Today's Content: { Tries / Heaps / Greedy / Back-tracking / Dp / Graphs

→ Tries data structure basics

→ Check if a given word is valid or not?

→ 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

1Q) 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
fea

Queries:

data ✓
draw ✓
drew ✓
dump ✗
drawed ✗

Idea1:

→ For every query iterate on all words & match with each & every word

TC: Q * [Match query with all N words ?]

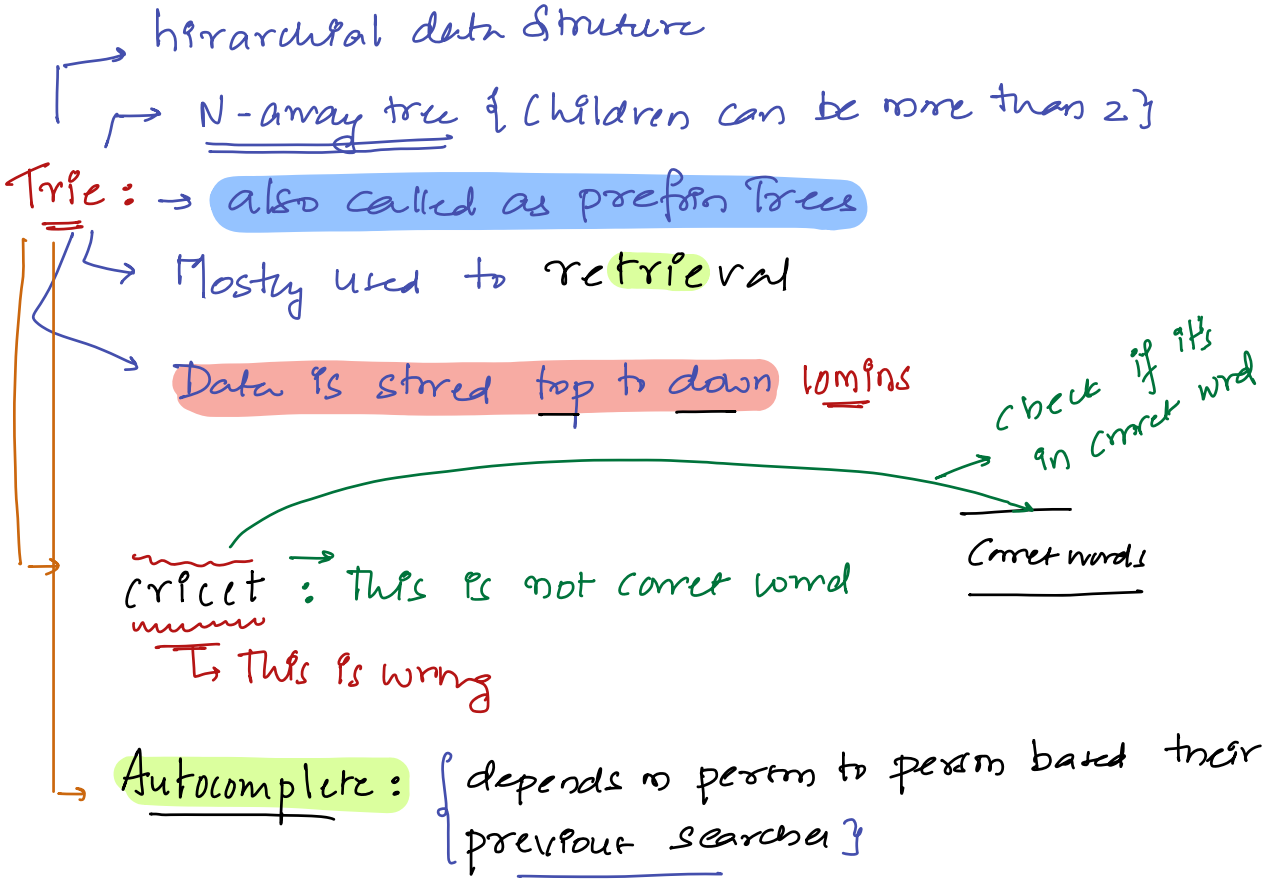
↳ Q [N * L] ⇒ TC: O(QNL)

Idea2: → Insert all words in hashset & for every check search if word present in HS

N * O(L) + Q * O(L)

TC: NL + QL SC: O(N*L)

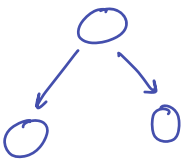
Note: To Insert / Search / Delete a String of len : M in HashSet / HashMap it will take O(M) time


hierarchical data structure
N-array tree { children can be more than 2 }
Trie : → also called as prefix Trees
Mostly used to retrieval
Data is stored top to down lowins
check if its an correct word
Correct words
cricet : This is not correct word
This is wrong
Autocomplete : { depends on person to person based their previous searches }

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

Trie

Binary Tree



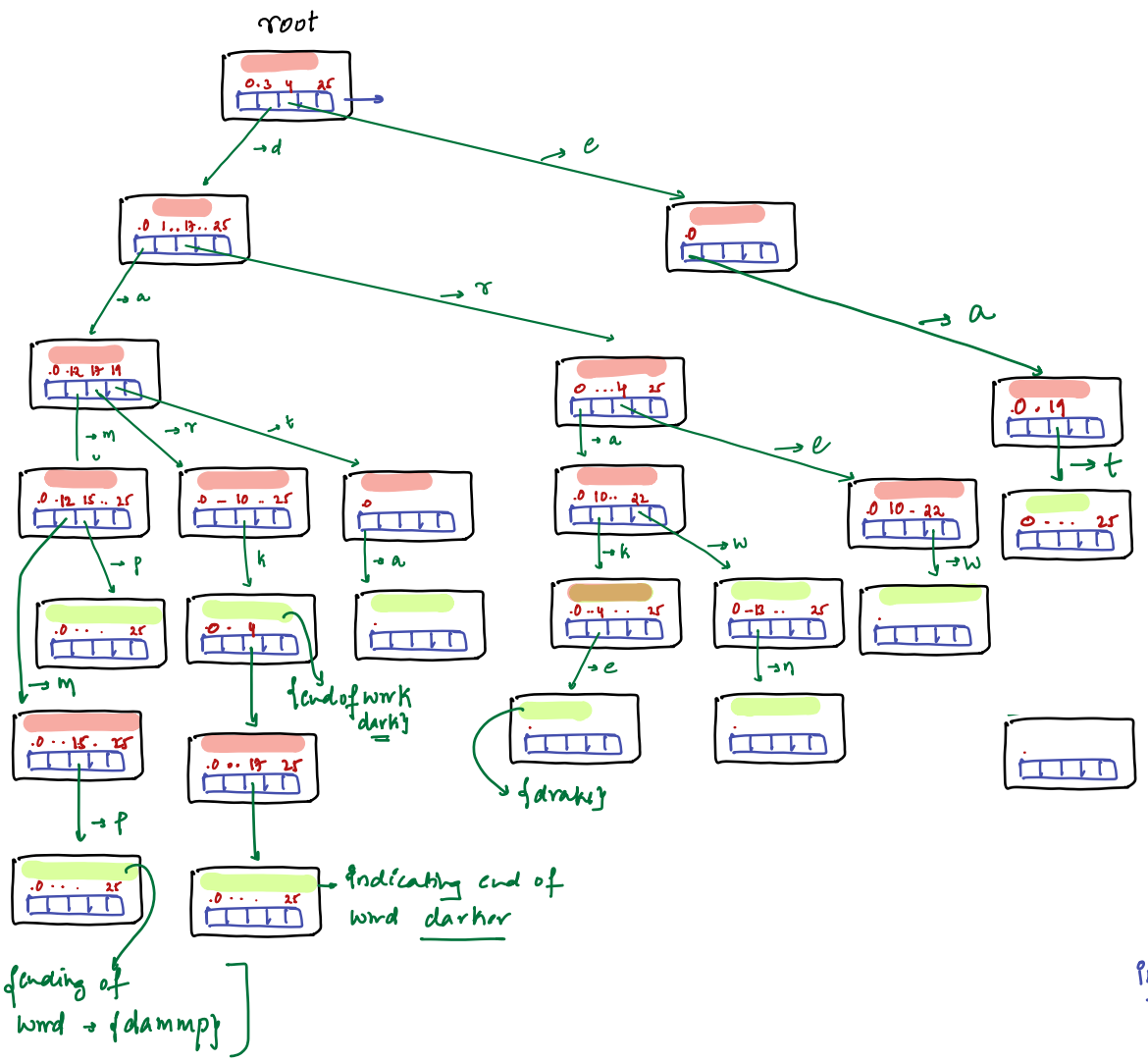
damp


```
class Node {
    bool isEnd
    Node ch[26];
    [All children initialize to Null]
}
```

[Initialize with False]

char	index
'a'	0
'b'	1
'c'	2
⋮	
'z'	25

→



Insert all words

damp ✓
dark ✓
data ✓
drake ✓
drawn ✓
drew ✓
eat

insert draw

Q) draw

Q) : draw *

insert draw

Q) draw ✓

insert drak

Q) drak

Q) drake

insert darker

dammp

// At every node we need to know, if word is ending there are not,

↳ at every node keep a boolean isEnd, if a word is ending at that particular, simply make it as True else false

// At every node we have 26 children, but we might not use all children → {huge space wastage}

TC: N*L + Q*L

better than hashset/hashmap

SC: N*L*26 | huge space wastage?

10min } space complexity → 10:55pm

// In Tries to insert a string of L → L iterations } → worst

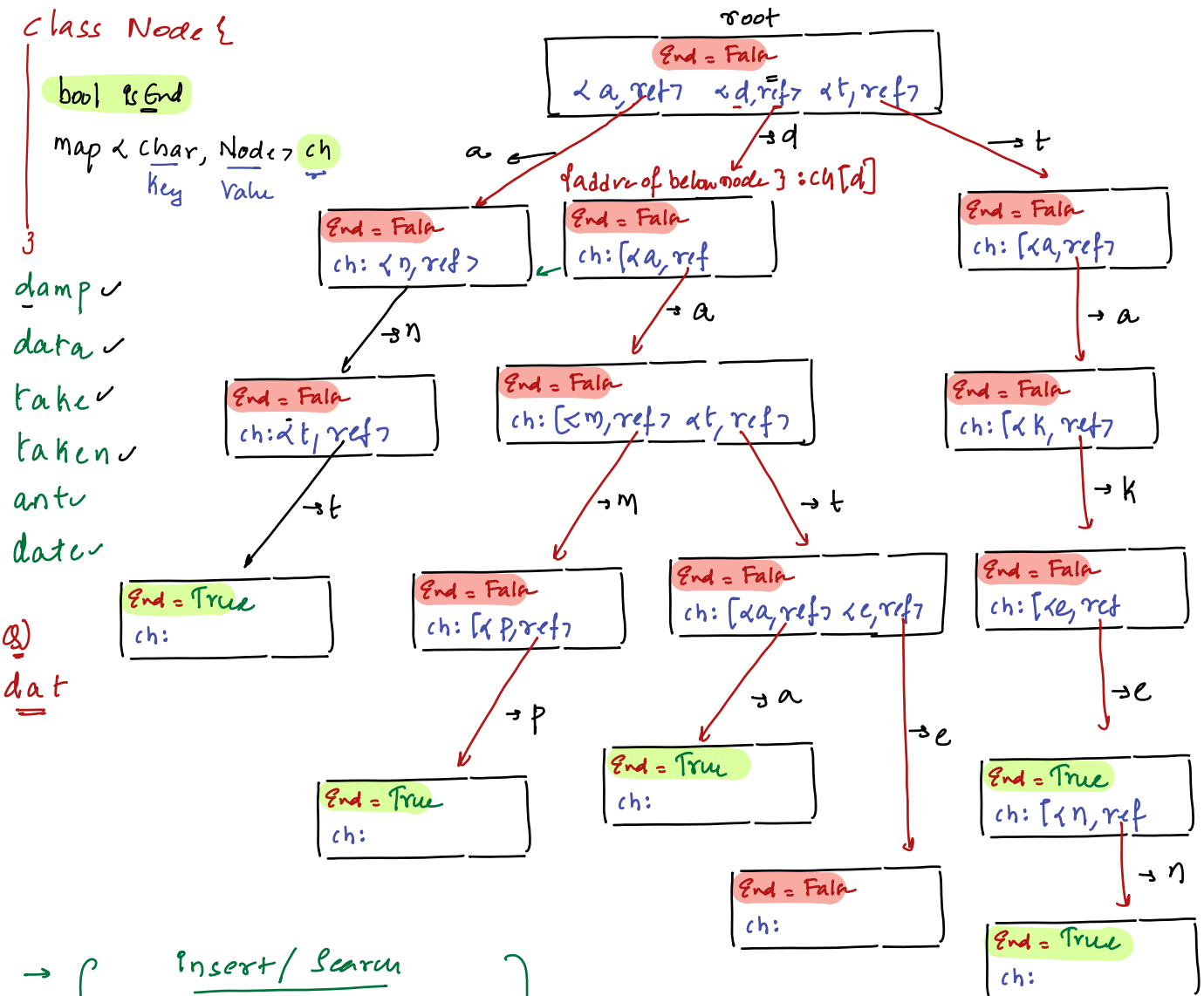
// In HashSet/HashMap to insert a string of L → O(L) → avg case

// Tries has better TC in searching & insertion

```
class Node {
    bool isEnd
    map < char, Node > ch
                key   value
}
```

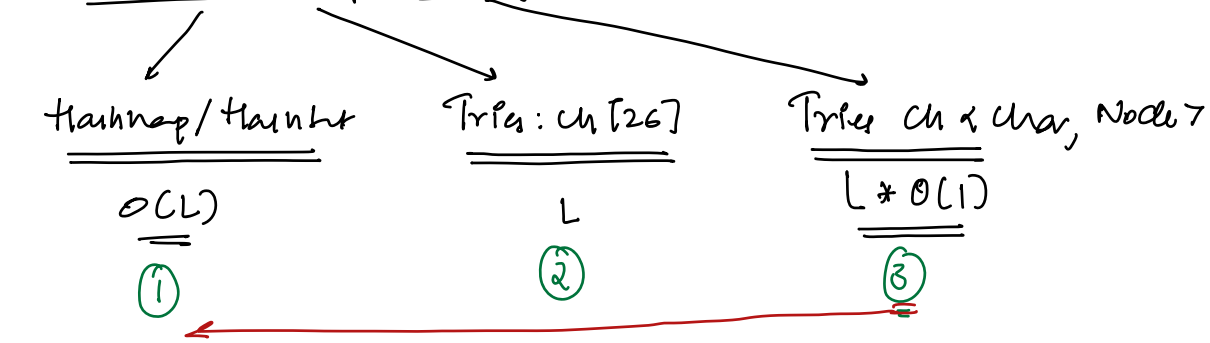
damp ✓
data ✓
take ✓
taken ✓
ant ✓
date ✓

Q) dat

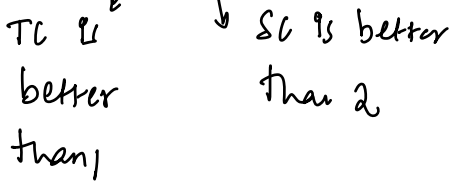


→ { Insert / Search a particular word from a dictnary of word in a optimized manner }

→ Insert a string of length of L



Best TC : 2 3 1



TC :

- Insert N words of len L → N * L * O(1)
- Query Q words of len L → Q * L * O(1)

SC : → Insert N words of len L → N * L * { worst case : for every character we will create a new node O(1) }

SC : O(N * L)

Code :

```
Class Node {
    bool IsEnd;
    Hashmap<char, Node> hm;
    Node() {
        IsEnd = False
    }
}
```

```
Node root = new Node();
Read N;
for(i=1; i<=N; i++) {
    Read word
    add(word, root)
}
Read Q;
for(i=1; i<=Q; i++) {
    Read word
    if( find (word, root))
        print (Exist)
    else {
        print ("Not present")
    }
}
```

```
void add (String str, Node r) {
    int n = str.length();
    for(i=0; i<n; i++) {
        // try to insert str[i]
        char ch = str[i]
        if( ch is not present in r.hm) {
            Node t = new Node();
            r.hm.insert(ch, t)   // char, node
            r = r.hm[ch] or r=t
                  t
        }
        else {
            // get ref of ch in r.hm
            r = r.hm[ch]
        }
    }
    // All charaters are inserted & we are in last node
    r.IsEnd = True;
}
```

```
bool find ( String str, Node r) {
    int n = str.length()
    for(i=0; i<n; i=i+1) {
        char ch = str[i]
        if ( ch is not in present in r.hm) {
            return False
        }
        else
            r = r.hm[ch]
    }
    // we are done with query word
    return r.IsEnd
}
```