Today's Content:

→ Tries data structure basics

→ Check if a given word is valid or not?

↳ 2nd Question:

good news: Iterative codes

→ TC: To compare 2 strings of len l = $O(l)$?

S_A : a a b c d

S_B : a a b c e

S_A : a c d e

S_B : c a e d

$S_A == S_B$:

↳ $\approx O(N)$

N is length of string.

IQ) Given (N) strings & Q queries, for each query check if it is present in N strings

Constraints:

All characters are ['a' - 'z'] & 1 ≤ len of each string ≤ l

Words:

damp
dark
data
drake
drawn
drew
dried
drunk
draw
frie
fried
frump
tea

Queries:

data ✓
draw ✓
drew ✓
dump ✗
drawed ✗

Ideal

→ For every query compare with all words

↳ TC: Q * { N * l }

↳ comparing N times

SC: O(1)

Ideal:

→ Insert all words in hashset

: For every query check if word is present in hashset

TC: N * O(l) + Q * O(l)

Note: To insert/update/delete/search string of len = l in hashset

TC: O(l)

→ Hierarchical data Structure

→ N-array Tree: no of child nodes > 2

Trie: { Data is filled top-down }

→ [This is used for information retrieval]

→ [autocomplete / autosuggest
Suggests are at a personal level
browser to browser it changes]

cricet → not there
mails →
Spell check

// all correct words are stored:
→ add elements in datastructure:
→ every word we type, we check if it's in correct words

Tries

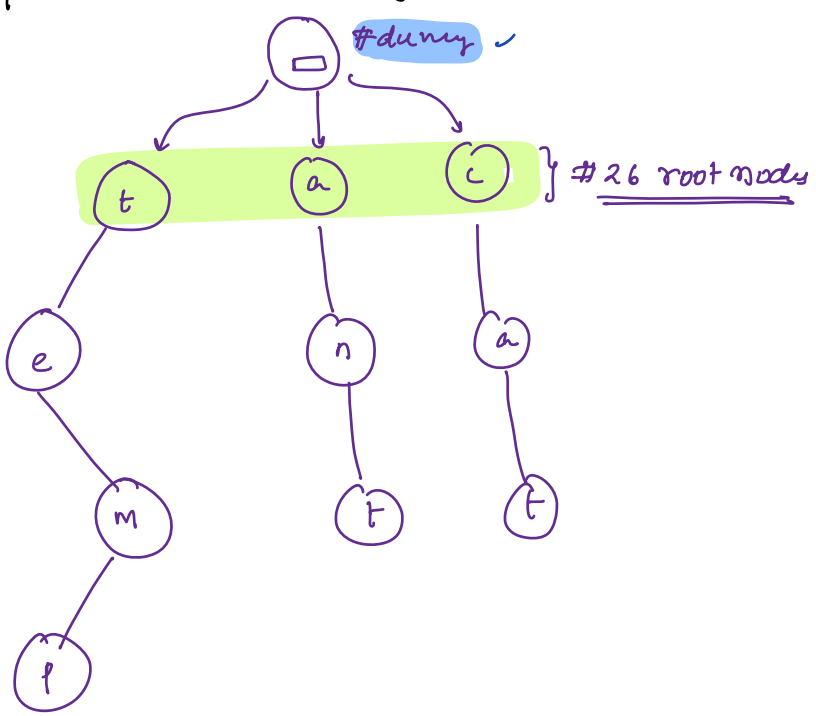
```
→ .....
class Node {
    int data
    Node left
    Node right,
    Node c[2]
        ↓
    array of object
    reference

    Node c[26]
        ↓
    array of object
    ref: 26
        ↓
[c[0], c[1], c[2] .. c[25]]
}
```

// Strings: { every character a-z' }

→ temp: character char by char

→ ant

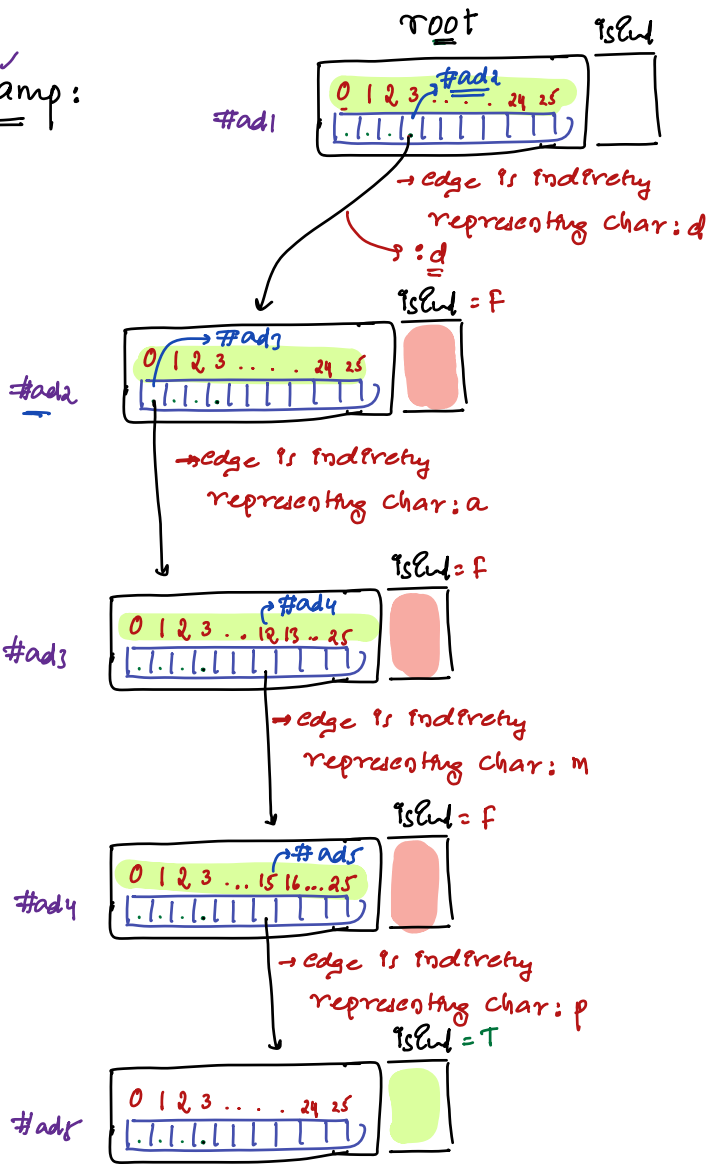
→ cat



Q): given a query, check if it belongs to [Correct Words]

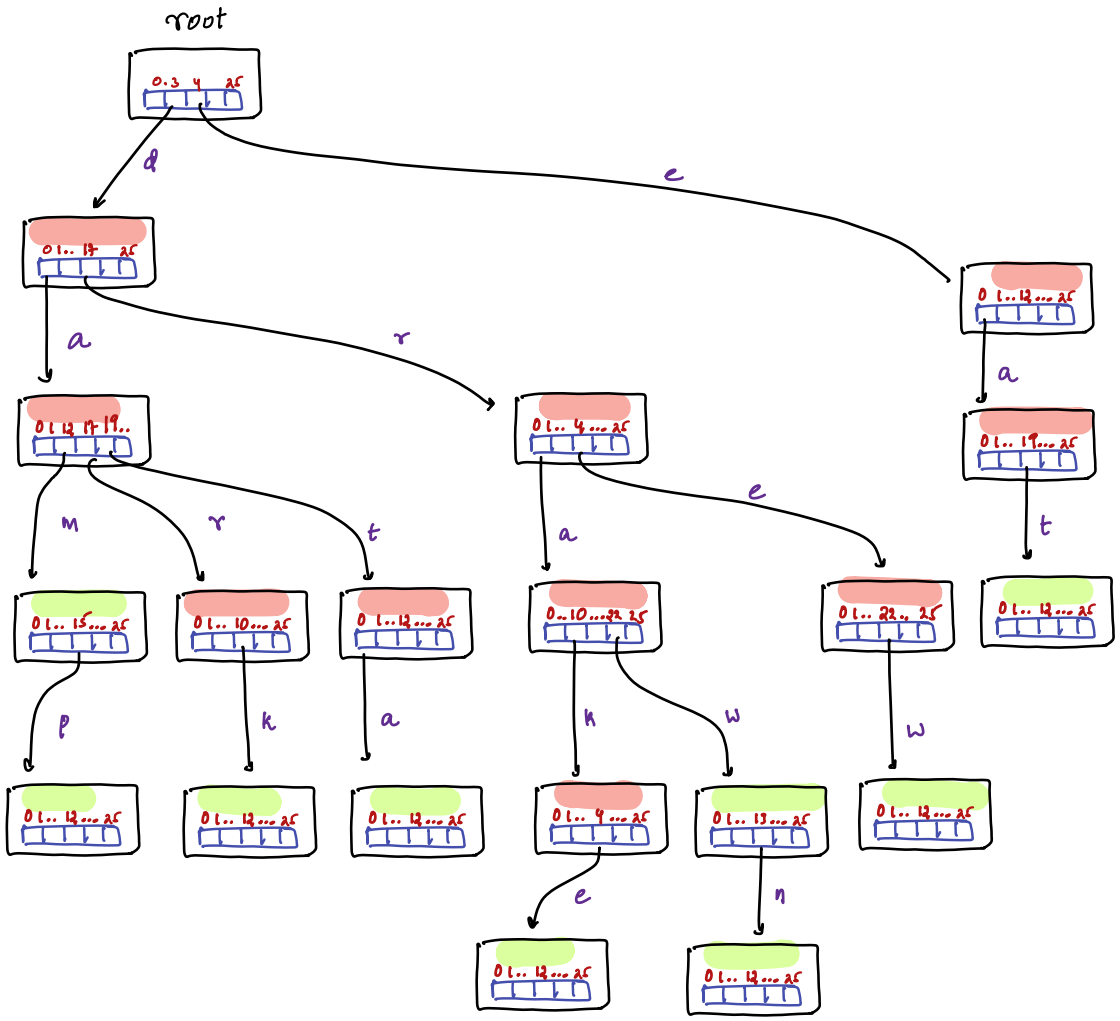
Trie

dump:




```
class Node {
    char ch; ?// not needed
    bool isEnd // end of string
    Node c[26];
    Node(char ch) {
        ch = ch;
        for(i = 0; i < 26; i++) {
            c[i] = null
        }
    }
}
```

- # a → 0
- # b → 1
- # c → 2
- ⋮
- # z → 25



Insert all words

dam ✓

dark ✓

data ✓

draki ✓

drawn

drew

eat

8): draw

draw

8) draw

8) drawe

dam

TC: In Insert 1 word of l length: L

In Search 1 word of l length: L

To Insert N words & Search 8 words: N*L + 8*L : faster

hashset : TC: N * O(l) + 8 * O(l)

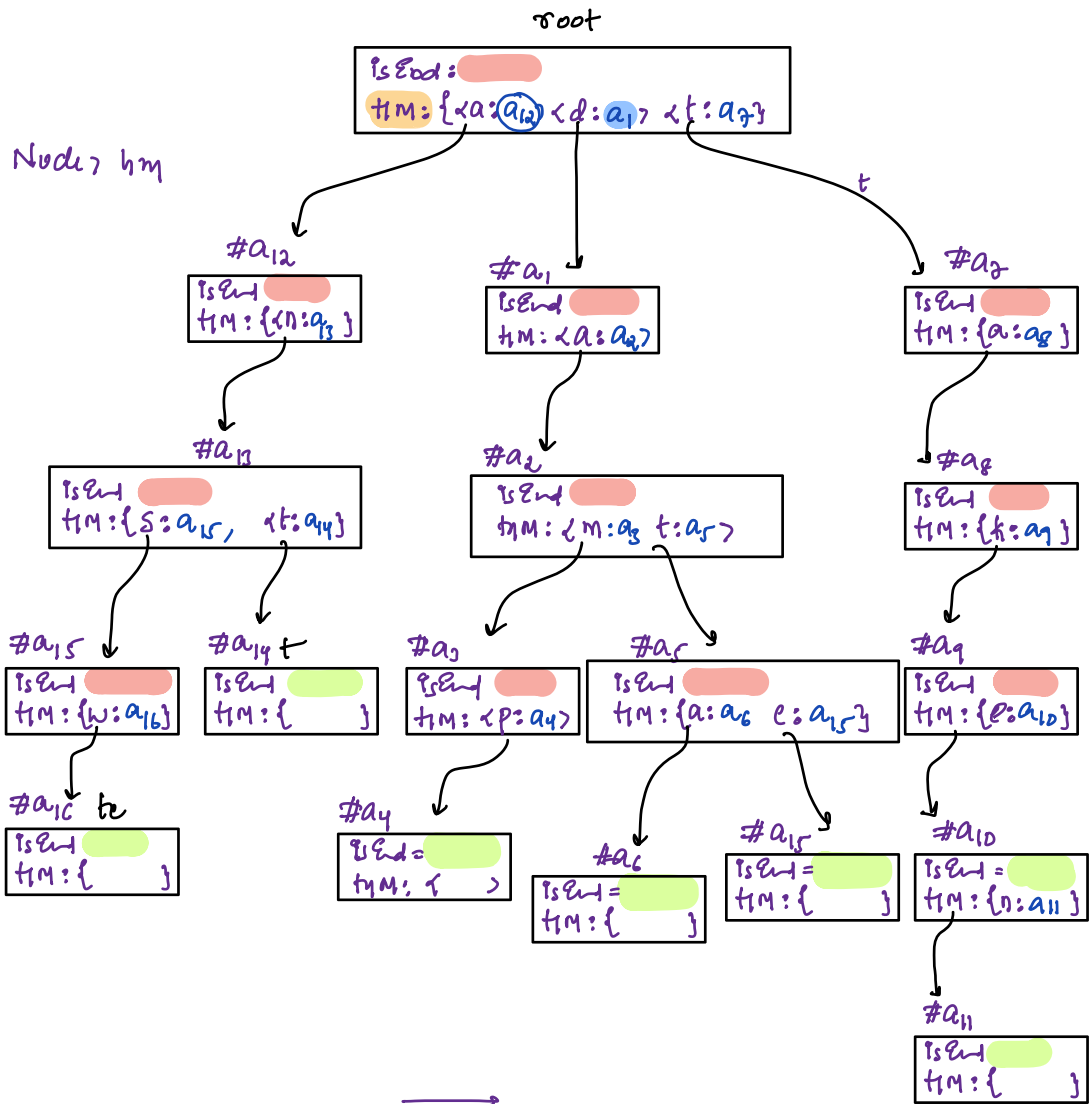
Scan Trie: 8:28 → 8:38

```
class Node {
    bool isEnd;
    Hashmap<char, Node> hm
}
```

damp ✓
data ✓
take ✓
taken
ant ✓
date ✓

0	1	2	3
a	n	s	w

#dalp
#dat
ant



TC: In Insert 1 word of l lentn: L*O(1)

In Search 1 wrd of l length: L*O(1)

↳ To Insert N wrds & Search M wrds: TC: N*L*O(1)

Appr1: Set: 1	Approach Trie: ch[26] 2	Approach Trie: hashmap 3
TC: N*O(1)	TC: N×L	TC: N×L×O(1)

Speed: 2 3 1 → { Approach: 1 & technique we used

Class Structure:

```
class Node {
    bool isEnd
    hashmap<char, node> hm
    Node() {
        isEnd = false
    }
}
main() {
    Node root = new Node();
    Read N
    while(N--) {
        Read word;
        insert(root, word);
    }
    Read Q
    while (Q--) {
        Read word
        if( search(root, word)) {
            print("present")
        }
        else {
            print "absent"
        }
    }
}
```

```
void insert( Node root, string data) {
    Node t = root
    i = 0; i < data.size(); i++) {
        char ch = data[i]
        //search ch in hashmap in node t
        if(t.hm.search(ch) == True) {
            t = t.hm[ch]
                    // In hashmap get addr of ch
        }
        else {
            Node nn = new Node();
            t.hm.insert({ch, nn})
            t = nn
        }
    }
    t.isEnd = True
}
```

```
bool search( Node root, string data) {
    Node t = root
    i = 0; i < data.size(); i++) {
        char ch = data[i]
        if(t.hm.search(ch) == True) {
            t = t.hm[ch]
        }
        else return false
    }
    return t.isEnd
}
```