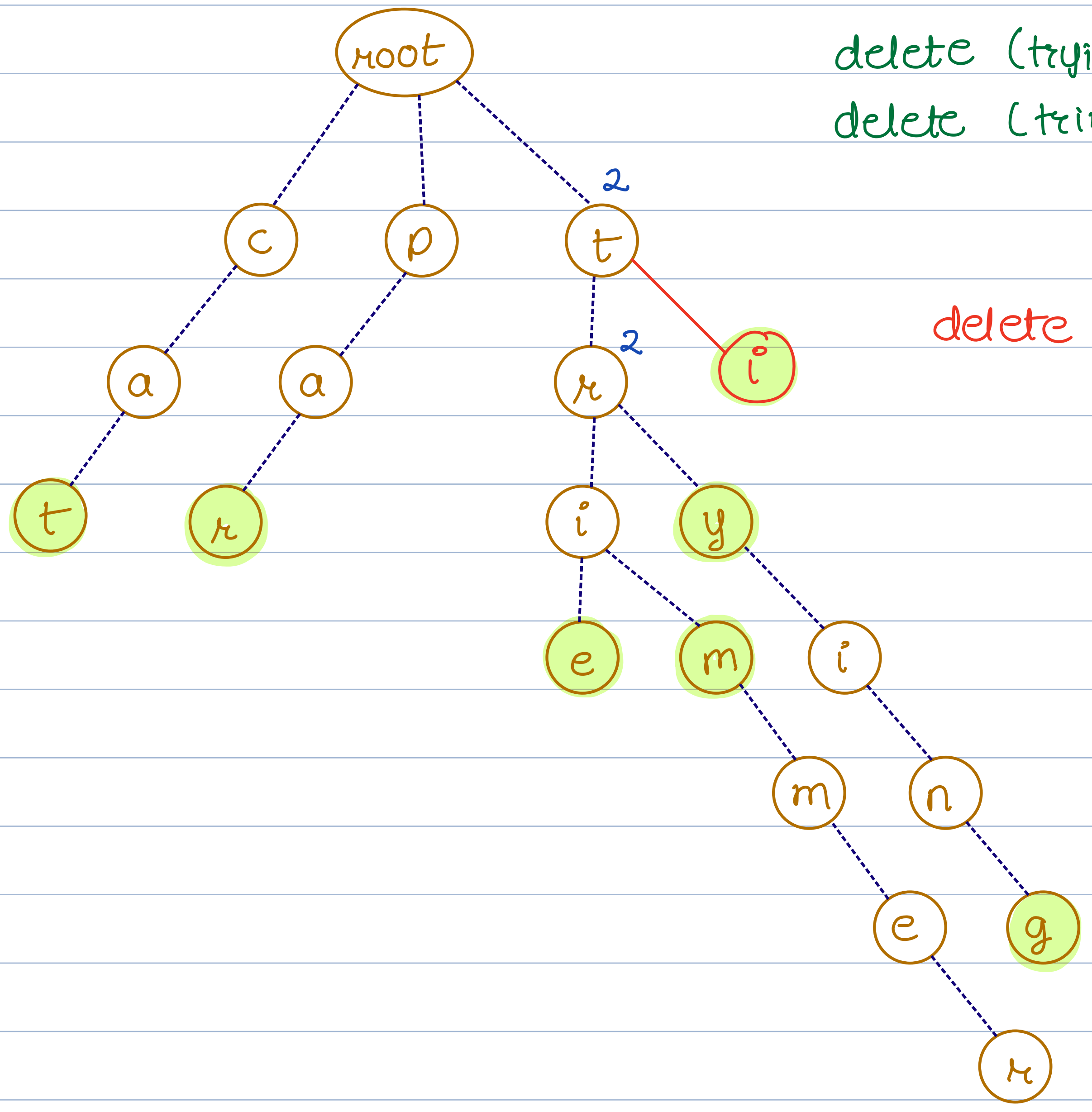
Pseudocode Search

```
boolean search ( TrieNode root, String word ) {
    curr = root

    for( i=0 ; i<N ; i++) {
        Char ch = word.charAt(i)
        idx = ch - 'a'

        if ( curr.children[idx] == null ) {
            return false
        }

        curr = curr.children [idx]

    }

    return curr.eow
}
```

TC : O(L)
SC : O(1)

Break : 8:32

Deletion in a trie



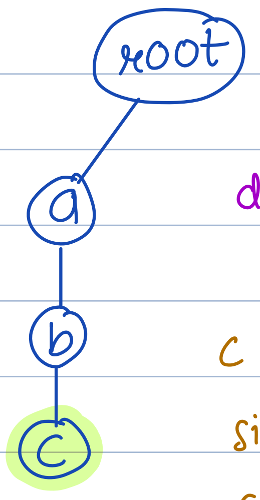
delete (trying)
delete (trimmer)

delete (t i)

Idea 1 Search for the word .

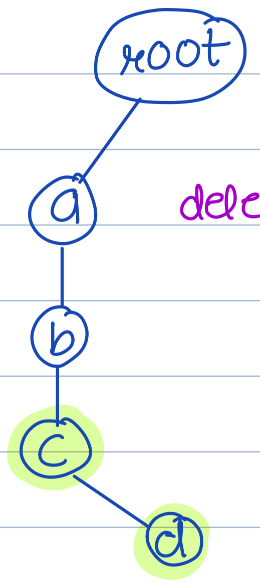
$$\begin{tikzcd} & \text{Search for the word} \arrow[dl, "\text{present}"] \arrow[dr, "\text{not present}"] & \\ \text{eow = false} & & \text{do nothing} \end{tikzcd}$$

Observation 1> we can only delete the last node if there are no chidren



delete (abc)

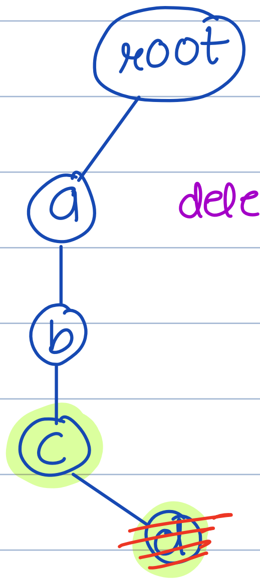
c can be deleted since it has no children



delete (abc)

mark ⓒ.eow = false

Observation 2> we can keep on deleting till eow is false



delete (abcd)

c cant be deleted because its eow

⇒ The node which cannot be deleted either are eow or has chidren more than 1 children.

Keep track of the last node that cannot be delete
$\longrightarrow$ no. of children > 1 or eow == true.

pseudocode delete

```
void delete ( root, word ) {
    // search and mark eow as false
    curr = root,
    lastNode = null        // TrieNode that cannot be deleted
    nextChar = '-'

    for( i=0 ; i<N ; i++) {
        Char ch = word.charAt(i)
        idx = ch - 'a'

        childCount = getCount ( curr ) // non null nodes
        if (childCount > 1 || curr.eow ) {
            lastNode = curr
            nextChar = ch
        }

        curr = curr.children [idx]
    }

    childCount = getCount ( curr )
    if (childCount >= 1) {
        return
    }
    lastNode.children [nextChar - 'a'] = null
}
```

TC : O(L)
SC : O(1)

Shortest Unique Prefix

Find the shortest prefix to represent each word.

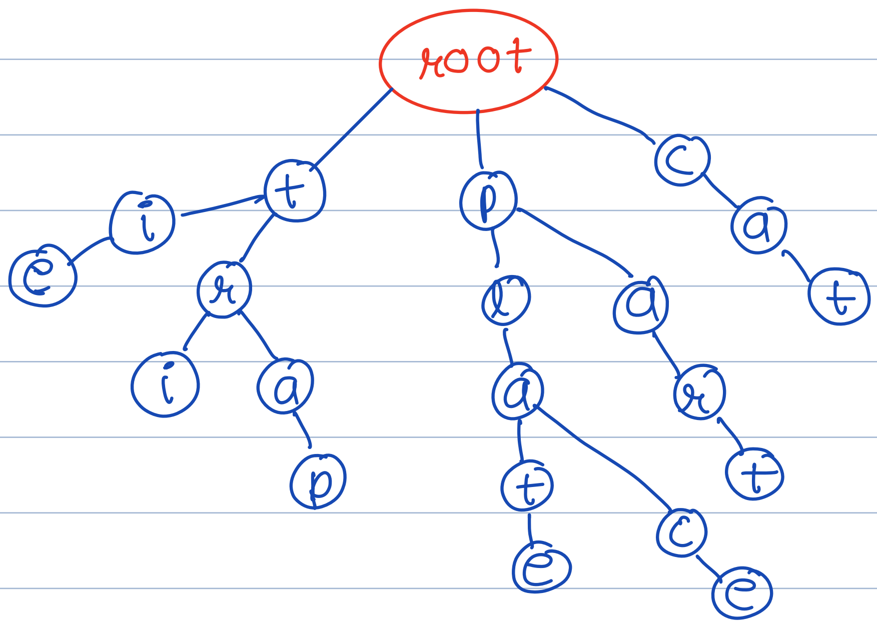
NOTE: Assume no word is a prefix of another word

ie, the representation is always possible.

$$\begin{tikzcd}
\text{words} \arrow[r] & \text{tri} \arrow[d] & \text{trap} \arrow[d] & \text{plate} \arrow[d] & \text{cat} \arrow[d] & \text{part} \arrow[d] & \text{place} \arrow[d] & \text{tie} \arrow[d] \\
 & \text{tri} & \text{tra} & \text{plat} & \text{c} & \text{pa} & \text{plac} & \text{ti}
\end{tikzcd}$$

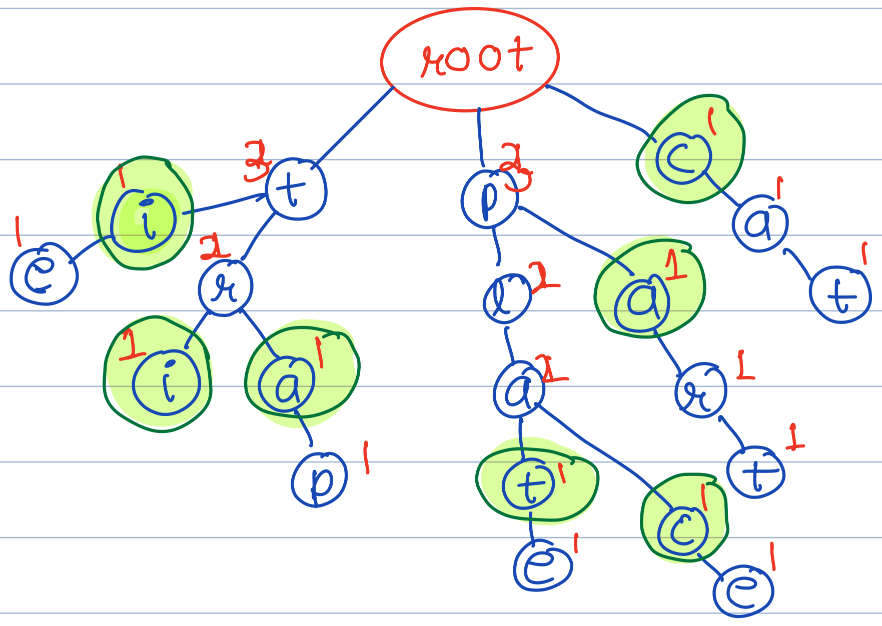
$$\begin{tikzcd}
\text{words} \arrow[r] & \text{zebra} \arrow[d] & \text{dog} \arrow[d] & \text{duck} \arrow[d] & \text{dove} \arrow[d] \\
 & \text{z} & \text{dog} & \text{du} & \text{dov}
\end{tikzcd}$$

tri trap plate cat part place tie



Idea 1> Same as deletion in a trie, keep track of last node that cannot be deleted.

Idea 2> tri trap plate cat part place tie



```
Class TrieNode {
    TrieNode [] children
    bool eow
    freq
    TrieNode () {
        freq = 0
        chidren = new TrieNode [26]
        eow = false
    }
}
```

N words with max word length as L

TC : O(N*L)

SC : O(N*L)