Tric.

① Spell - checker

playgriund.

loyal

life

lyfe.

word → l_i

Collection of valid words.

[N]

T.C → O(N*l)

↓

HashMap / Hashset.

② Auto. complete.

play

place

plate

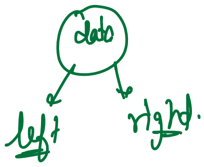
playground.

Tric. (prefix-tree)

→ hierarchical data structure.

→ N - children tree
(ary)

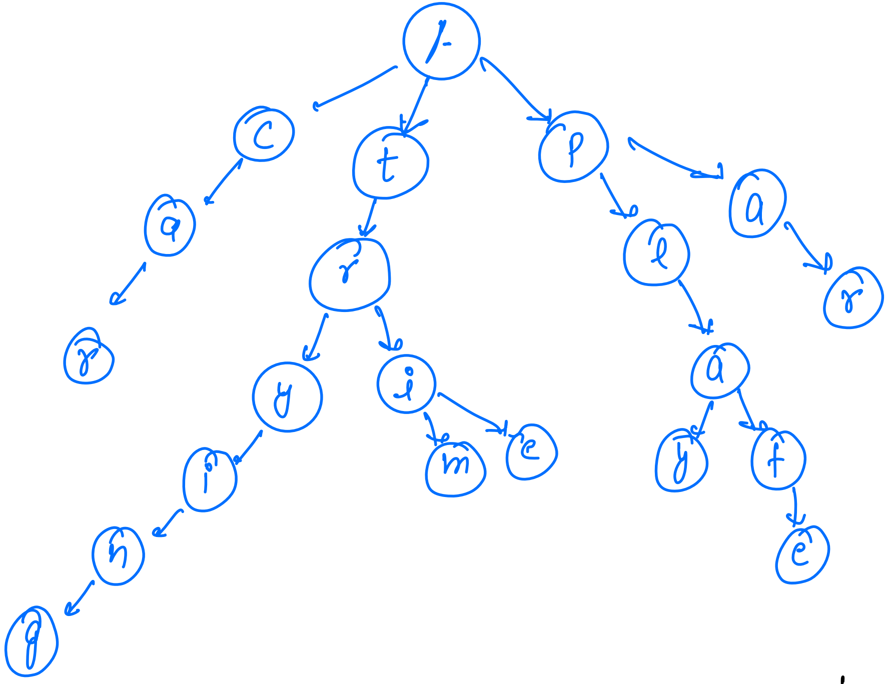
→ It is used for information retrieval.



→ It is a data-structure which stores the information from top to down.

dict →

try trim trie play trying
plate car par ~~trimmer~~ pla

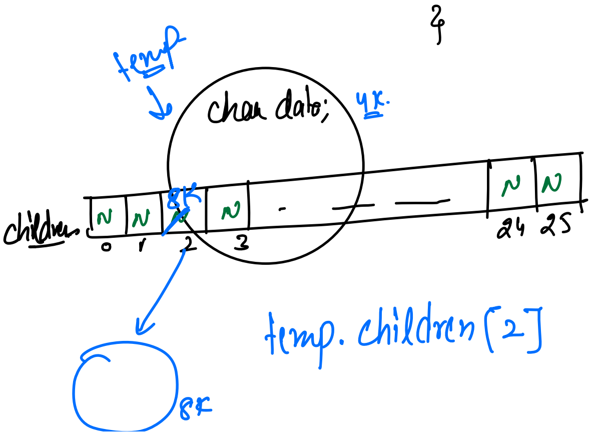


```
Class Node {
    char data;
    Node a;
    Node b;
    Node c;
      |
      |
    Node z;
}
```

⇐

```
Class Node {
    char data;
    Node children[26]

}
```

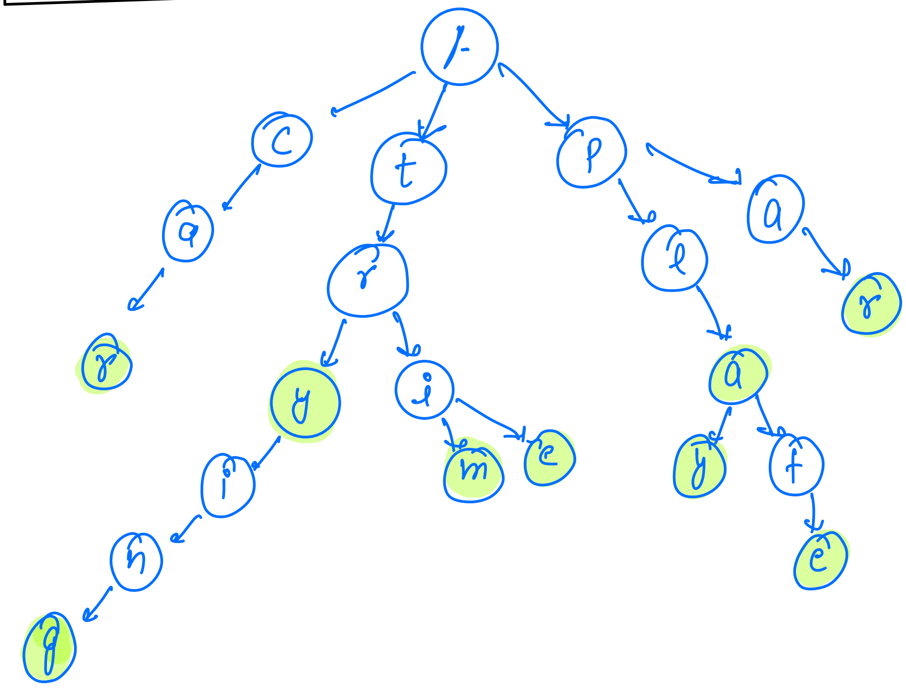


temp.children[2] = new Node('c');

a → '0'
b → '1'
c → '2'
'd' → '3'
|
'z' → '25'

dict →

try	trim	trie	play	trying
plate	car	par	~~trimmer~~	pla



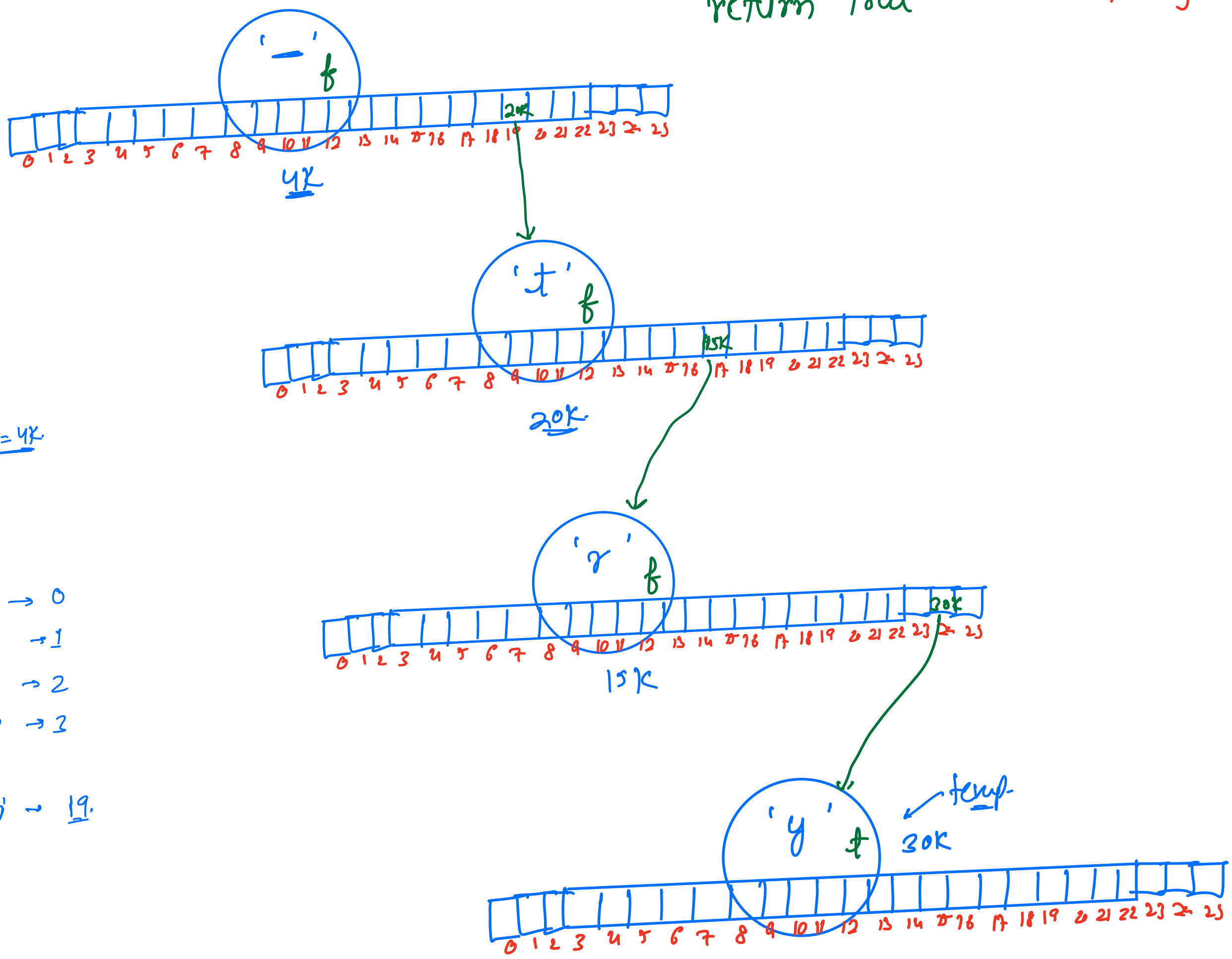
search(trie)
search(tri)
search(try)
search(tr)

There should be a marker whether the current node is denoting the end of word or not.

```
class Node {
        char data;
        Node children[26]
        boolean isEnd;
}
```

traverse on the string and for every character, we need to traverse trie.

- if any character of string is not found in trie.
  ⇓
  return false
- all characters are there
  - last char of the string.
    ch.isEnd == true
    return true
  - last char of the string
    ch.isEnd == false
    return false;



temp = 4K

try.

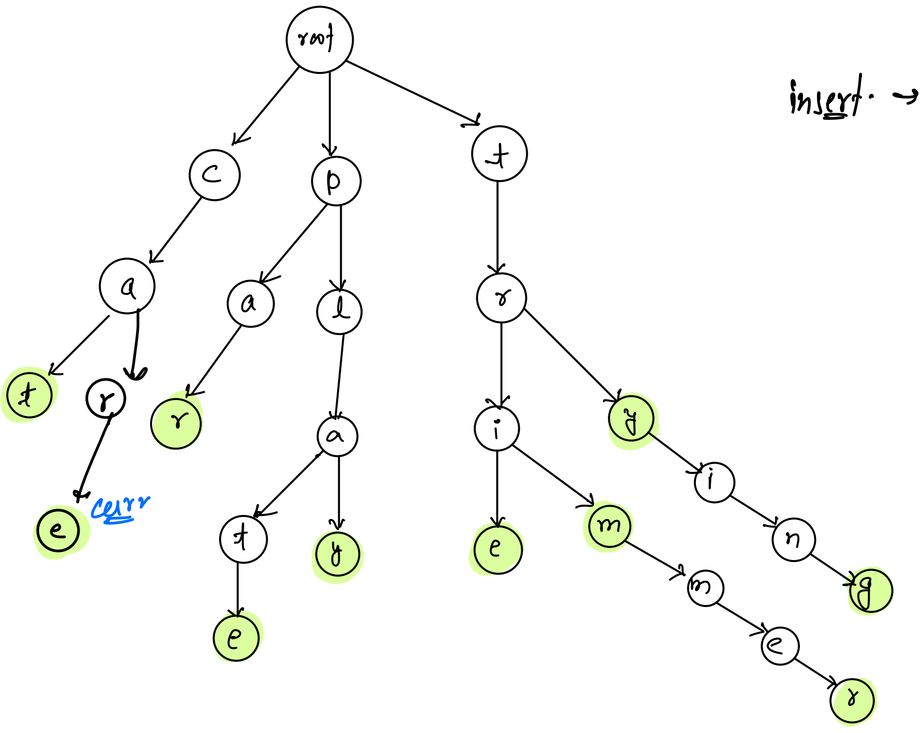
'a' - 'a' → 0
'b' - 'a' → 1
'c' - 'a' → 2
'd' - 'a' → 3

't' - 'a' → 19.

insert

```
void insert( Node root , String word ){
    Node curr = root ;
    for( i=0; i< word.length(); i++){
        idx = word[i] - 'a' ;
        if ( curr. children[idx] == NULL){
            curr.children[idx] = new Node ( word[i]) ;
        }
        curr = curr. children[idx]
    }
    curr. isEnd = true ;
}
```

{ T.C → O(L)
S.C → O(L) }



insert → care.

Search.

```
boolean search ( Node root , String word ){
        Node curr = root ;
        for( i=0; i< word.length(); i++){
                idx = word[i] - 'a' ;
            if ( curr.children[idx] == NULL){
                    return false;
            }
            curr = curr.children[idx]
        }
        return curr.isEnd ;
}
```

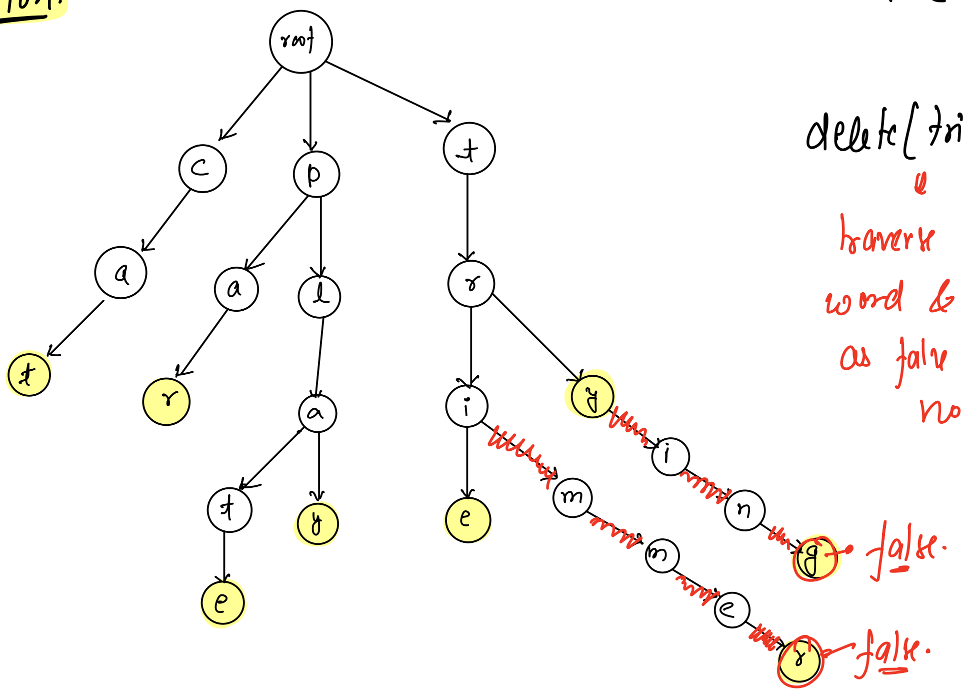
T.C → O(l)

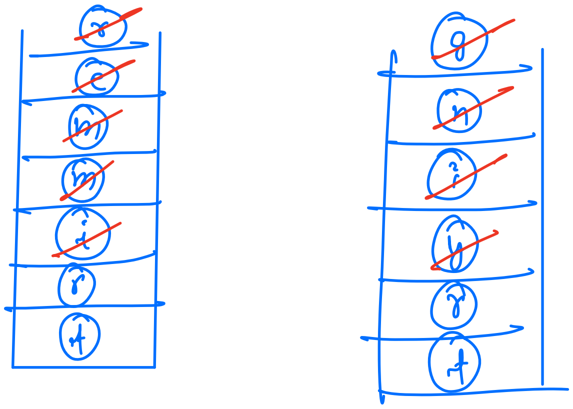
deletion.

delete(trying)

delete(trimmer)

traverse for current word & mark isEnd as false for the last node.





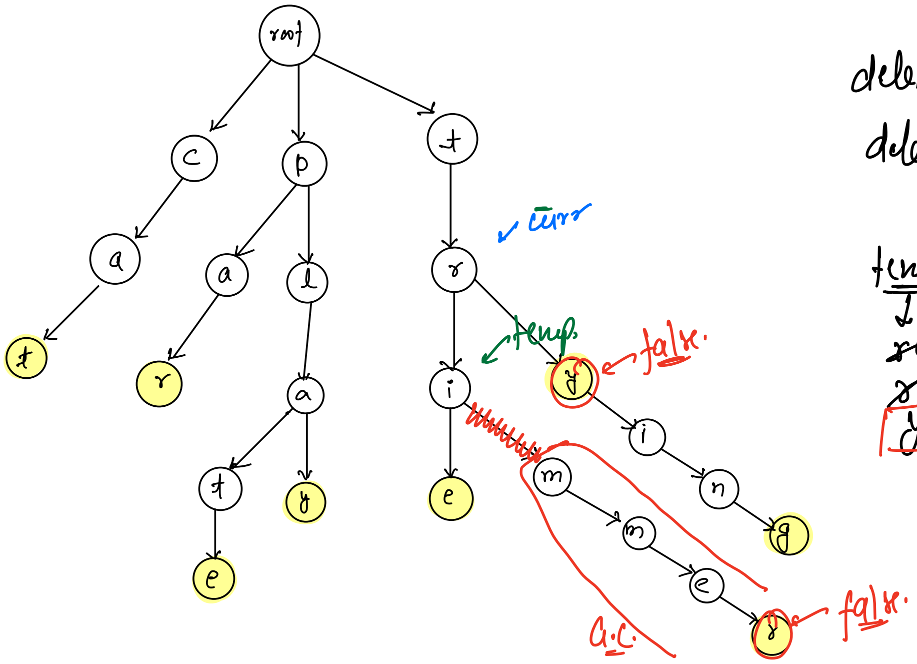
{ T.C → O(l) }
{ S.C → O(l) }

nodes that can't be deleted.

→ nodes where isEnd is marked as true.

→ nodes which are having > 1 children.

idea. → find the last node that can't be deleted.



delete(trimmer)

delete (try)

temp
↓
~~root~~
~~r~~
$\boxed{y}$

nextchar
↓
~~t~~
~~y~~
-

last node that can't be deleted.

~~root~~

~~r~~ i

temp.children [m - r] = NULL;

pseudo-code.

```
void DeleteWord(Node root, String word){
    Node curr = root, temp = null, nextChar = '-'
    for( i=0; i< word.length() - 1; i++){
        int count = 0;
        for( i = 0; i<25; i++){
            if ( curr.children[i] != NULL){
                count++
            }
        }
        if (count > 1 || curr.isEnd == true){
            //we can't delete this node
            temp = curr, nextChar = word[i]
        }
        idx = word[i] - 'a';
        curr = curr.children[idx] ;
    }
    idx = word[N-1] - 'a';          //'y'-'a'
    curr = curr.children[idx];
    curr.isEnd = false;
    int count = 0;
    for( i = 0; i<25; i++){
        if ( curr.children[i] != NULL){
            count++
        }
    }
    if (count != 0) { // do nothing }
    else{
        temp.children[nextChar - 'a'] = NULL;
    }
}
```

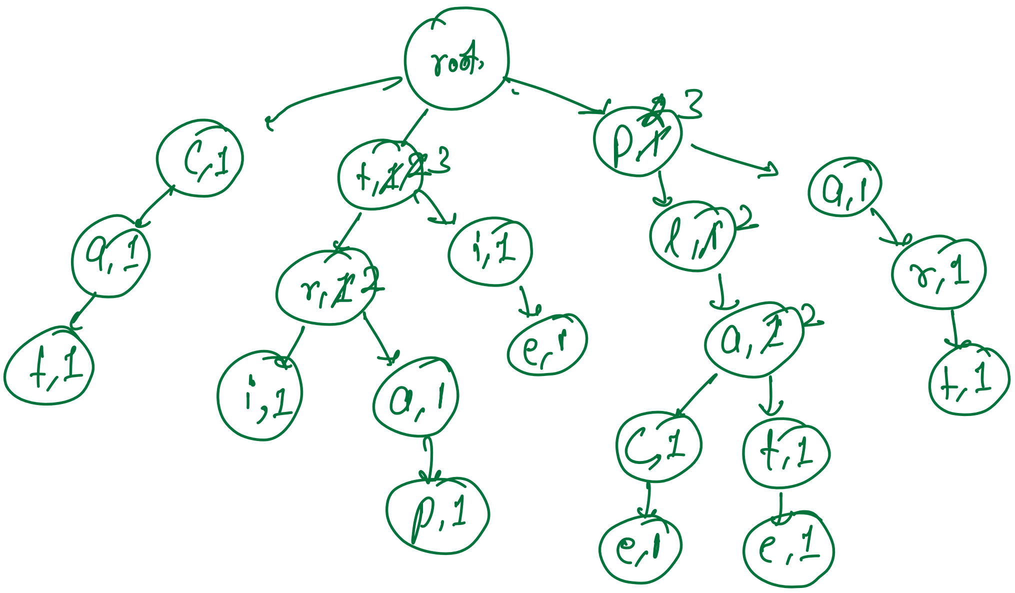
Q.1 Find shortest unique prefix to represent each word.

Note: Assume that no word is prefix of another word. In other words, the represtation is always possible.

tri	trap	plate	cat	part	place	tie
tri	tra	plat	c	pa	plac	ti

~~trip~~ ✗

trie.

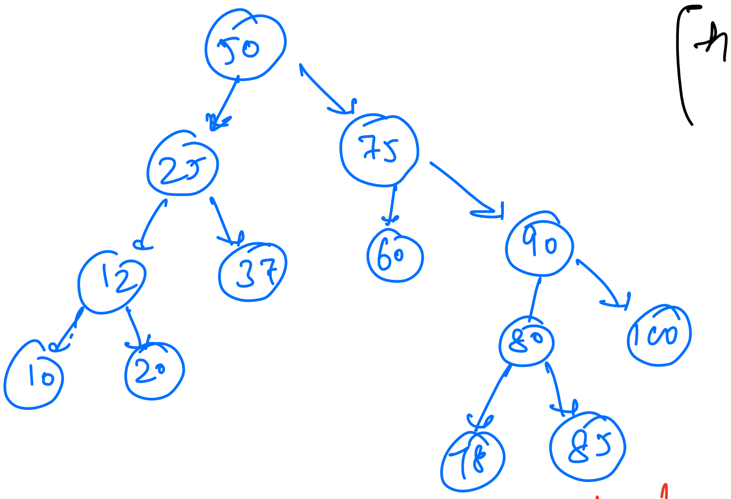


- Small variation { including frequencies of characters }

```
Node {
      char data;
      Node children[26];
       int freq;
}
```

$|\text{height of l.s.t} - \text{height of r.s.t}| \leq 1 \Rightarrow$

[height balanced binary tree]



→ false.

boolean isHb = true;

```
int height (Node root){
    if (root == NULL){ return -1 }
    lh = height(root.left);
    rh = height(root.right);
    if (|lh-rh| > 1 ){ isHb = false }

    return max(lh, rh)+1
}
```