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

damp
dark
data
drake
take
taken
trie
drunk
eat
try
scaler

(Q) Query

data ✓
take ✓
scaler ✓
laptop ✗

Idea 1:

Check the query word against all the N given words.

TC: $O(N \cdot \underset{\uparrow}{l} \cdot Q)$

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 : $\underbrace{N \times \overset{\text{Avg}}{O(l)}}_{\text{HashSet Creation}} + \underbrace{Q \times O(l)}_{\text{Search}}$

SC : $O(N \cdot l)$

TRIE : Hierarchical DS.

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

⇒ cricet ⇒ Spell checker.

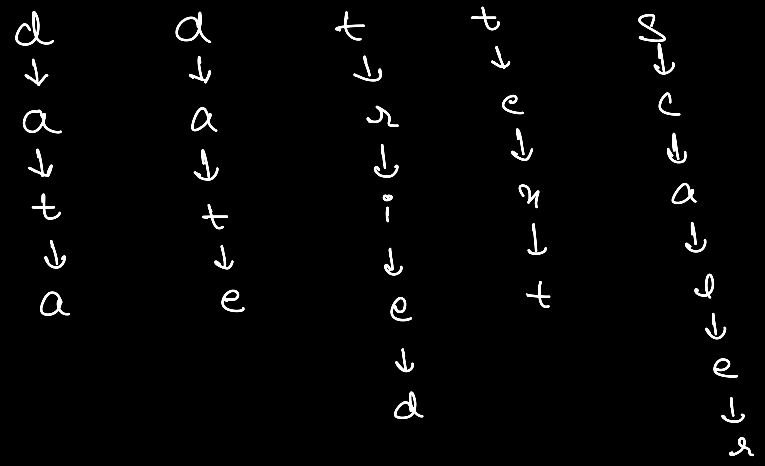
- → Not a correct word.
- → Searching this word in the SET of correct words will return false.

⇒ Auto-Complete.

↓

Personalised Search.

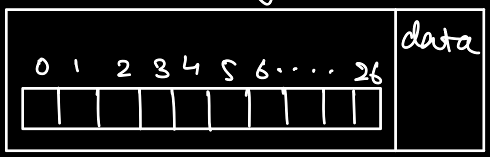
Q. Given a word, check if it belongs to correct words.



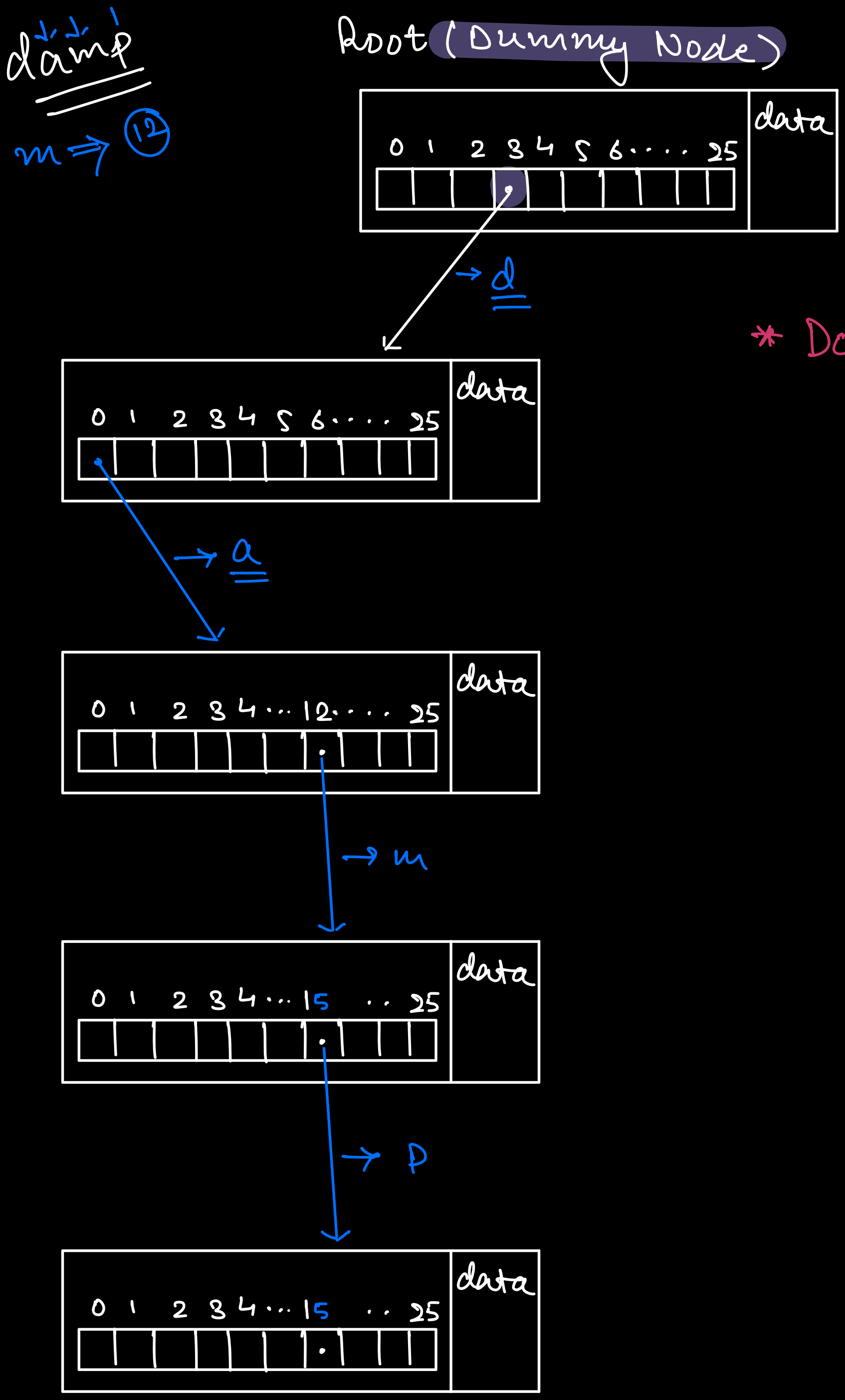
⇒ A word can start from any character from 'a' – 'z'.

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

Root (Dummy Node)



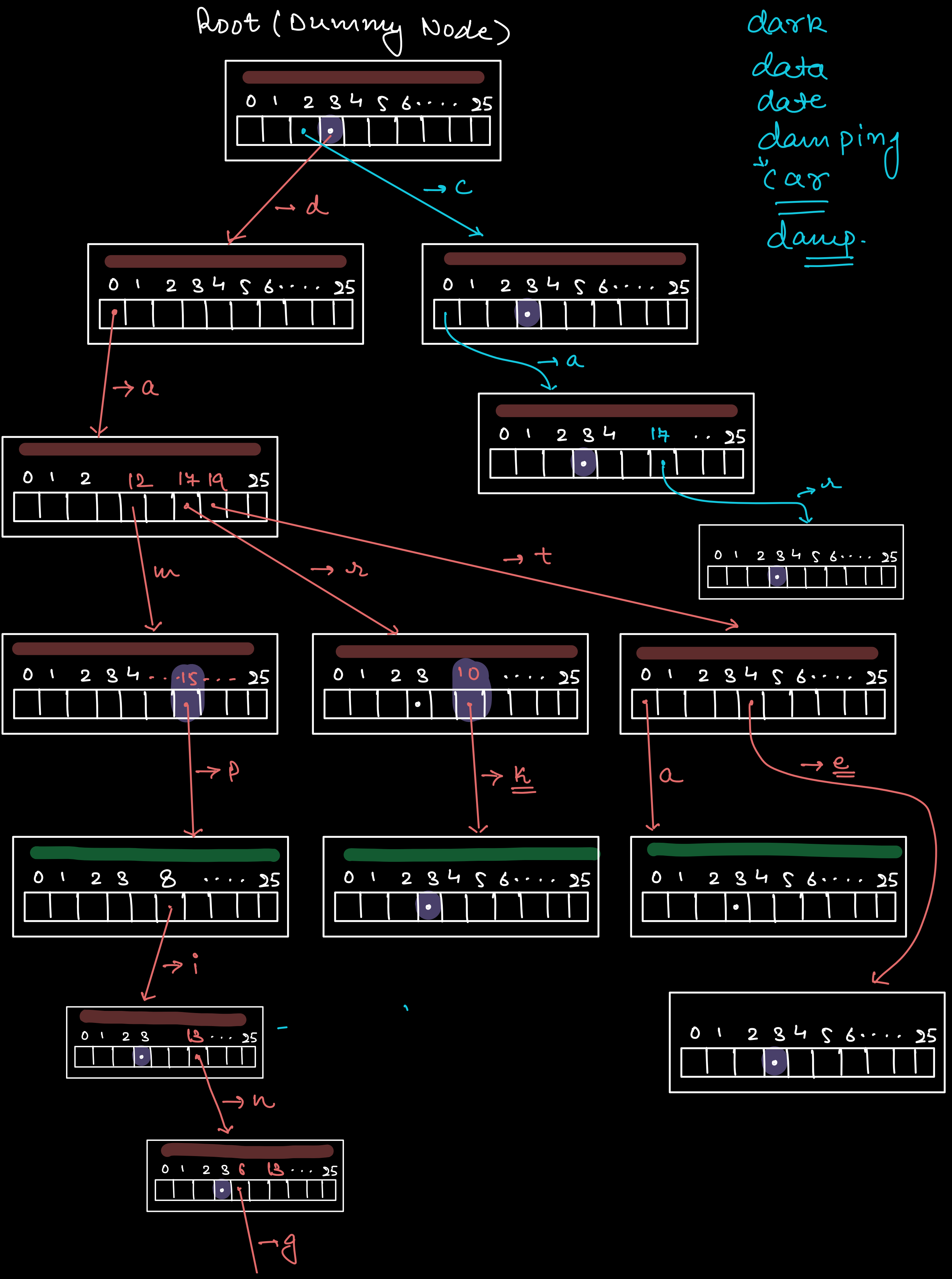
a → 0
b → 1
c → 2
d → 3
⋮
z → 25



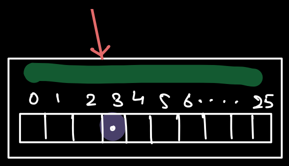
* Do we need data?

NO

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



dark
data
date
damping
car
damp.



→ search "damp"

```
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.

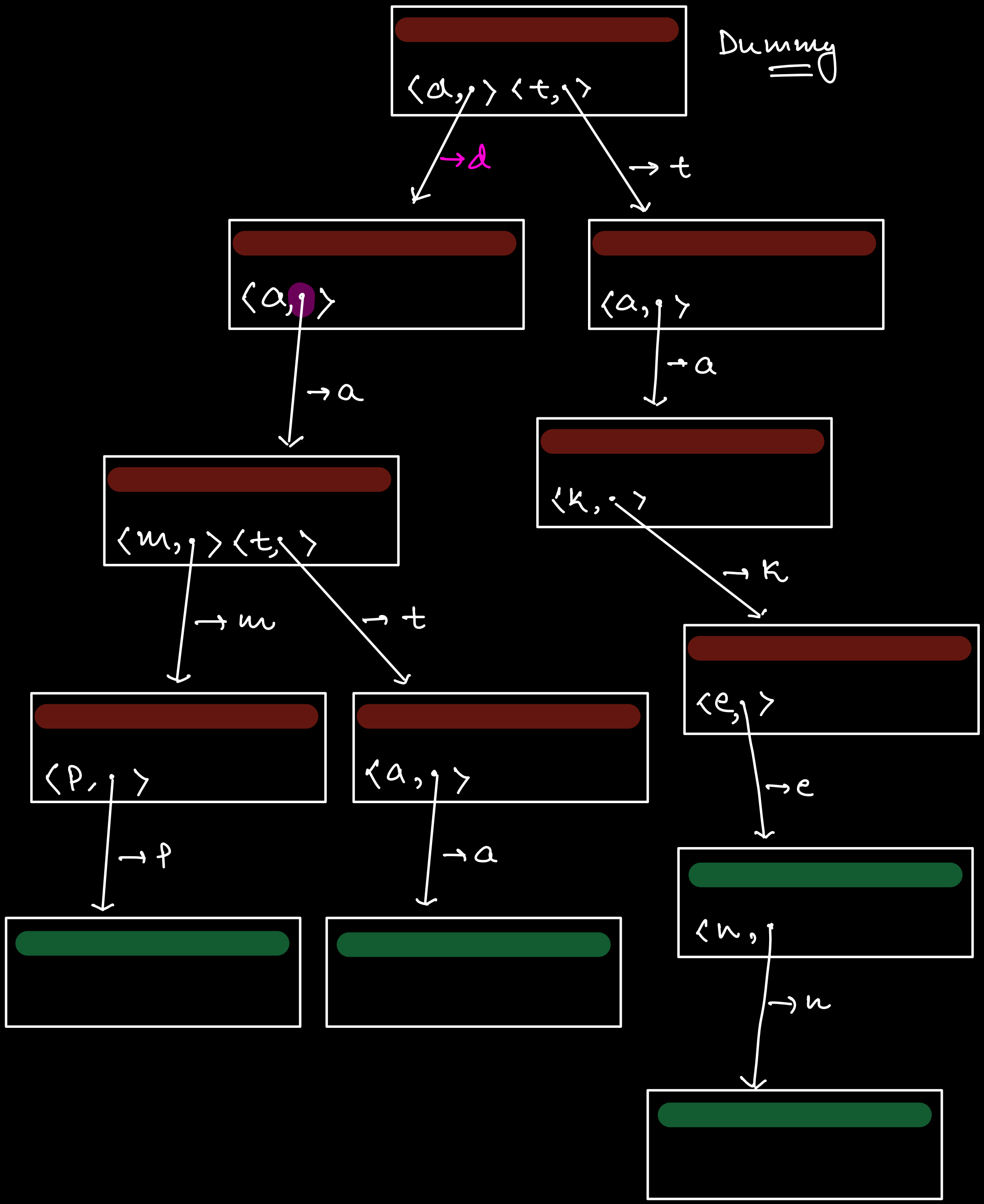
↑ Length of the word.

⇒ 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;
}
```



damp

data

take

taken

* NO space wastage.

SC : N×L

To insert a word of length ⓛ

① HashMap / HashSet

$O(l)$ {Avg}

② Trie + Array

$\underline{\underline{L}}$

③ Trie + HM

$\underline{\underline{L}} \times \underbrace{O(1)}_{Avg}$

TC : 2 3 1

SC : 3rd is Best

Trie ≡ Using HashMap.

* Node Structure

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

```
Node root = new Node();

void add(String str, Node r){
    l = Str.Length()
    for(i=0; i<l; i++){
        char ch = str[i];                        O(1) Avg
        if(!r.hm.contains(ch)){
            Node t = new Node();
            r.hm.insert(ch, t);                  O(1) avg.
            r = r.hm[ch];
                    t
        }
        else{
            r = r.hm[ch];
        }
    }
    //All characters are inserted in Trie &
    //we are at last node.
    r.isEnd = true;
}
```

TC ⇒ $l \times O(1)$

⇒ $O(l)$

```
bool  find (String str, Node r) {
      l = str.length()
      for (i=0; i<l; i++) {
            char ch = str[i];                     → O(1) avg.
            if ( !r.hm.contains(ch) ) {
                  return false;
            }
            else {
                  r = r.hm[ch];
            }
      }
      return  r.isEnd;
}
```

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

H/W Implement Trie.

——— * ———