Q: Given N strings & Q queries. For each query check if it's present in N strings.

Constraints :-

- All characters are ['a' - 'z']
- Length of each string is from [1, l]

  $1 <= \text{length} <= l$

(N) Words	(Q) Query	
dcmp	data	✓
dark	take	✓
data	scaler	✓
drake	laptop	✗
take		
taken		
trie		
drunk		
eat		
toy		
scaler		

Idea 1 :

Check the query word against all the N given words.

TC : $O(N \cdot l \cdot Q)$

↑ TC to compare 2 strings.

Idea 2 :

Insert all the N words in HashSet and for every word in Query check if it is present in HashSet or not.

TC to insert/search 1 word in HashSet ⇒ $O(l)$

TC to insert N words in HashSet ⇒ $O(N \cdot l)$

Overall TC : $N \times O(l) + Q \times O(l)$

(Avg)

$N \times O(l)$: HashSet Creation

$Q \times O(l)$: Search

SC : $O(N \cdot l)$

TRIE : Hierarchical DS.

- → retrieval of words.
- → N-array tree.
- → Data is stored in top-down manner.

⇒ cricet ⇒ Spell checker.

- → Not a correct word.
- → Searching this word in the SET of correct words will return false.

⇒ Auto-Complete.

↓

Personalised Search.

Q. Given a word, check if it belongs to correct words.

d → a → a → t → a

a → a → t → e

t → r → i → e → d

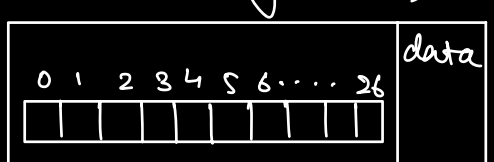
t → e → n → t

s → c → a → d → e → r

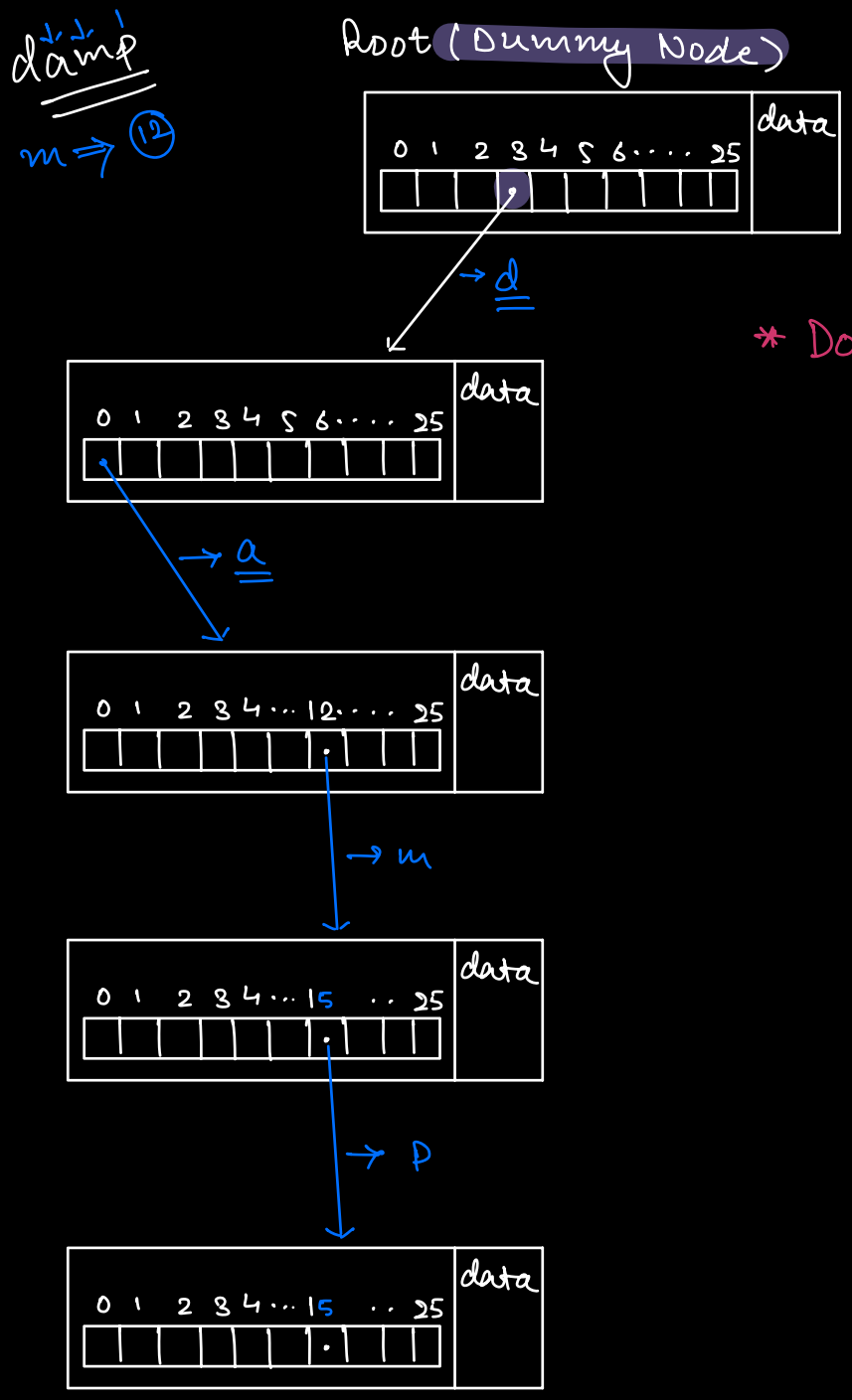
⇒ A word can start from any character from 'a' – 'z'.

```
Class Node {
    Char data
    Node Ch[26];
    // Initialize all children to NULL.
}
```

Root (Dummy Node)



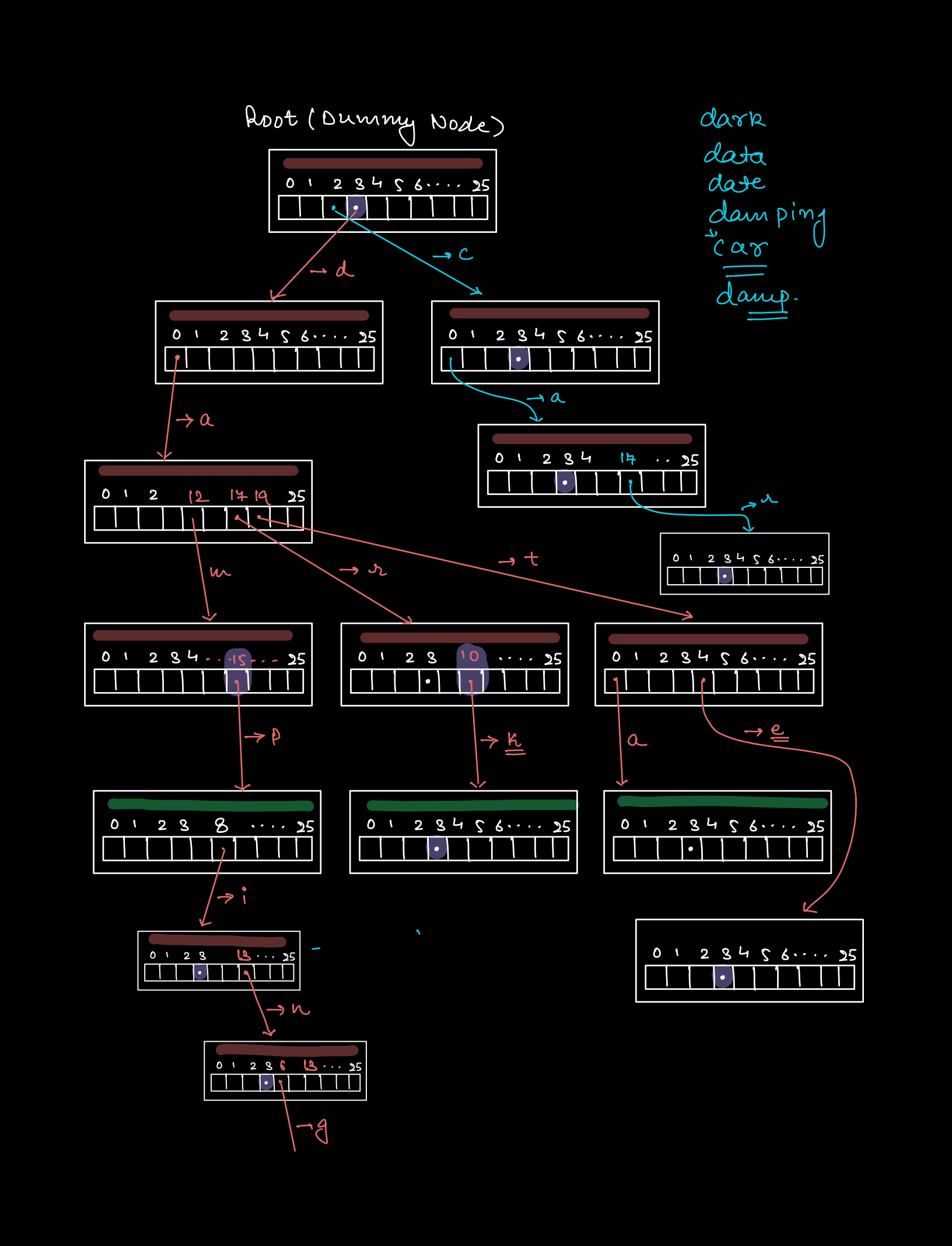
'a' → 0
b → 1
c → 2
d → 3
⋮
z → 25

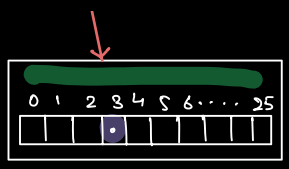


* Do we need data?

NO

```
Class Node {
    Node Ch[26];
    // Initialize all children to NULL.
}
```


Root (Dummy Node)
dark
data
date
damping
car
damp.
0 1 2 3 4 5 6 25
→ c
→ d
→ a
→ a
→ r
→ t
→ m
→ n
→ p
→ k
→ e
→ i
→ g
a
0 1 2 12 17 19 25
0 1 2 3 4 17 .. 25
0 1 2 3 4 5 6 25
0 1 2 3 4 .. 15 .. 25
0 1 2 3 10 25
0 1 2 3 8 25
0 1 2 3 13 ... 25
0 1 2 3 6 13 ... 25



→ Search "dump"

```
Class Node {
    bool isEnd;
    Node Ch[26];
    // Initialize all children to NULL
}
```

⇒ When we are at a particular node, how do we get to know that this is the end of some valid word.

⇒ bool isEnd;

Total space : N × L × 26 ⇒ Worst Case.

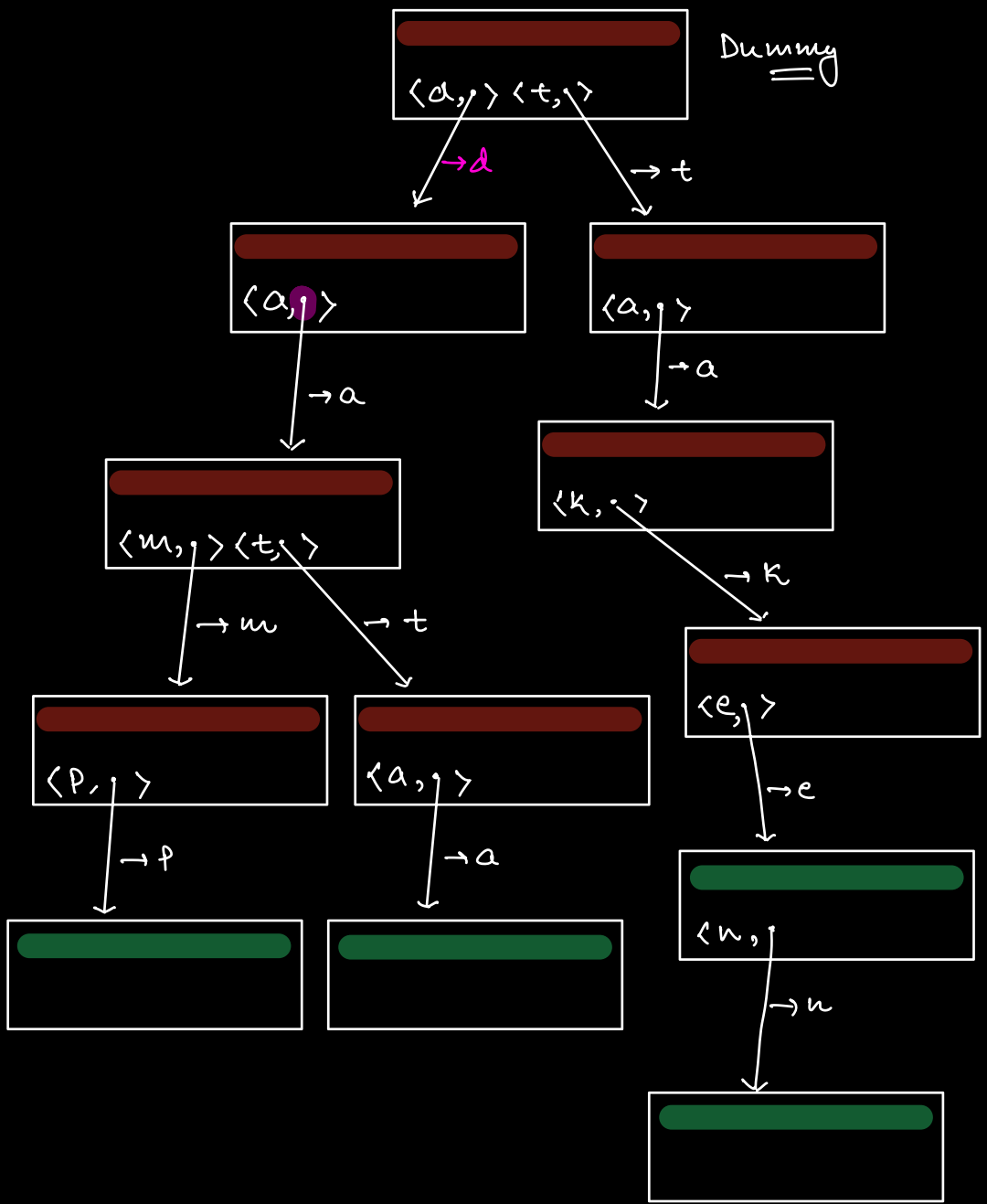
L ↑ Length of the word.

⇒ for any node if isEnd variable is true that means a valid is Ending at this node.

TC: $N \times L + Q \times L$

map < char, Node >

```
Class Node {
    bool isEnd;
    HashMap < char, Node > ch;
}
```

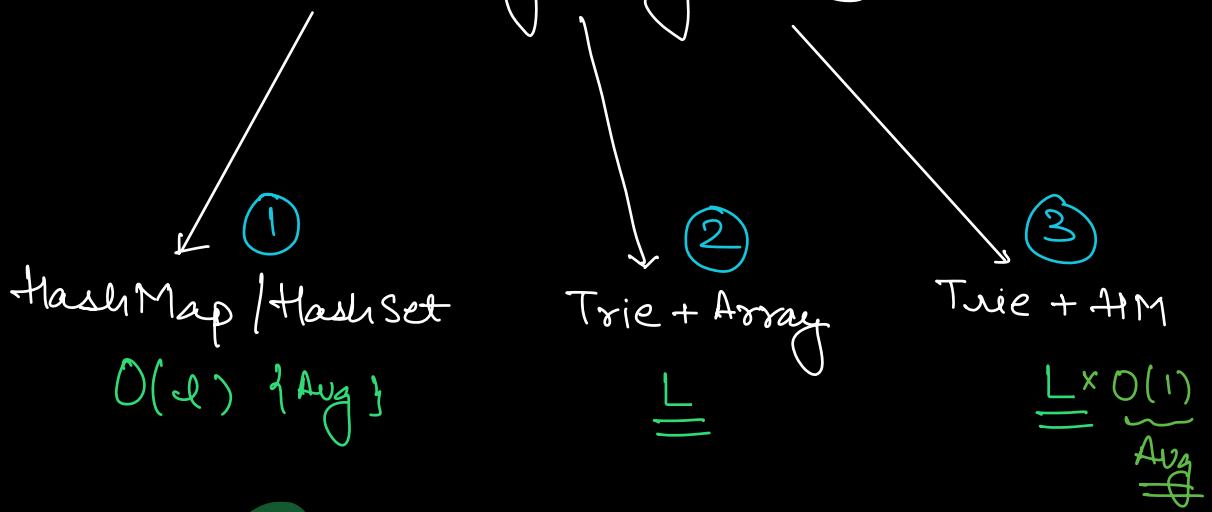


dump
data
↑
take
taken

* NO space wastage.

SC : N x L

To insert a word of length (L)



TC: 2 **3** 1

SC: 3rd is Best

Trie ≡ Using HashMap.

* Node Structure

```
class Node {
    bool isEnd;
    HashMap<char, Node> hm;
    Node() {
        isEnd = false;
    }
}
```

hm = new HashMap<Char, Node>();

```
Node root = new Node();

void add(String str, Node r){
    l = str.length()
    for(i=0; i<l; i++){
        char ch = str[i];
        if(!r.hm.contains(ch)){
            Node t = new Node();
            r.hm.insert(ch, t);
            r = r.hm[ch];
        }
        else{
            r = r.hm[ch];
        }
    }
    //All characters are inserted in Trie &
    //we are at last node.
    r.isEnd = true;
}
```

O(1) Avg

O(1) avg.

t

TC $\Rightarrow l \times O(1)$

$\Rightarrow O(l)$

```
bool find (String str, Node r) {
    l = str.length()
    for(i=0; i<l; i++) {
        char ch = str[i];
        if( !r.hm.contains(ch) ) {
            return false;
        }
        else {
            r = r.hm[ch];
        }
    }
    return r.isEnd;
}
```

O(1) avg.

TC: $l \times O(1) \equiv O(l)$

HW Implement Trie.
