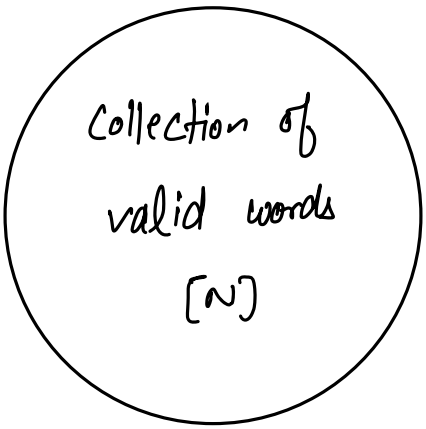
① Spell-checker

playgriund
loyal
friend
lyfe

length of word → l



HashSet<String>

T.C → O(l)

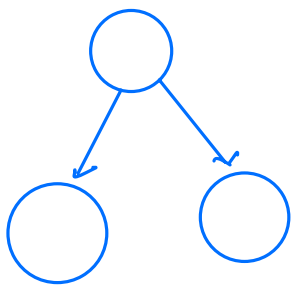
because of hashcode.

Tric:

- hierarchical data structure
- pre-fix tree
- It is used for information retrieval.

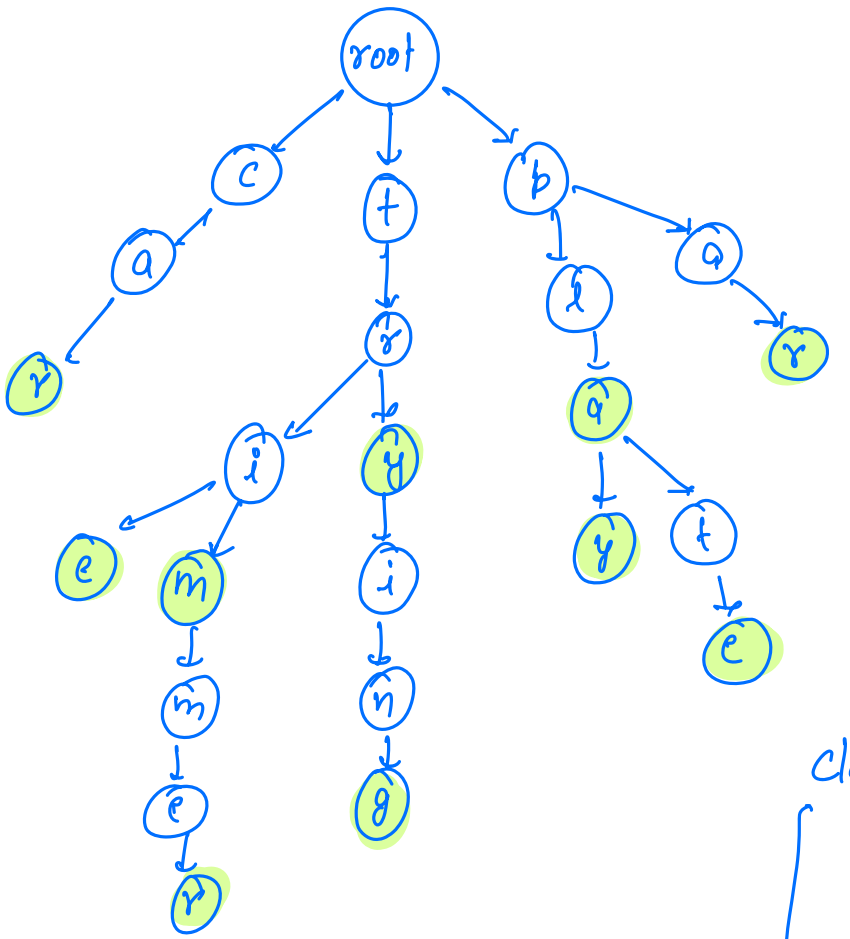
∴ It is a data structure which stores the information from top to down.

```
class Node{
    int val;
    Node [2] children;
```

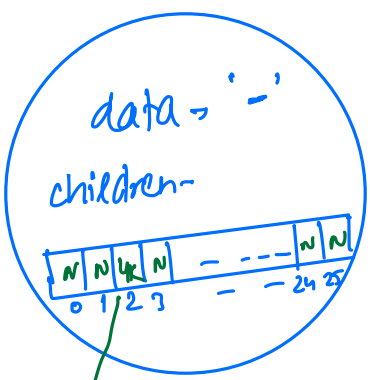


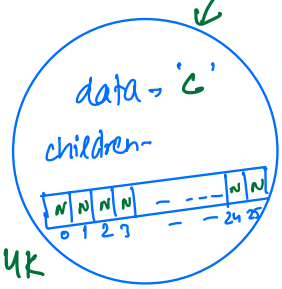
dict →

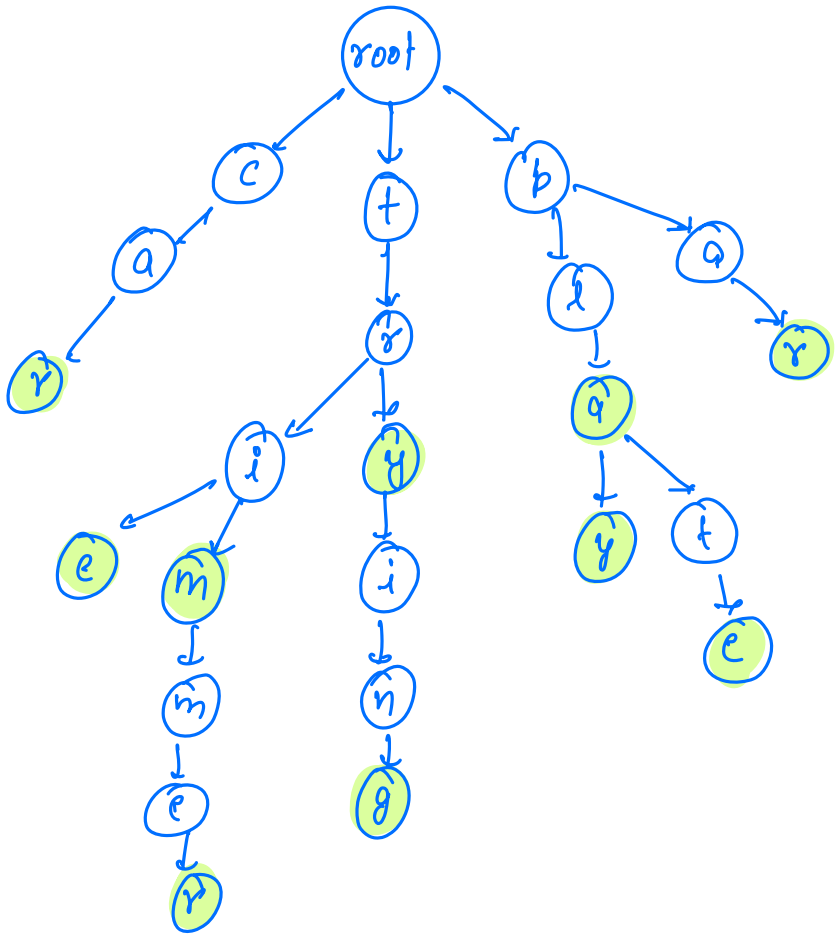
try	trim	trie	play	trying
plate	car	par	trimmer	pla




```
class Node {
    char val;
    Node children[26];
    public Node (v) {
        val = v;
        children = new Node[26];
    }
}
```







search(trie) → ✓

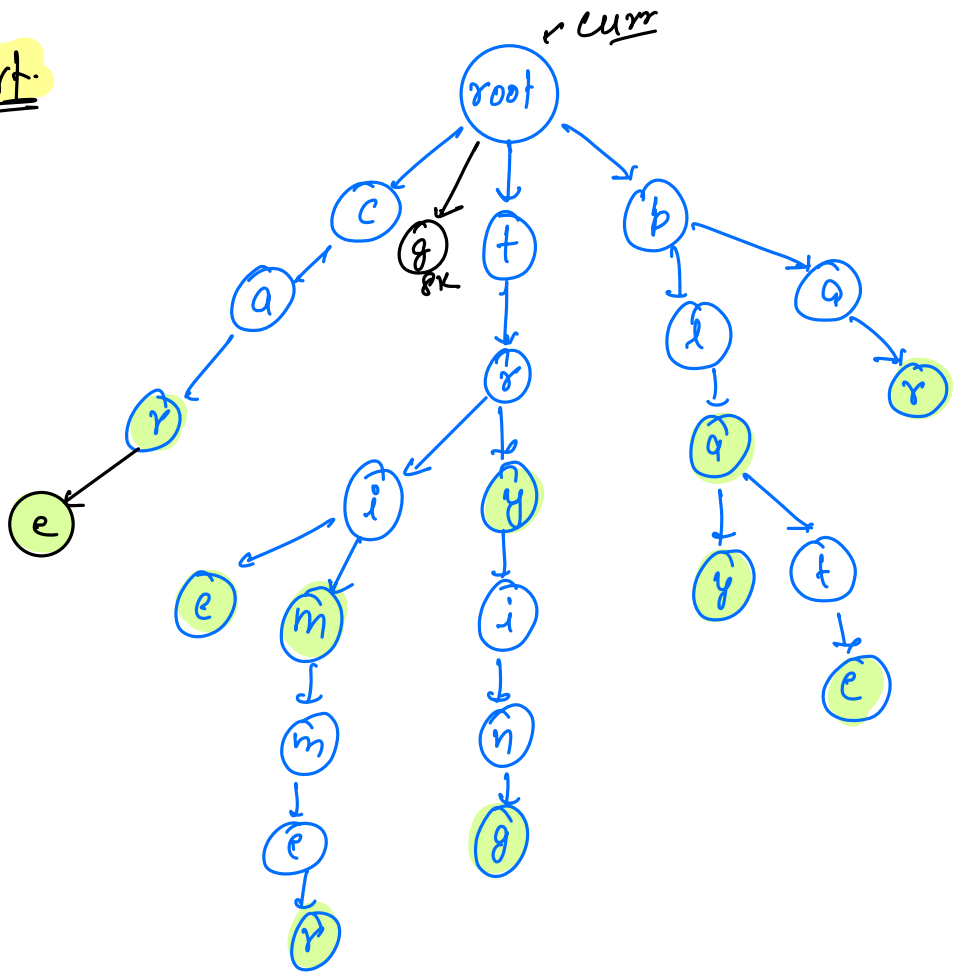
search(tri) → ✗

search(trim) → ✓

∴ There must be a marker to denote whether the current node is end of the word or not.

```
class Node {
    char val;
    Node children[26];
    boolean eow;
    public Node (v){
        val = v;
        children = new Node[26];
        eow = false;
    }
}
```

insert.



insert('care')

a -'a'→ 0
b -'a'→ 1
c -'a'→ 2
d -'a'→ 3
e -'a'→ 4
f -'a'→ 5
z -'a'→ 25

word → get

code.

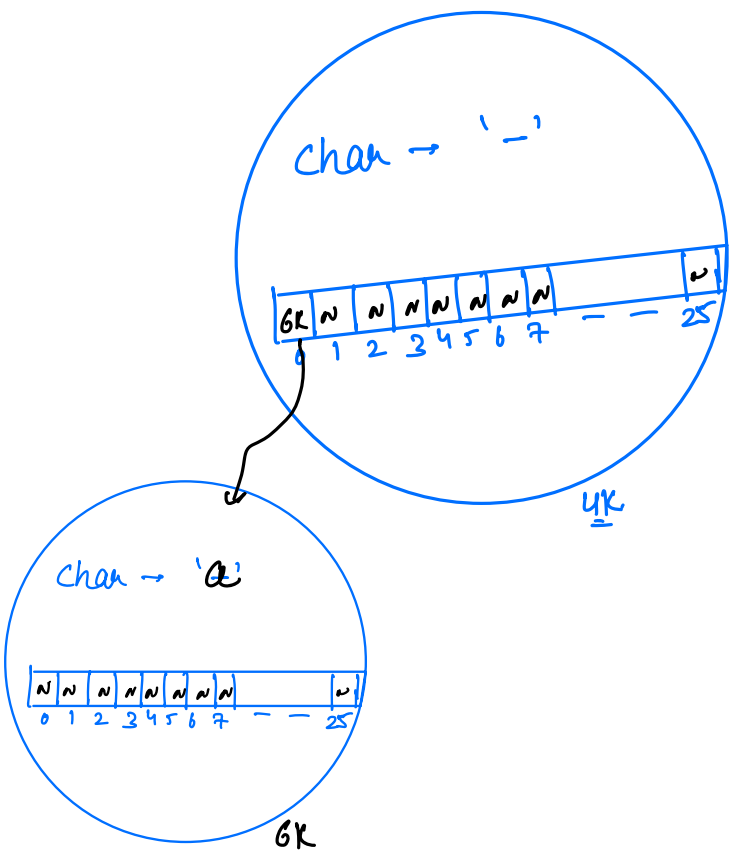
```
void insert(Node root, String word){
    Node curr = root;
    for(i = 0; i < len(word) ; i++){
        ch = word[i];
        if( curr.children[ch - 'a'] == NULL){
            curr.children[ch - 'a'] = new Node(ch);
        }
        curr = curr.children[ch - 'a']
    }
    curr.eow = true;
}
```

T.C → O(l)
S.C → O(l)

Search.

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

[T.C → O(L)
S.C → O(1)]



abc

root → 4K

Size → 5

words → 100

Size of trie?

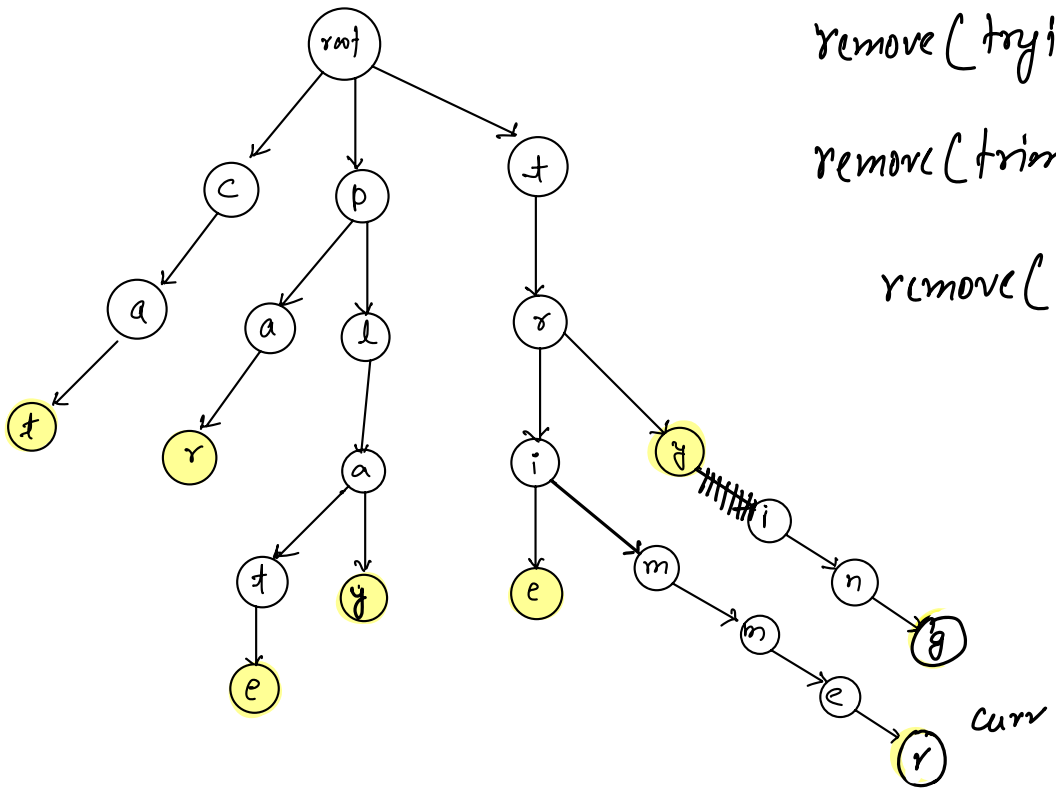
→ pneumonoultramicroscopisilicovolcanoconiosis

44

max-levels possible → 44

Max possible size of trie → 26^{44}

deletion.



remove ("trying")

remove ("trimmer")

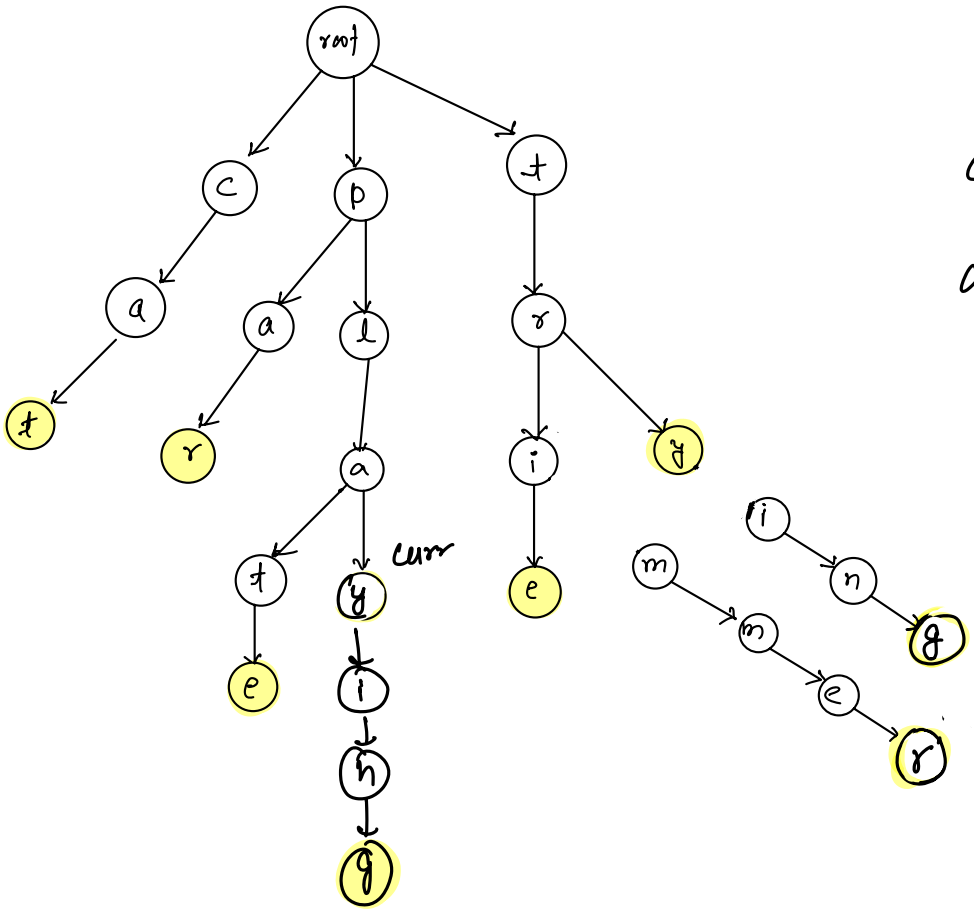
remove ("try")

nodes that can't be removed from trie.

→ nodes where eow is marked as true.

→ nodes which have more than one children.

[if word that we want to remove is a prefix for another word, then we can't remove any nodes.]



delete (trimmer) ✓

delete (trying) ✓

delete (play)

temp.

~~NULL~~

~~root~~

~~b~~

q

nc.

~~p~~

~~l~~

y

temp.children[nc - 'a'] = NULL;

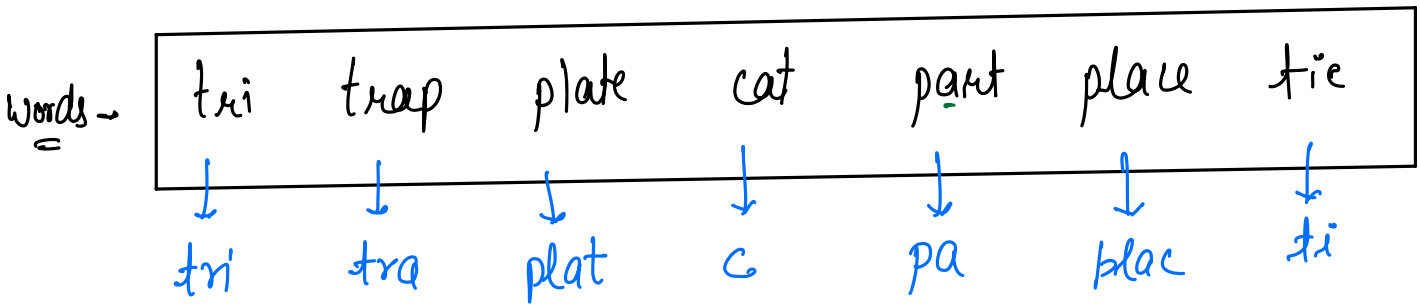
pseudo-code.

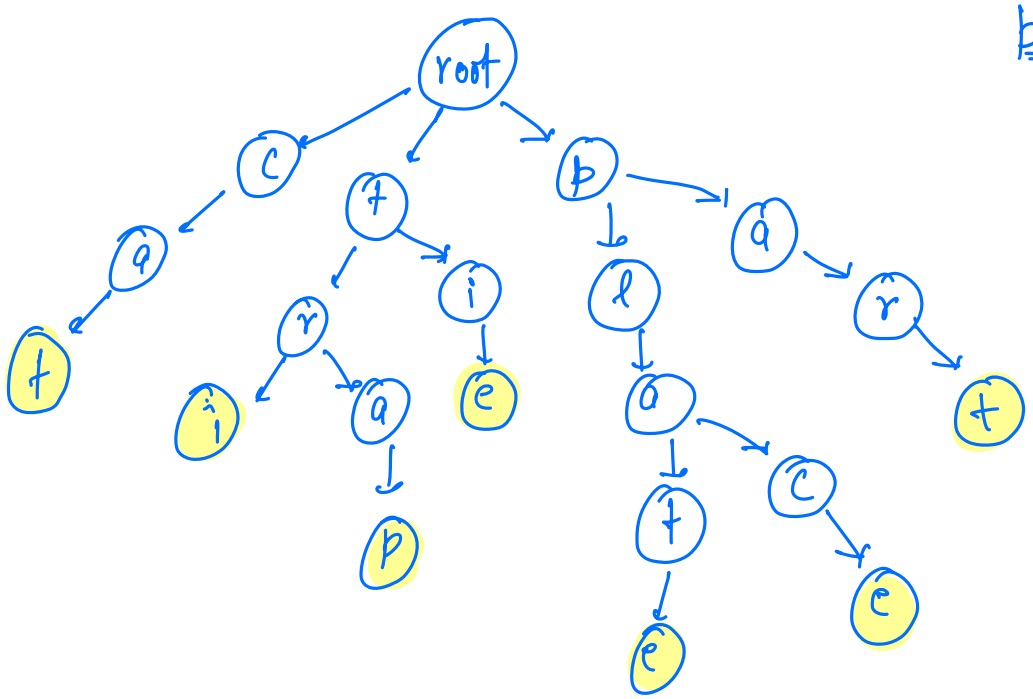
```
void deleteNode ( Node root, String word) {

    Node curr = root, temp = NULL, nc → '-' ;

    for( i = 0; i < len(word); i++){

        //count = 0 ;        //no. of children of curr node

        for( j = 0; j < 25 ; j++){
            if ( curr.children[j] != NULL){
                count ++;
            }
        }

        if ( count > 1  ||  curr.eow == true ){
            temp = curr , nc = word[i];
        }

        curr = curr.children[word[i] - 'a'];
    }

    curr.eow = false;

    count = 0
    for( j = 0; j < 25 ; j++){
        if ( curr.children[j] != NULL){
            count ++;
        }
    }

    if (count > 0)  { return }
    else    { temp.children[ nc - 'a'] = NULL ; }

}
```

T.C → O(l)
S.C → O(1)

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

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



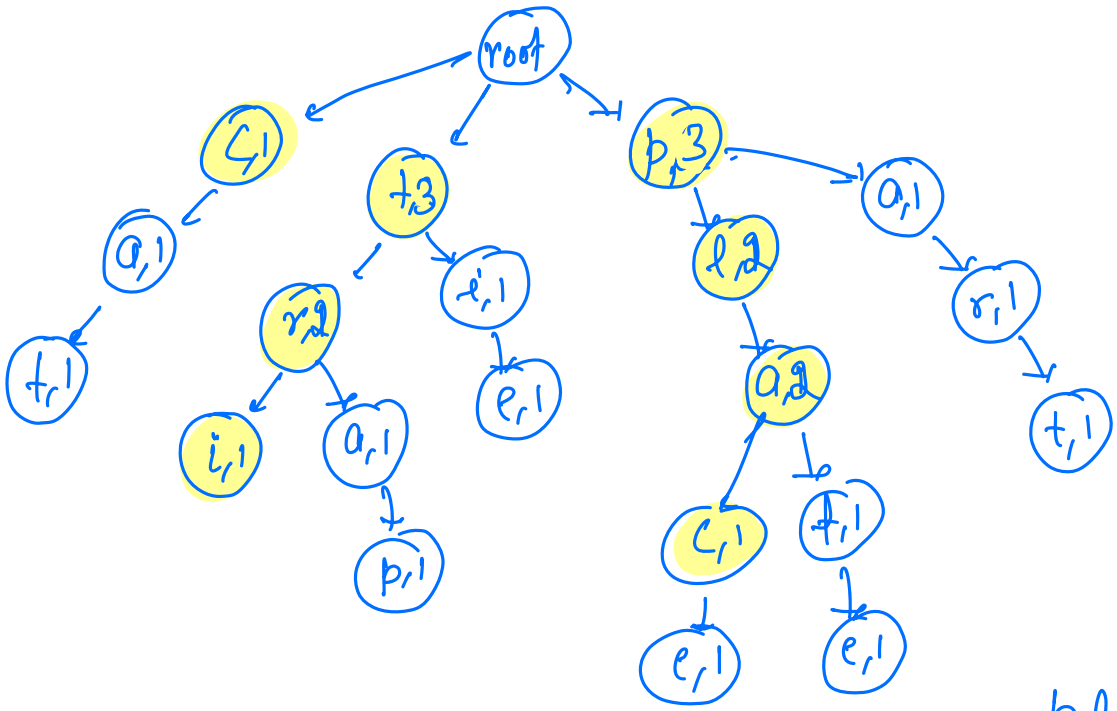


place

- first/last node with single child. X

Words - tri trap plate cat part place tie

tri tra plat c pa plac ti

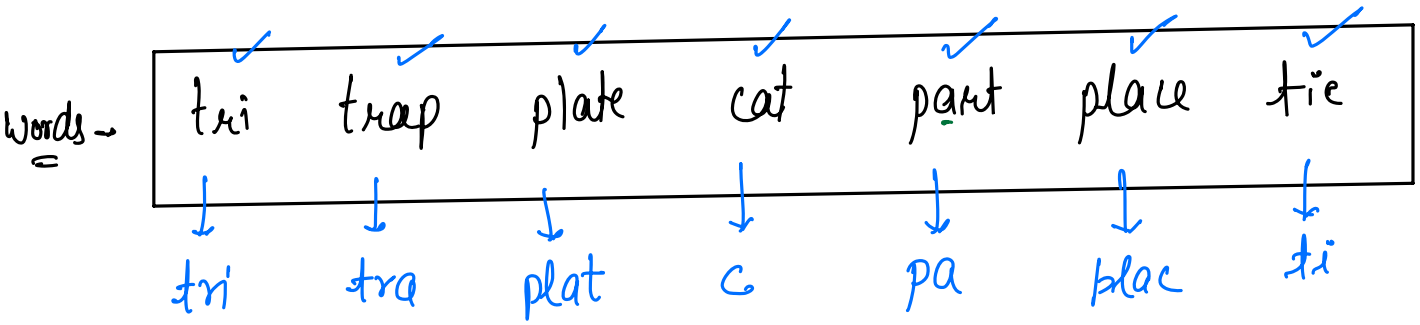


place

plac.

[For every word, stop when you find a character with freq = 1

#code → todo



① psudo code - Not the exact code.

Code

① Code Readability ② Stable code

Generic Tree - N-Ary Tree

```
class Node {
    int data;
    List<Node> children;
}
```

pre-Order Traversal

```
void pre-order( TreeNode node){
    print(node.val);
    for( Node child : node.children){
        pre.Order(child);
    }
}
```