Today's content

(i) Basics of tries

(ii) Internal working of tries

(iii) Search for a query in set of words.

[hashmap implementation -> recording].

TC to compare if 2 strings of length 'l' are equal or not?

TC : 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

Q. Given 'N' strings and 'Q' queries, for each query check if it is present in 'N' strings or not.

Constraints:

All characters are ['a'-'z'] & $1 \le$ length of each string $\le L$

Words:
- damp
- dark
- data
- drake
- drawn
- drew
- dried
- drunk
- draw
- trie
- tried
- trump
- tea

Queries:
- data ✓
- draw ✓
- drew ✓
- dump ✗
- drawed. ✗

ideal:

→ For every query, compare with all words

TC: $Q * (N * L)$, SC: $O(1)$

(Q: queries; N: words; L: compare)

idea 2:

→ insert all words into hashset,

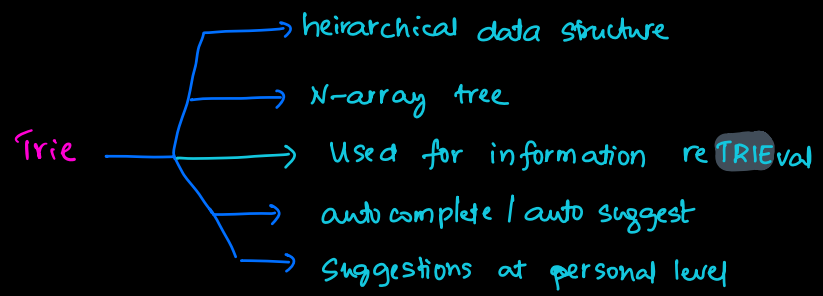
→ for every query, check if its present in hashset or not.

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

Note: To insert / update / delete / search a string in hashset

=> TC: $O(L)$.

A new data structure => Trie.



Cricet → not there

mails

// store all correct words

// every word that is typed, we need to check if its correct word

} Trie.

Class Node

int data

Node left

Node right

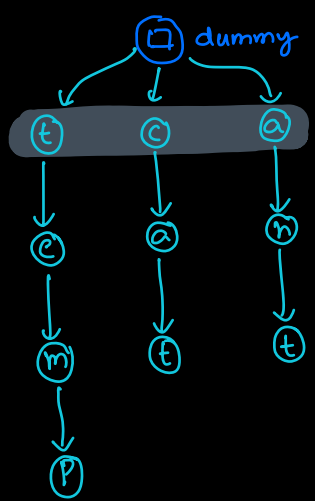
// We'll store strings only. ['a-z']

temp

cat

ant

data



Class Node

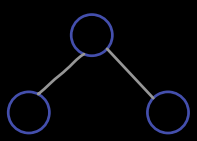
data

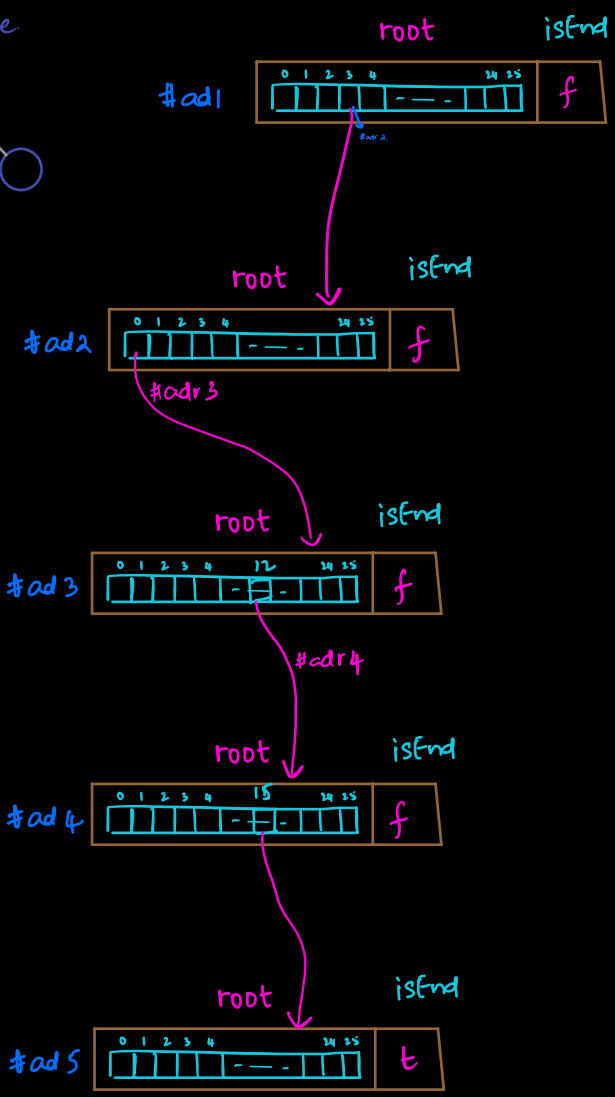
Node c[26]

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

Trie:

Binary Tree



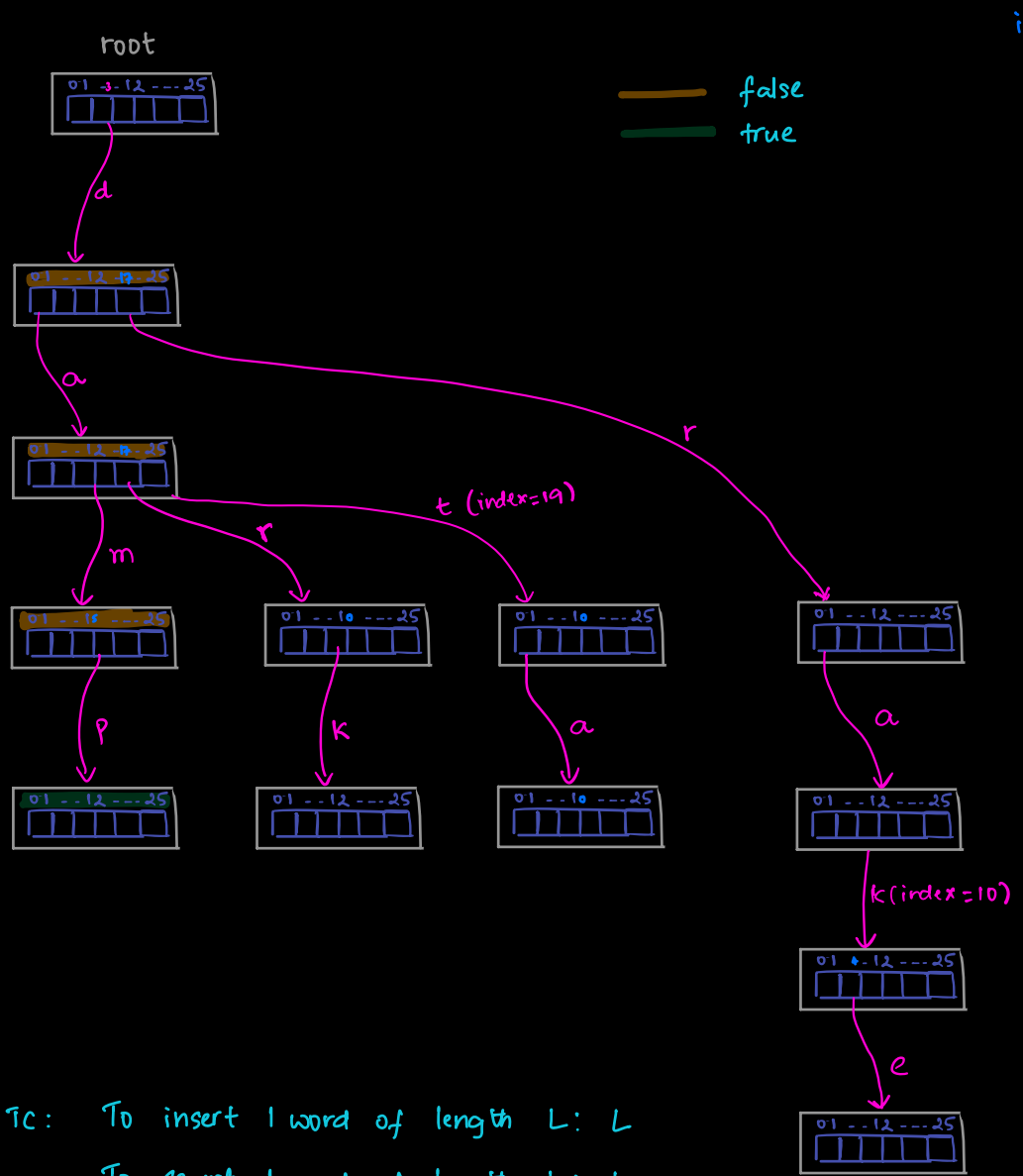


dump.

```
class Node
    char ch, bool isEnd   ← redundant
    Node c[26]
    Node (char c)
        ch=c
        c[i]: null
            //26 pos
```

a -> 0
b -> 1
c -> 2
d -> 3
:
z -> 25

Insert below elements into trie.



insert
- damp ✓
- dark ✓
- data ✓
- drake ✓
- drawn
- drew
- eat
- damper

search
- ✓ dark
- ✗ dam

TC: To insert 1 word of length L: L

To search 1 word of length L: L

To insert N words & search Q words : N*L + Q*L

search "Scaler" → hashset approach → generate hash value & check → L → false.

→ tries approach → just 1 comparision.

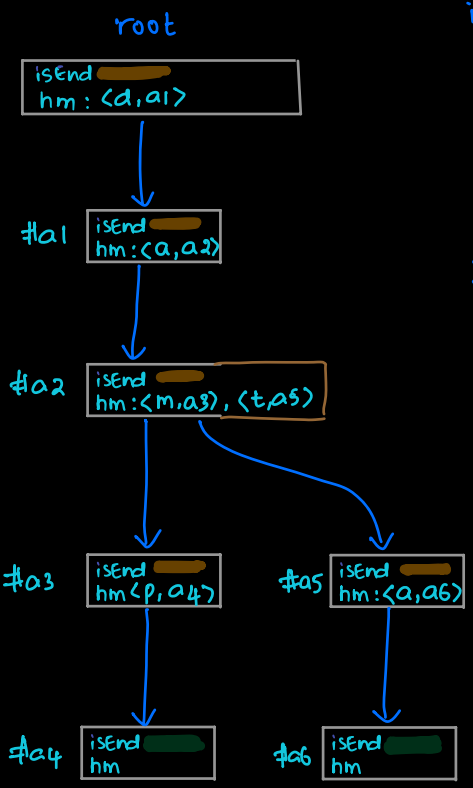
=> In practice, tries are faster than hashset approach.

The problem here is we're wasting too much space.

Try using hashmap instead of array.

Class Node

- boolean isEnd
- Map < Char, Node > hm



insert:

- damp
- data
- take

search:

- date

Code:

```
Class  Node
    boolean isEnd
    Map < Char, Node > hm

main()
    Node root = new Node()
    Read N
    while (N--)
        Read word
        insert(root , word)
    Read Q
    while(Q--)
        Read query
        if ( Search(query , root))
            print ("Present")
        else
            print ("absent")
```

```
Void  insert( Node root, string data)
    Node t= root
    i=0; i<data.length ; i++
        Char ch= data[i]
        //check if its in hm
        if( t.hm.search(ch) ==true)
            t= t.hm[ch]
        else
            Node nn = new Node()
            t.hm.insert ( ch, nn)
            t= nn
    t.isEnd = true.
```

```
boolean  search ( String data, Node root)
    Node t= root
    i=0; i<data.length ; i++
        Char ch= data[i]
        //check if its in hm
        if( t.hm.search(ch) ==true)
            t= t.hm[ch]
        else
            return false
    return  t.isEnd
```