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

Constraints :-

* All characters are ['a' - 'z']
* length of each string is <= L.

Words	Queries.
damp	scaler ✓
dark	draw ✓
data	dump ×
drake	
trie	
drawn	
drunk	
tried	
try	
draw	
scaler	
interviewbit	
amazon	

Approach 1 :-

For every query word, iterate over all the words match with the given set of N words.

TC : $O(L * N * Q)$

Approach 2:-

* Use Hashmap / HashSet.

1) TC of inserting / searching 1 String of length L in set
$\Rightarrow O(L)$

2) TC of inserting / searching N Strings of length L in set
$\Rightarrow O(N*L)$

Overall TC : $\underbrace{O(NL)}_{\text{HashSet creation}} + \underbrace{O(LQ)}_{\text{searching Q words.}}$

$O((N+Q)L)$

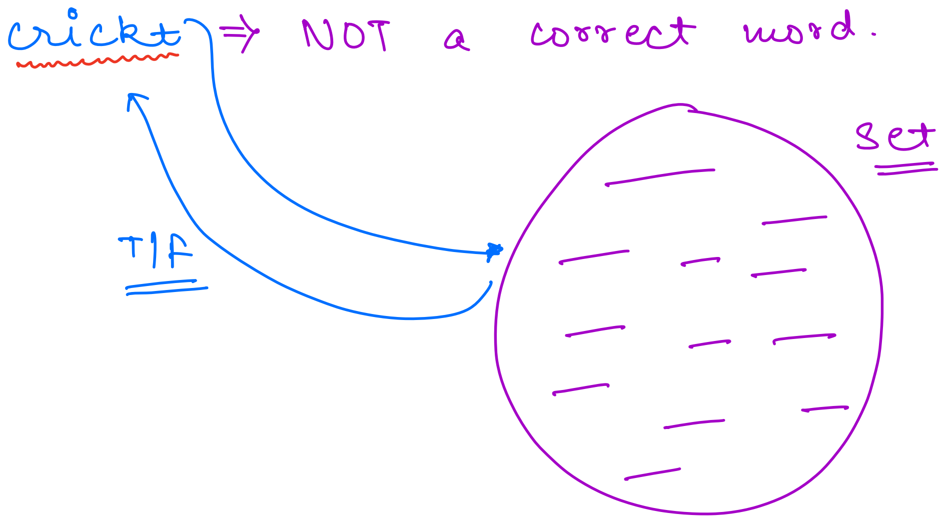
SC : $O(NL)$

TRIE
→ Hierarchical DS
→ N-array Tree.

⟹ Mostly used for retrieval

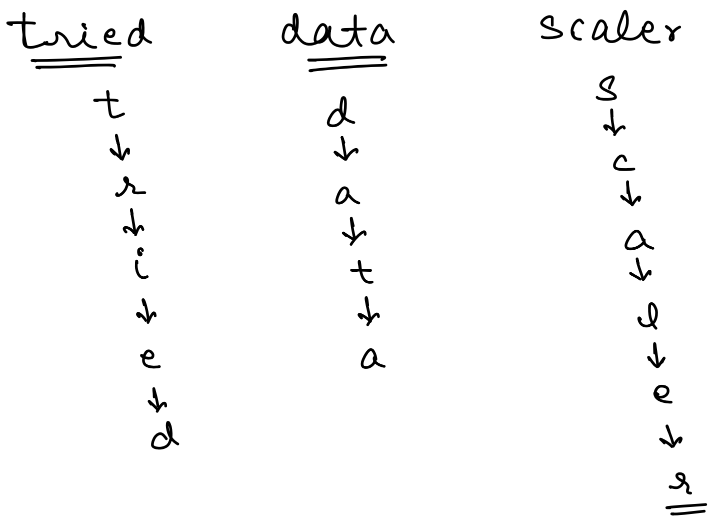
⟹ Data is stored in top down manner.

Google Doc



Auto complete. ⇒ Personalised search feature.

Q: Given a word, check if it is present in set of correct words?



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

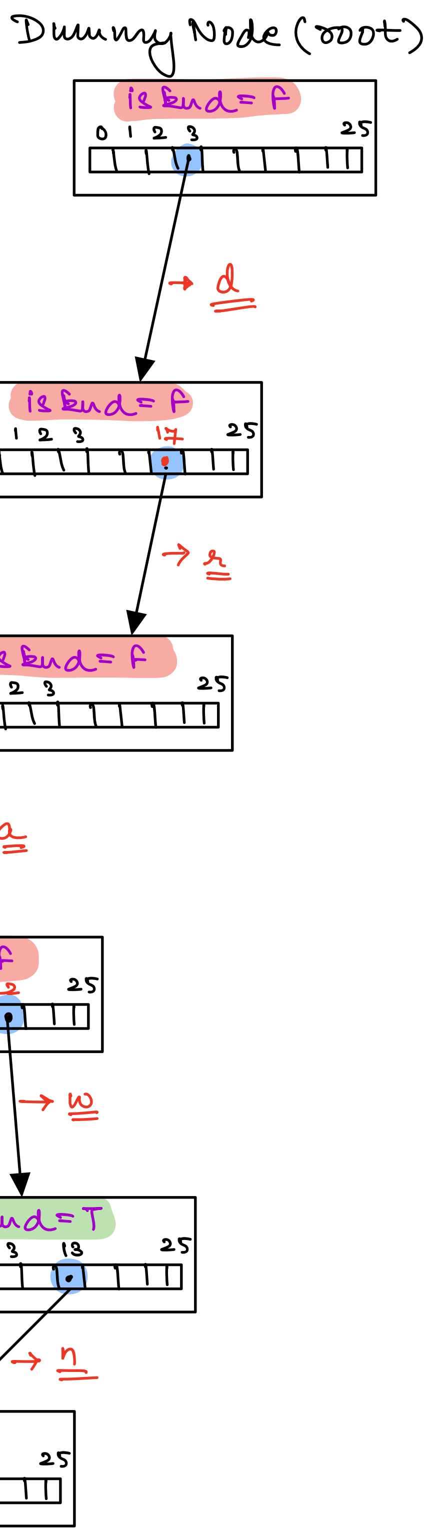
Char	index
a	0
b	1
c	2
⋮	⋮
z	25

```
Class Node {
        bool    isEnd;
        // If isEnd is T, valid word is ending
        // at this Node else NOT.
        Node ch[26]
        // Initialize all 26 Children
        // to NULL.
}
```

Trie creation

draw.

drawn



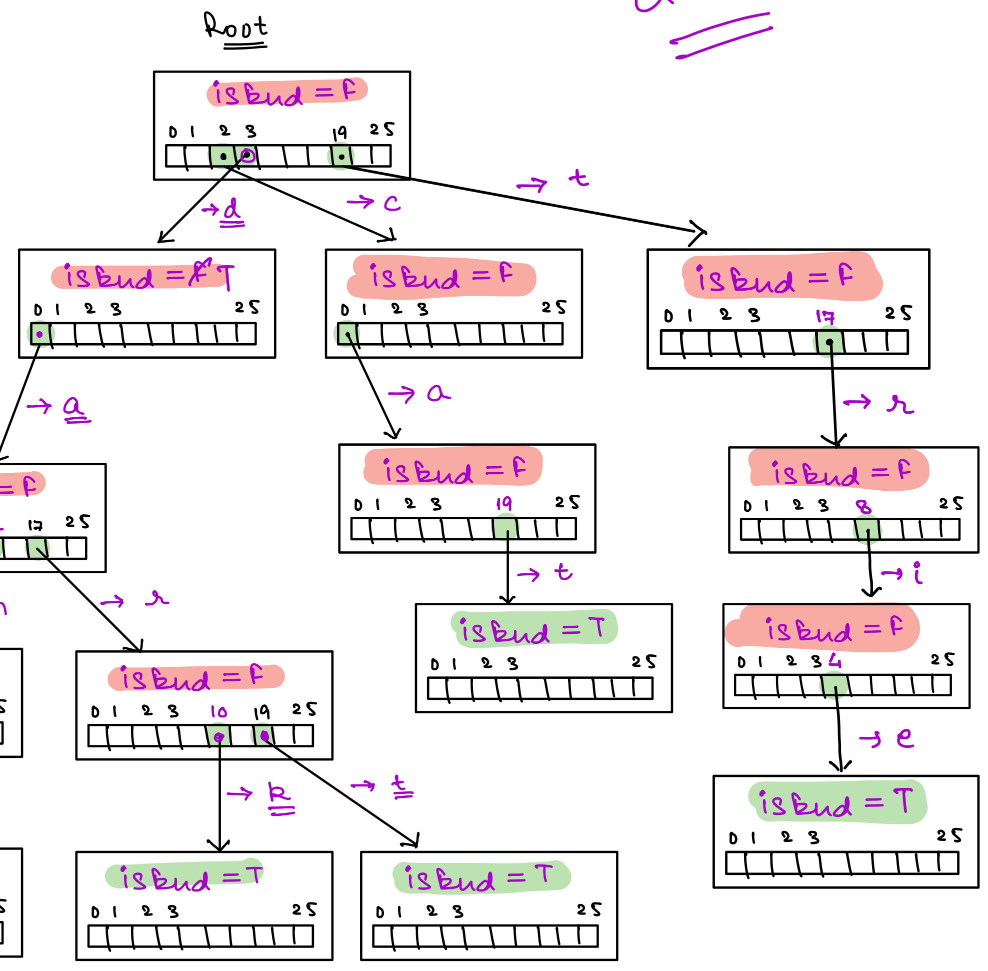
* Do we need data ? NO.

damp
dark
dart
cat

trie
drawn
drake

d

dar



for any node, if isEnd variable is True it means a valid word is ending at this Node.

Space Complexity

$O(N * L * 26)$ ⇒ Worst Case.

Every node has 26 children.

⇒ Huge Space Wastage.

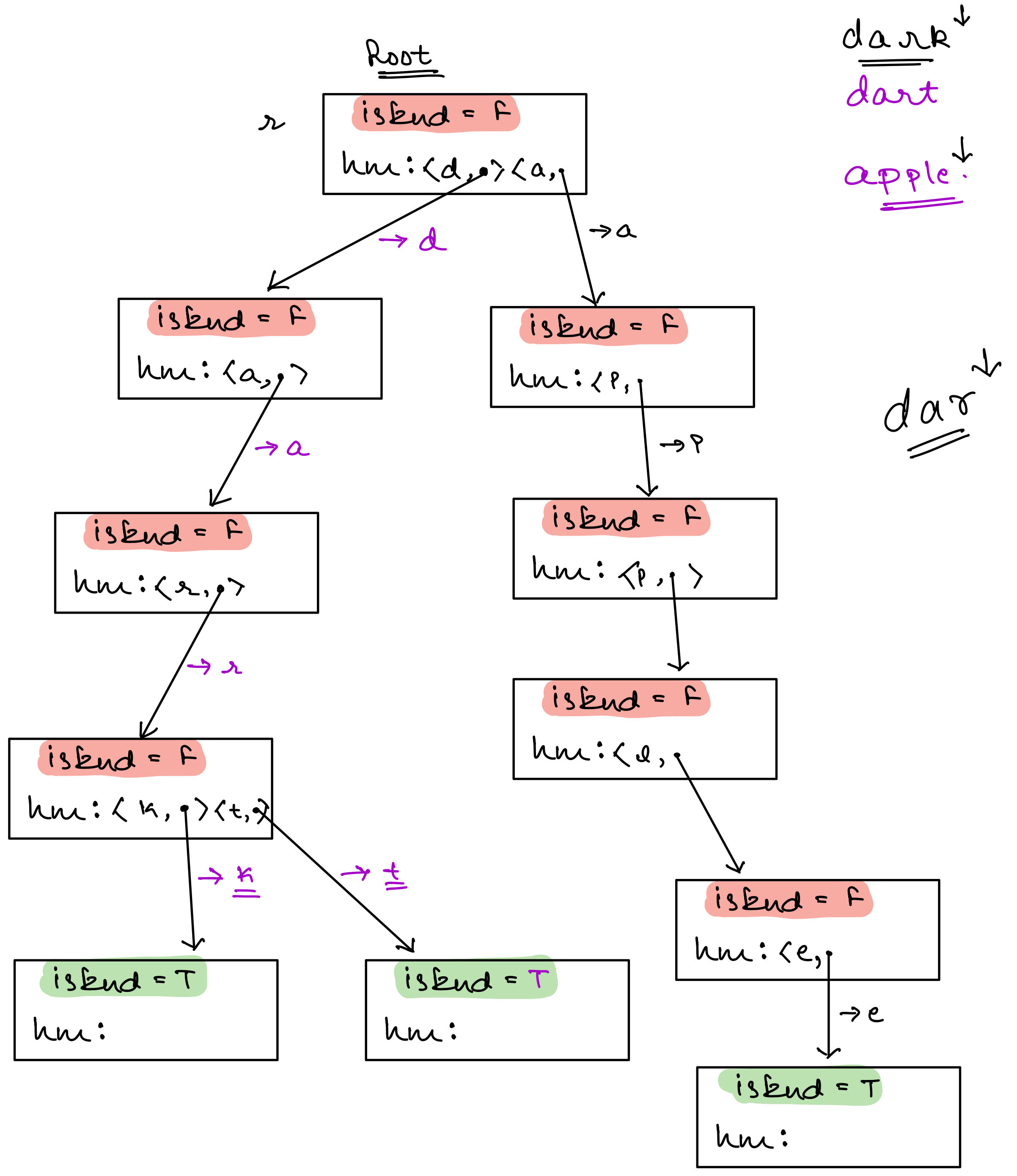
TC

* In trie, # of iterations to search 1 word ⇒ L
* In trie, # of iterations to insert 1 word ⇒ L

$O(N \times L + Q \times L)$

* Use HashMap to reduce the space wastage.

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


Root
r
isEnd = F
hm: <d, •> <a, •
→ d
→ a
isEnd = F
hm: <a, •>
isEnd = F
hm: <p, •
→ a
→ p
isEnd = F
hm: <r, •>
isEnd = F
hm: <p, • >
→ r
isEnd = F
hm: < k, •> <t, •>
isEnd = F
hm: <l, •
→ k
→ t
isEnd = F
hm: <e, •
→ e
isEnd = T
hm:
isEnd = T
hm:
isEnd = T
hm:
dark
dart
apple
dar

SC: O(NL)

```
Class Node {
        bool isEnd;
        HashMap< Char, Node > hm;
        Node () {
            this.isEnd = false;
        }
}

1) add (Str, root)

2) find (Str, root)

Node root = new Node();


* Void add ( String str, Node r) {
        int n = str.length()
        for (i=0; i< n; i++ ) {
                Char ch = str[i];
                // insert str[i]
                if ( ch is NOT present in r.hm) {
                        Node t = new Node ();
                        r.hm.insert (ch, t);
                        r = r.hm[ch] // (t)
                }
                else {
                        // Get the ref of ch in r.hm
                        r = r.hm[ch];
                }
        }
        // All characters are inserted in Trie & we
        // are in last Node
            r.isEnd = true;
}
```

```
*   bool find ( String str, Node r) {
        int n = str.length()
        for (i=0; i< n; i++ ) {
            Char ch = str[i];
            if ( ch is NOT present in r.hm) {
                    return false;
            }
            else {
                    r = r.hm [ch];
            }
        }
        return r.isEnd;
    }
```

```
for( i=0; i< N; i++ ) {
        read word;
        add (word, root)
}
```
$\Rightarrow O(N*L)$

```
for( i=0; i< Q; i++) {
        read word;
        if ( find ( word, root) )
                print (Present)
        else
                print (Not present)
}
```
$\Rightarrow O(Q*L)$

——— * ———