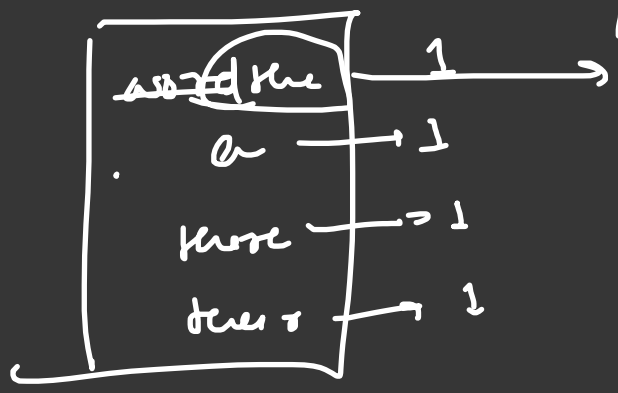
Trie

words[] = { "the", "a", "there", "their", "any" }

Q. We have to search the word "the" in the word array?

→ Hash map



word → T.C.

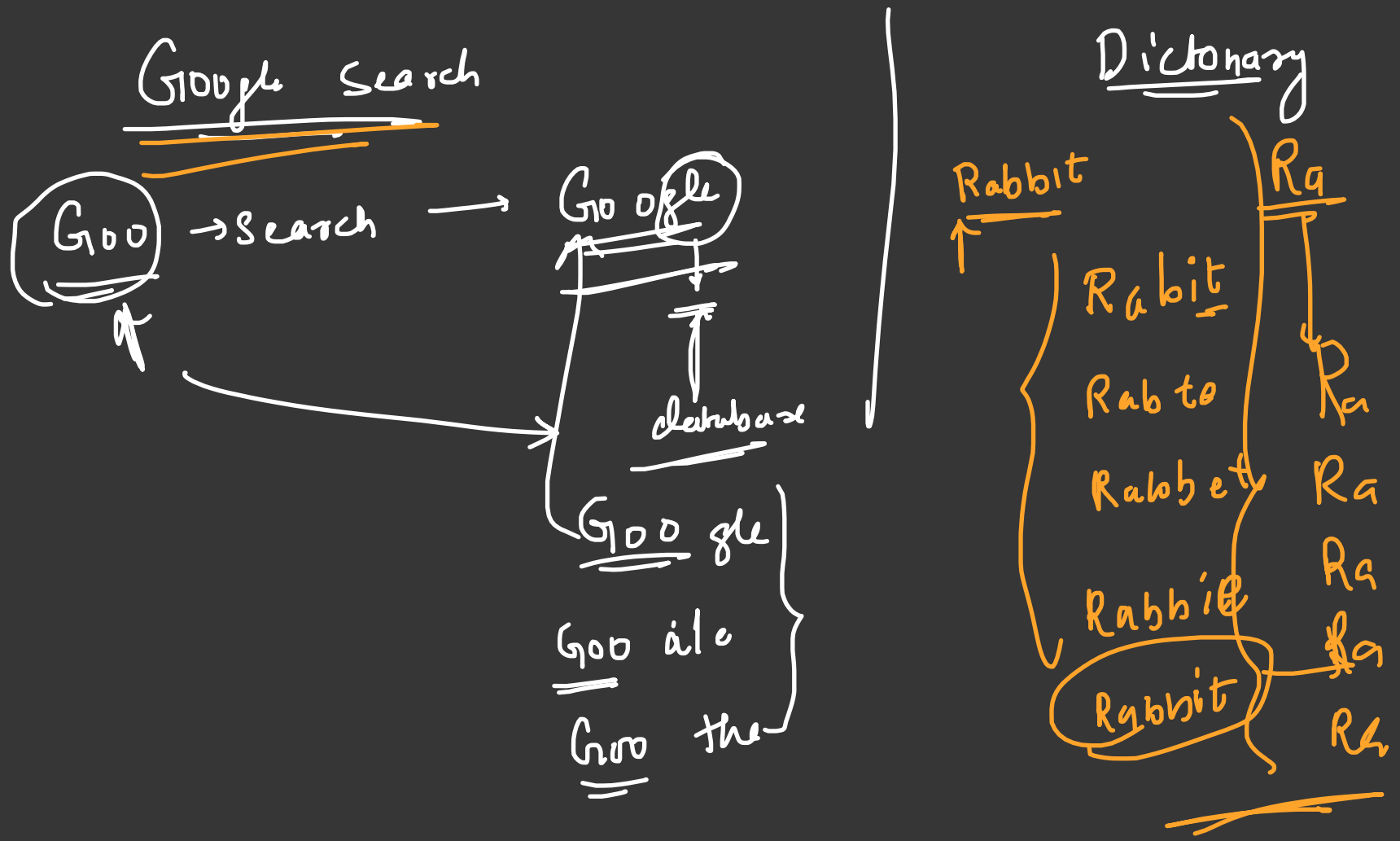
O(1) → Avg. case.

Q. Count / Does any word start with "the" exist in array?

- the → prefix there exist 3 word with "the" prefix

the, their, there

Eg:

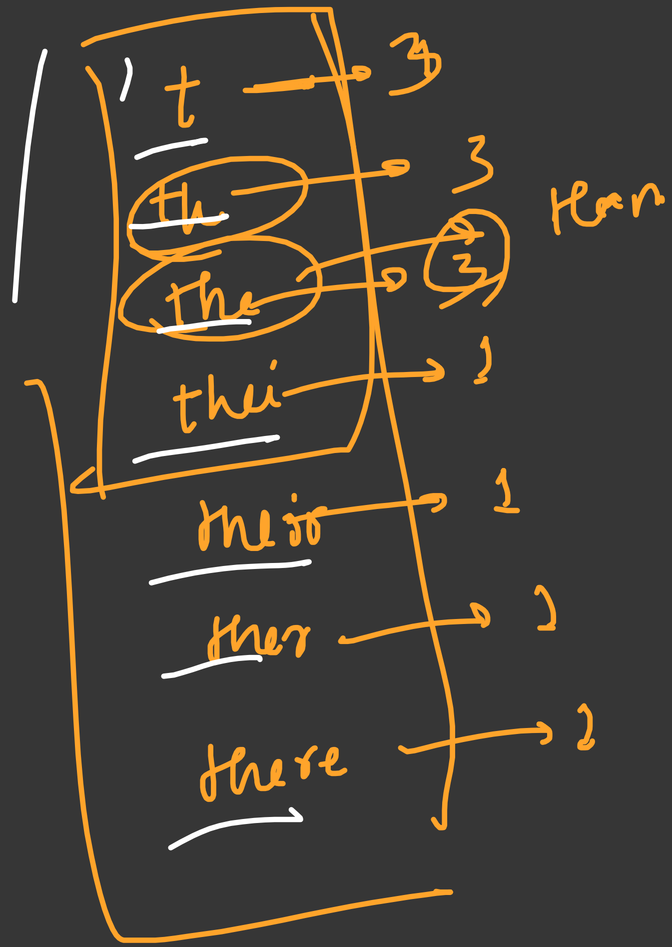

Google Search
Goo
→ Search
Google
database
Google
Google ale
Goo the
Dictionary
Rabbit
Ra
Rabit
Rab to
Rabbet
Rabbit
Rabbit
Ra
Ra
Ra
Ra
Ra

Hashmap

N { the, their, there }

Sol^n ✓ one type of sol^n

Where is the problem?



When we are storing word in Map

$\Theta(t)$ →

$O(l)$ →

{abcdefgh . . . .} → $O(l)$

+ Space. high space → which not efficient

2. Trie → Come in play

Trie → Tree

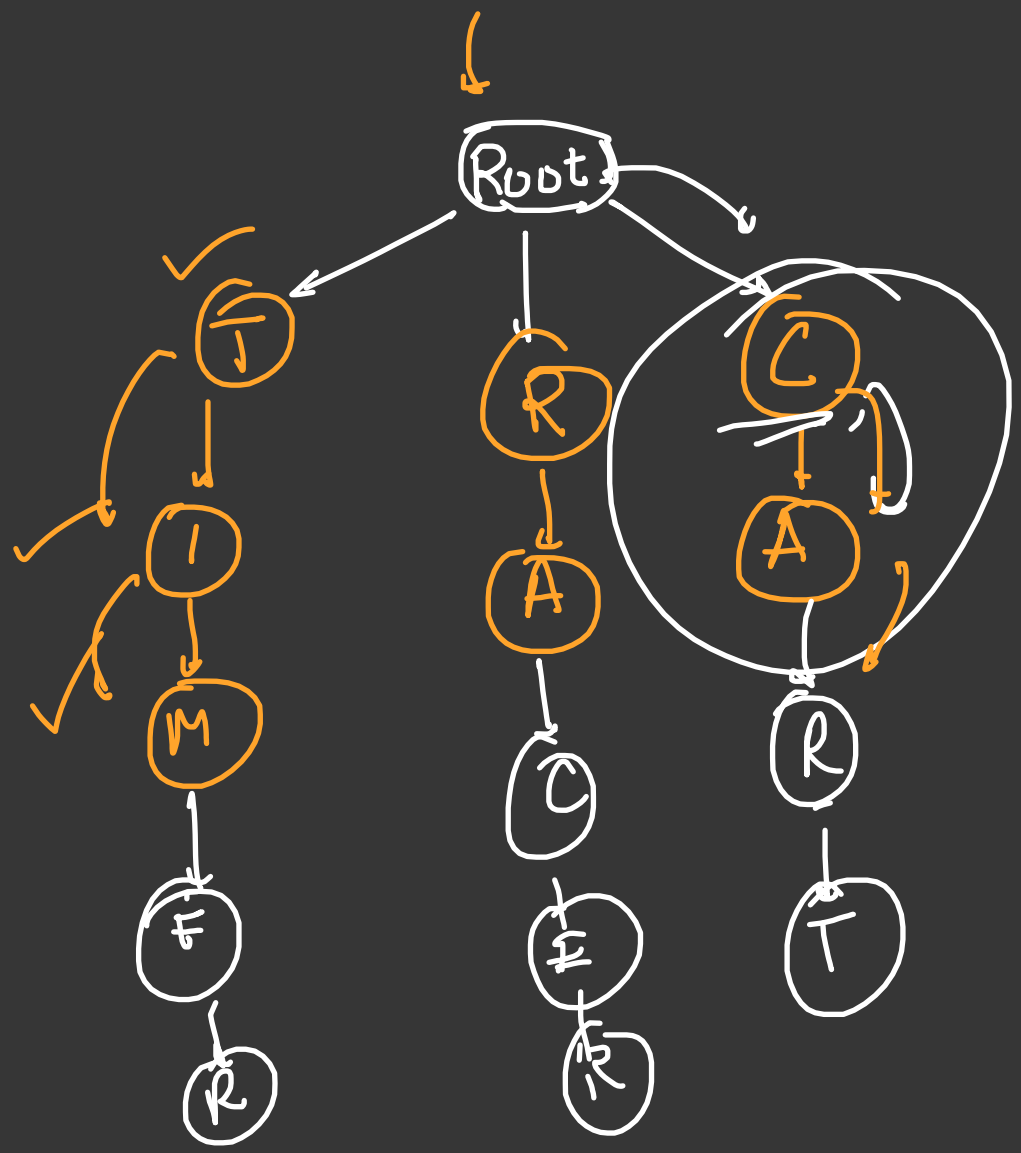
Retrieval Tree

Prefix Tree

→ Tree data structure

• Stores String → String → Searches
→ prefix
→ Large String and pattern

Trie



TIMER

RACE

CA →

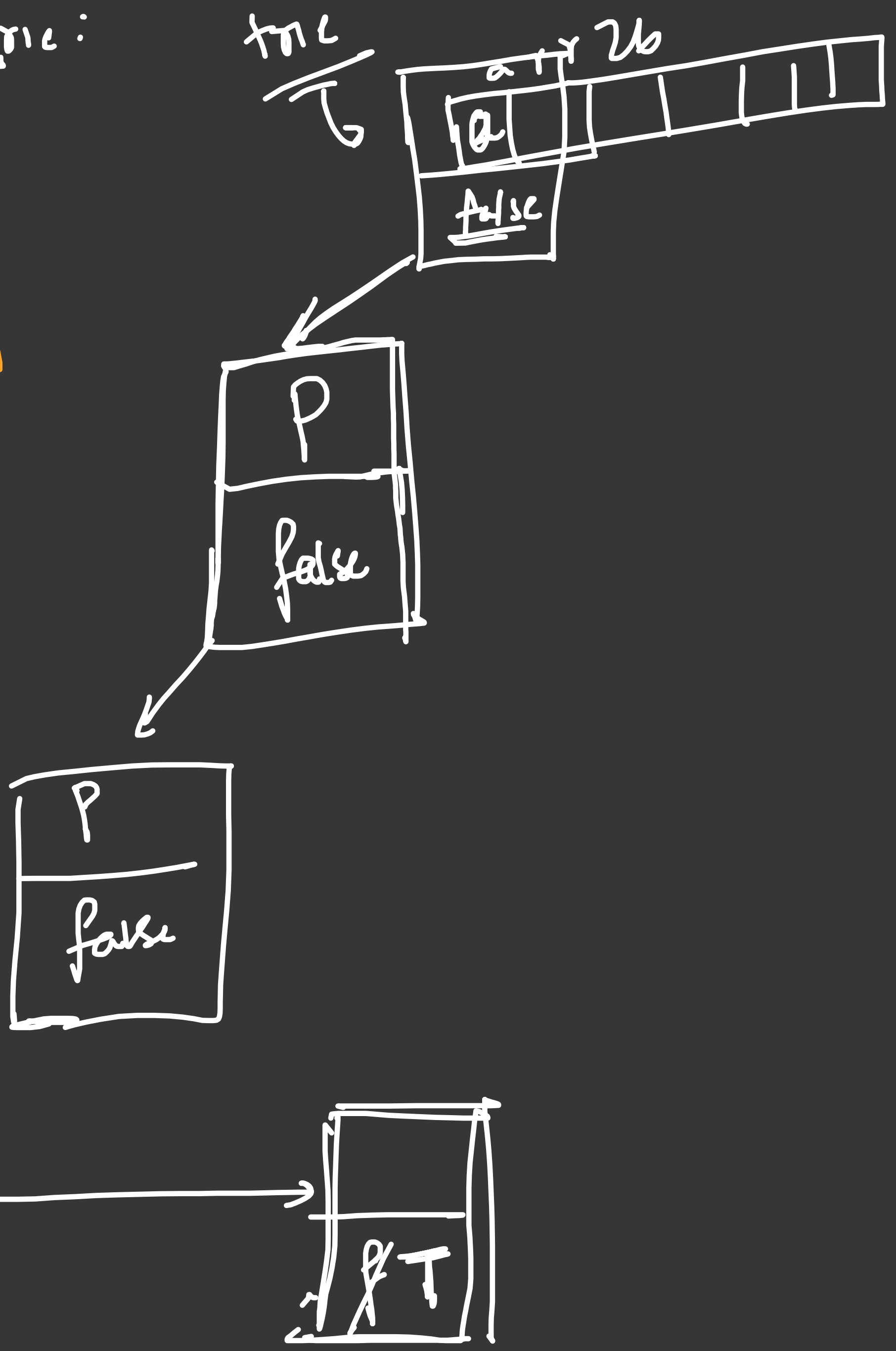
CAR

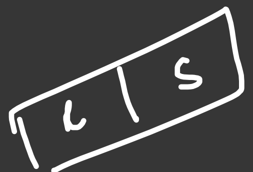
T.C. : O(L)

S.C. : O(N)

Structure of Trie:

```
trie {
    char a[26];
    bool flag;
}
```





apple

apps

applk

appel

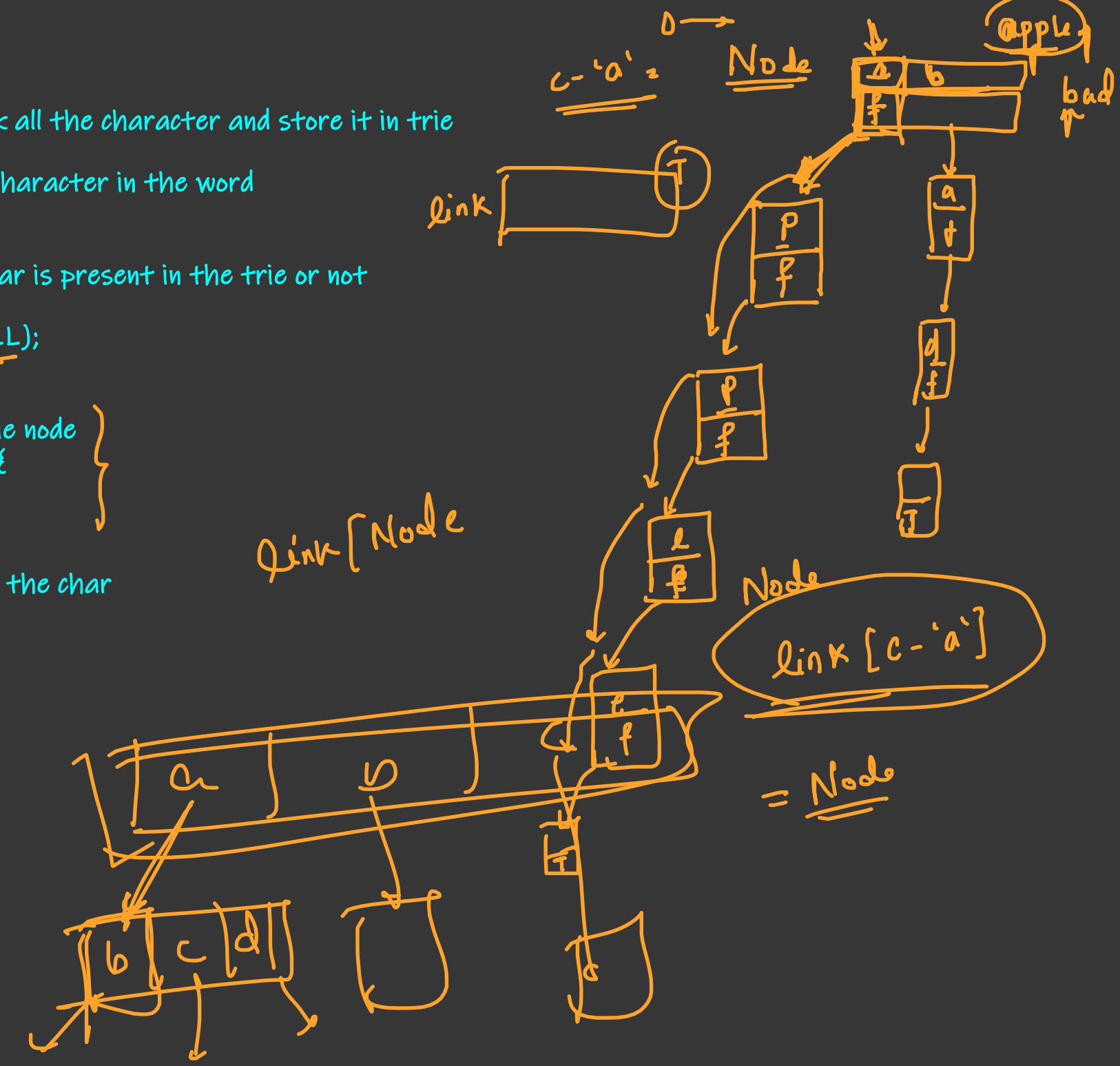
appeal

0 → a

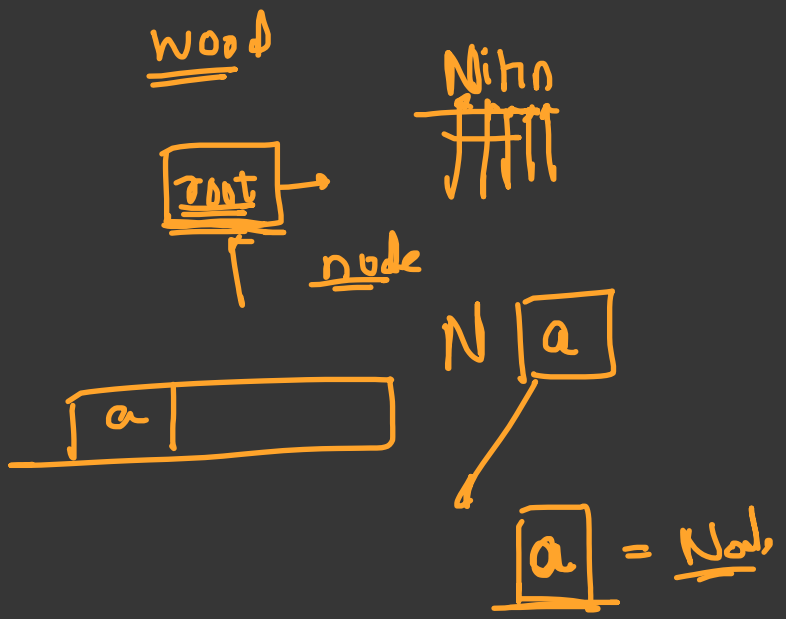
⋮

25 → z

```
struct Node{
    // Node base array which link all the character and store it in trie
    Node* link[26];
    //To keep the track of the character in the word
    bool flag = false;

    //Function to check if the char is present in the trie or not
    bool contain(char c){
        return (link[c - 'a'] != NULL);
    }

    //putting the char in the trie node
    void put(char c, Node* node){
        link[c-'a'] = node;
    }

    //give the trie reference for the char
    Node* get(char c){
        return link[c - 'a'];
    }

    //set the end of the word
    void setEnd(){
        flag = true;
    }

    //check if the char is end
    bool isEnd(){
        return flag;
    }
};
```



```
class Trie{
    //We always start with the root node
    private:
        Node * root;

    public:
    //Construct the node for the trie
    Trie() {
        root = new Node();
    }

    //Insert
    //TC: O(L)
    void insert(string word){
        //We create a dummy node to track the trie
        Node* node = root;
        for(int i=0; i<word.size(); i++){
            //There are things to keep in mind that we need check if there already exist the word char or not
            //so for that we check if the char already exists in trie
            //This if statement check if the char is not present in the trie
            if(!node->contain(word[i])){
                //so we create a reference trie for the char
                //We are refernce this char to new trie and inserted in the trie
                node->put(word[i],new Node());
            }
            //now we have to move to new reference trie so
            node = node->get(word[i]);
        }
        //so we have reach the end of the word we set the end of trie
        node->setEnd();
    }

    //Search
    bool searchWord(string word){
        Node* node = root;
        for(int i=0; i<word.size(); i++){
            if(!node->contain(word[i])){
                return false;
            }
            node = node->get(word[i]);
        }
        return node->isEnd();
    }
```



node → get(

```
bool startWith(string prefix){
    Node* node = root;
    for(int i=0; i<prefix.size(); i++){
        if(!node->contain(prefix[i])){
            return false;
        }
        node = node->get(prefix[i]);
    }
    return true;
}
```

Prefix

word, ...

Key : Appl

app

5
apple
apple
apple
apps
apps

Trie-II

Search → Tries

Count the word

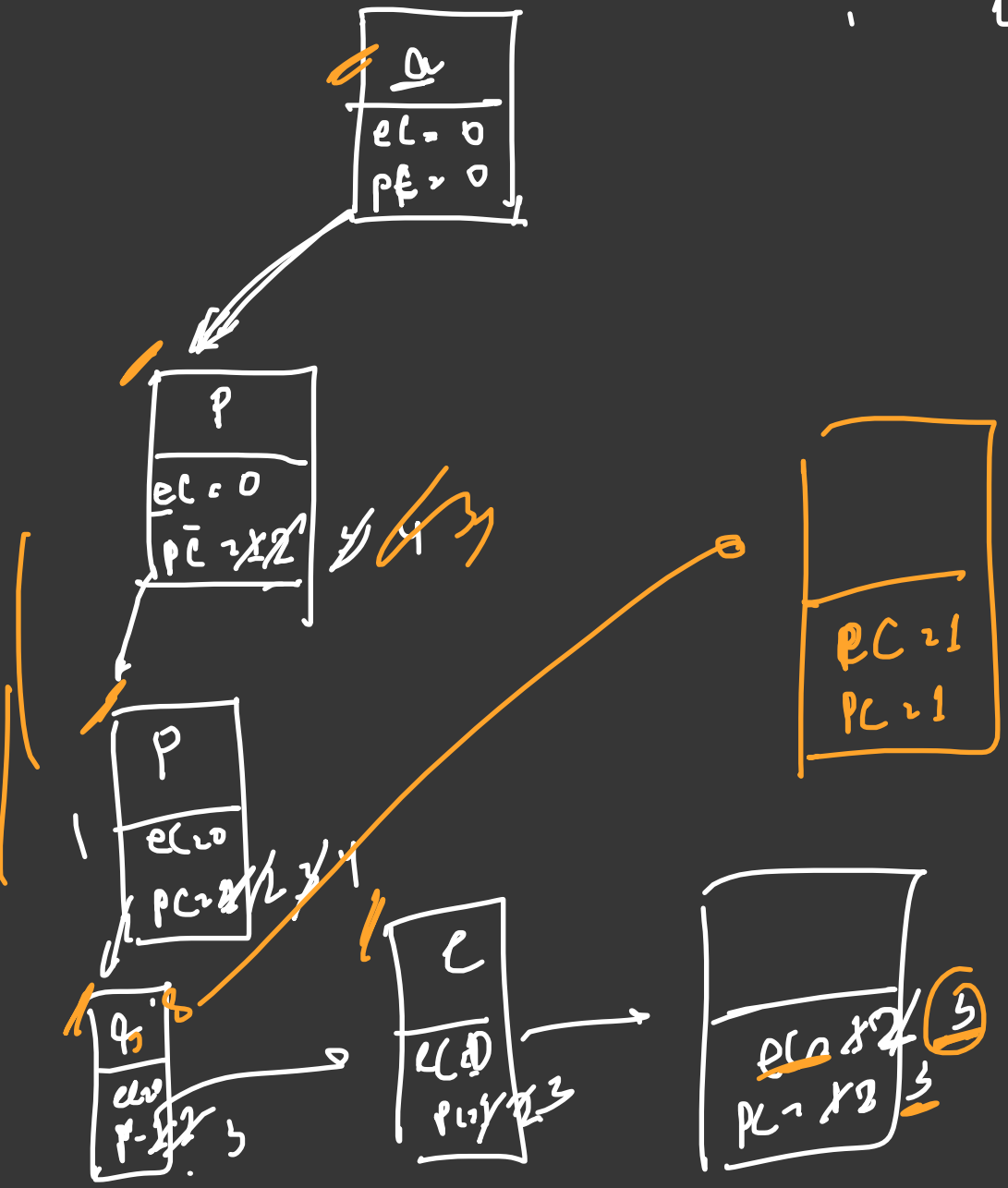
Count the prefix

Erase

Insert ap

Initialize root

Link [4]
endNum = 0
prefix = 0

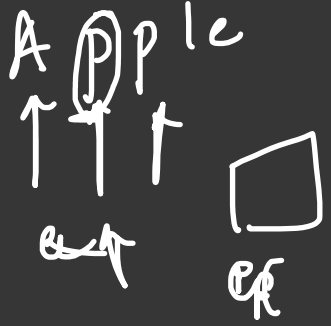


Structure Trie

Struct Trie {

Node* Link [26]
endCount
prefix

a
ap
app
appl
apple
a



Count the word

apple → 3

Count the prefix →

Erase → word