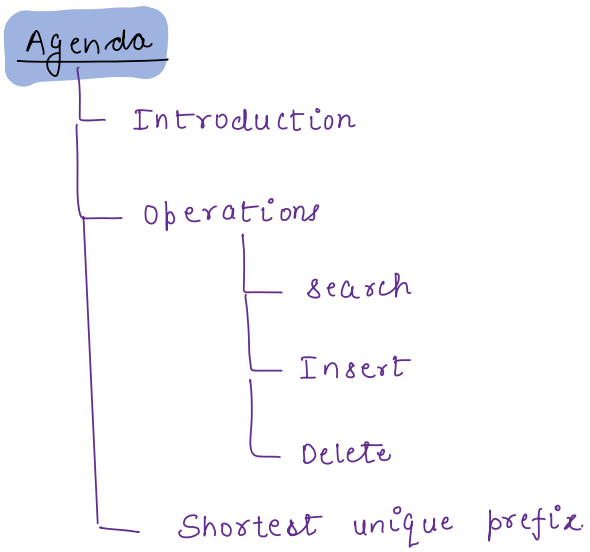
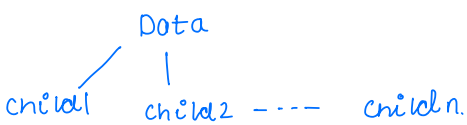
Lecture :- Tries - 1



Trie: Retrieval → fetch. search.

- DS that stores data from top to bottom.
- characters
- bits [Next class]

Structure

Trees [Binary]	Tries
	

`class Node {`
`int data;`
`Node left;`
`Node right;`
`}`
	`class Node {`
	`char data;`
	`Node[] children;`
	`Node(x) {`
	`data = x;`
	`}`
	`}`

In words

try

art

play

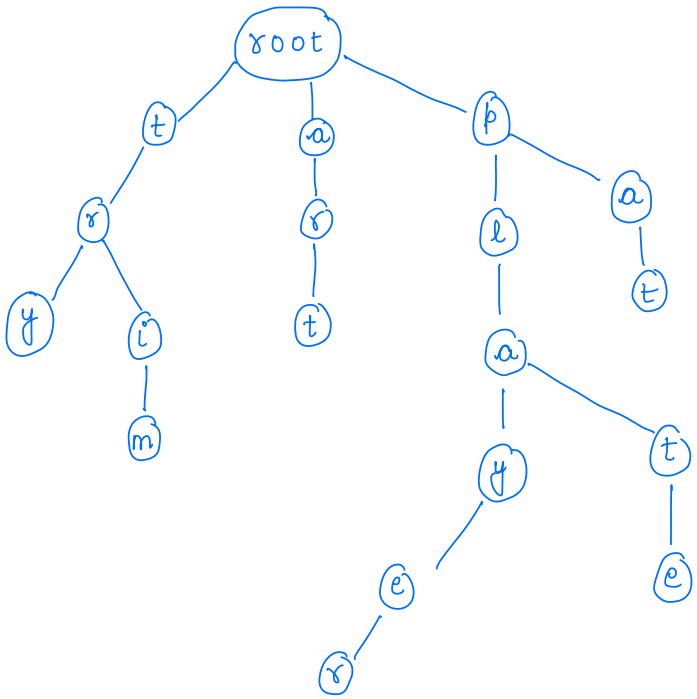
trim

plate

pat

player.

trim.



Qu: Check valid words.

isvalid(plote) = true

isvalid(srk) = false

isvalid(pla) = false

isvalid(player) = true

isvalid(aye) = false

Map: TC: o(len)

SC: o(len)

↑

1 word

n words:

TC: o(n*len)

SC: o(n*len).

try
art
play
trim
plate
pat
player.

Idea! Travel complete word in trie and check for leaf.

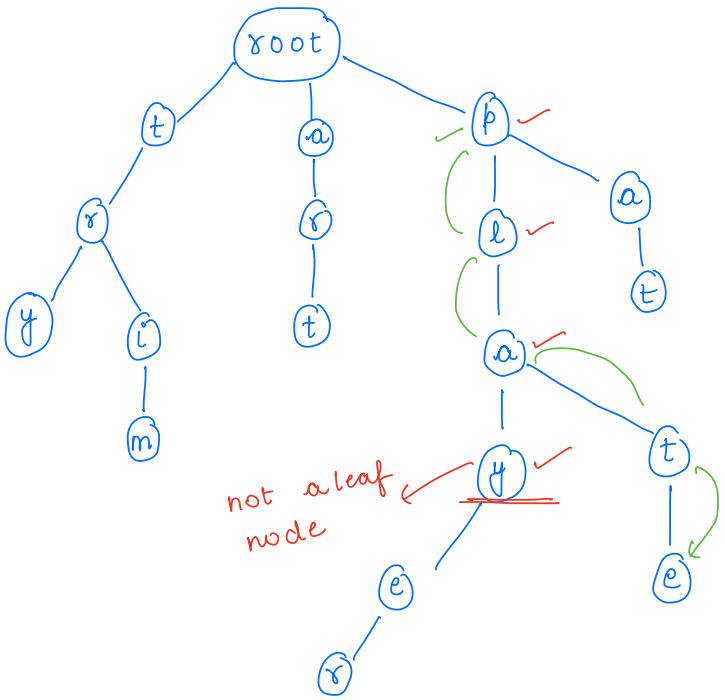


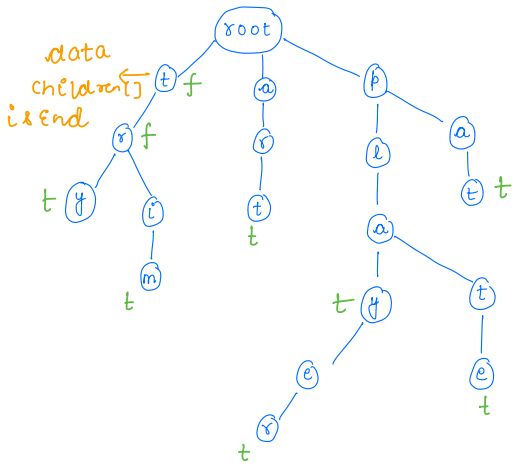
plate :- true

play → Idea does not work here.

Idea2: Use a flag to denote end of word.

```
class Node {
    char data;
    Node[] children;
    boolean isEnd;

    Node(x) {
        data = x;
        children = new Node[26];
        isEnd = false;
    }
}
```



Modified Idea:

Travel complete word in trie and check whether the last character is end of word or not?

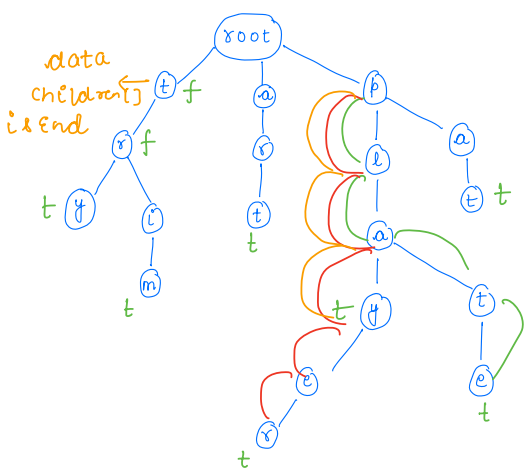


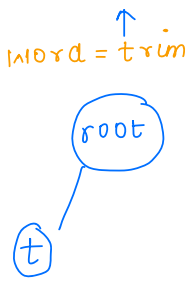
plate :- true

player :- true

play :- true

pla :-

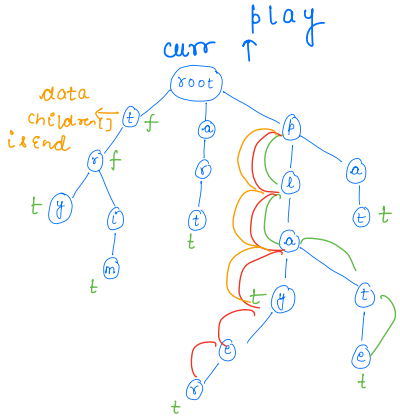
Insertion [Insert a word in a trie]




```
void insert ( root , word) {

        curr = root;
        n = word.length;
       for (i=0; i < n; i++) {
            ch = word.charAt(i);
            idx = ch - 'a';
            if ( curr . children [idx] == null) {
                 Node nn = new Node( ch);
                 curr . children[ idx ] = nn;
            }
            curr = curr.children[ idx ];
       }
      curr. isEnd = true;
}
```

search for a word whether it is valid or not?




```
boolean search (root, word) {
    curr = root;
    n = word.length;
    for(i=0; i<n; i++) {
        ch = word.charAt(i);
        idx = ch - 'a';
        if(curr.children[idx] == null) {
            return false;
        }
        curr = curr.children[idx];
    }
    return curr.isEnd;
}
```

Qu

Frequency of words

trim

try

art

play

trim

trimmer

plate

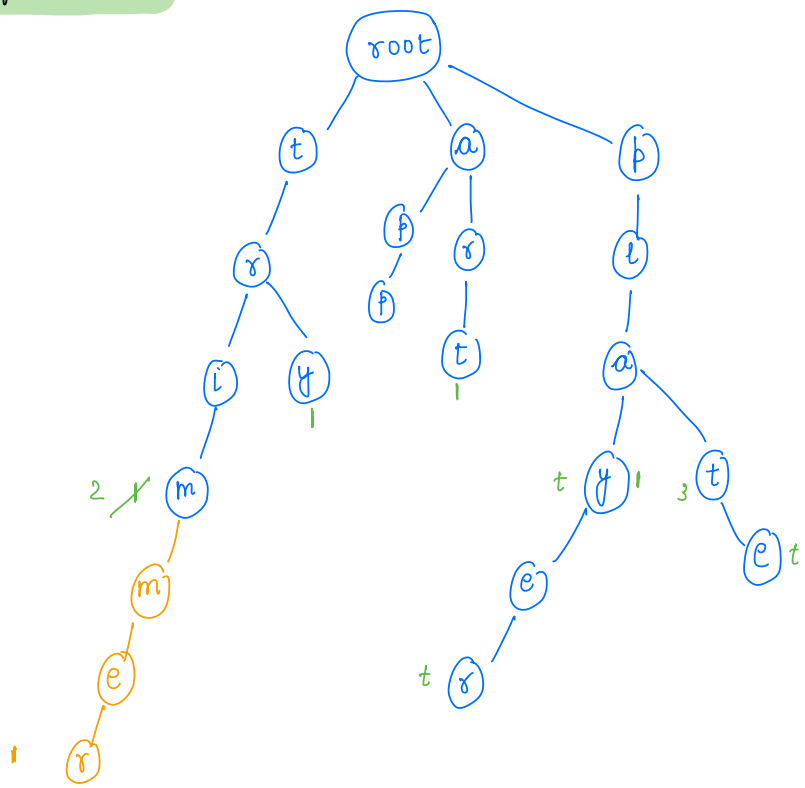
player

play

art

play

app



frequency (trim) = 2

freq (art) = 2

freq (play) =

freq (try) =

Structure:

```
class Node {
    char data;
    Node[] children;
    boolean isEnd;
    int freq;
    Node(x) {
        data = x;
        children = new Node[26];
        isEnd = false;
        freq = 0;
    }
}
```

Insert

```
void insert(root, word) {
    curr = root;
    n = word.length;
    for(i=0; i<n; i++) {
        ch = word.charAt(i);
        idx = ch - 'a';
        if(curr.children[idx] == null) {
            Node nn = new Node(ch);
            curr.children[idx] = nn;
        }
        curr = curr.children[idx];
    }
    curr.isEnd = true;
    curr.freq += 1;
}
```

find frequency

```
int frequency ( root , word) {
        curr = root;
        n = word.length;
        for(i=0; i<n; i++) {
            ch = word.charAt(i);
            idx = ch - 'a';
            if( curr.children[idx] == null) {
                return 0;
            }
            curr = curr.children[idx];
        }
        if( curr.isEnd) {
            return curr.freq;  ⟹  return curr.freq
        }
        return 0;
}
```

TC : O(len) = 1 word.

Break: 8:19 - 8:30 AM

* **Deletion in a tree**

trim

try

art

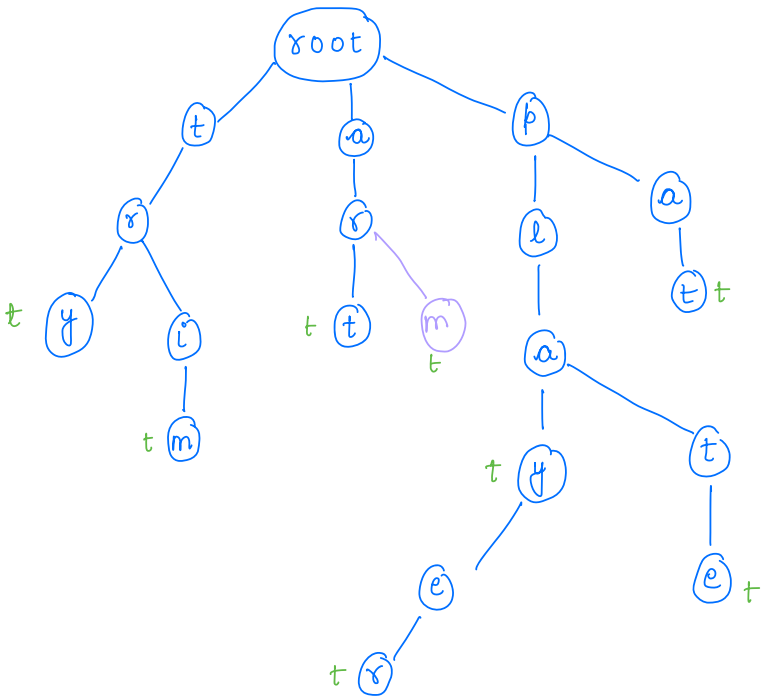
play

plate

player

arm

pot



delete(trim) =

delete(play) =

Ideal: Traverse complete word and mark end of word = false.

└ Space is wasted

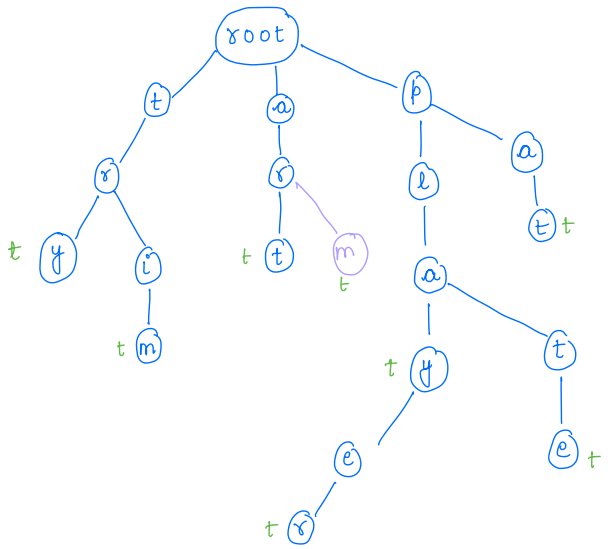
Idea2

Reach end of word.

while coming back,

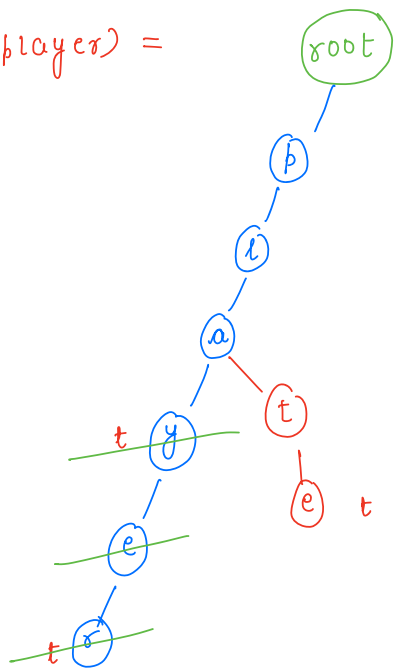
if leaf node – delete

else – stop



delete(trim) = works fine

delete(player) =



Modified idea

Reach end of word —

while coming back —

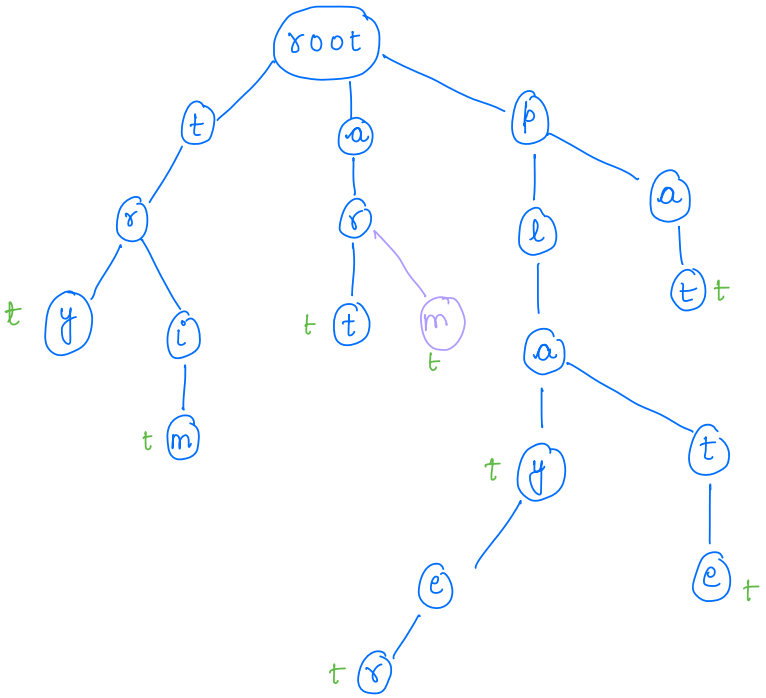
if leaf node && isEnd = false.

delete it

else

stop

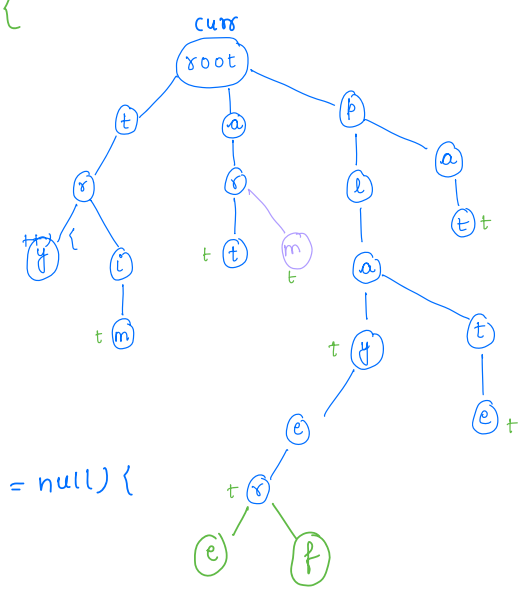
Idea3: find last | latest node that can't be deleted.



delete(player)

Code:

```
void deleteWord(root, word) {
    curr = root;
    lastNode = null;
    deleteChar = ' ';
    n = word.length();
    for(i=0; i<n; i++) {
        cnt = 0;
        for(j=0; j<26; j++) {
            if( curr.children[j] != null) {
                cnt++;
            }
        }
        if( cnt > 1  ||  curr.isEnd == true) {
            lastNode = curr;
            deleteChar = word.charAt(i);
        }
        idx = word.charAt(i) - 'a';
        curr = curr.children[idx];
    }
    curr.isEnd = false;
    // find whether last character is a leaf
       node or not?
    cnt = 0;
    for(j=0; j<26; j++) {
        if( curr.children[j] != null) {
            cnt++;
        }
    }
    if(count > 0) { return; };
    lastNode.children[ deleteChar - 'a' ] = null;
}
```



SC: O(1)

Dry run:

delete (player) ⟶

curr = root

deletechar = ' ';

i = 0 , p ⟶ curr = root
last = null
dc = ''

last = root

dc = p
curr = (p)

i = 1 , l curr = p
last = p
dc = l
curr = (l)

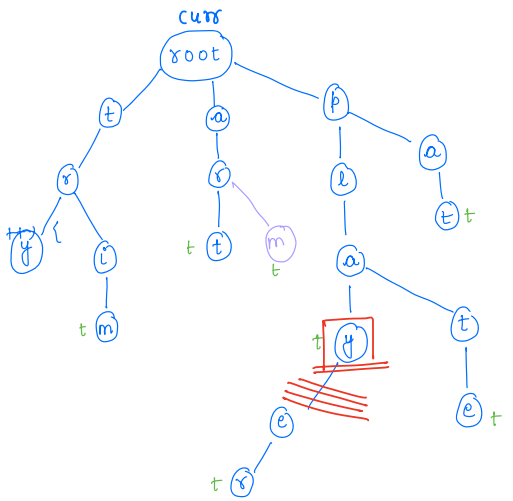
i = 2, a curr = l

curr = (a)

i = 3, y : curr = (a)
lastnode = (a)
dc = y.
curr = (y)

i = 4 e : curr = y
lastnode = (y)
dc = 'e'
curr = (e)

i = 5 r : curr = (e)
curr = (r)




```
for(i=0; i<n; i++) {
    cnt = 0;
    for(j=0; j<26; j++) {
        if( curr.children[j] != null)
            cnt++;
    }
}
if( cnt > 1 || curr.isEnd == true) {
    lastNode = curr;
    deletechar = word.charAt(i)
}
idx = word.charAt(i) - 'a';
curr = curr.children[idx];
}
```

lastNode.children [deletechar - 'a'] = null;

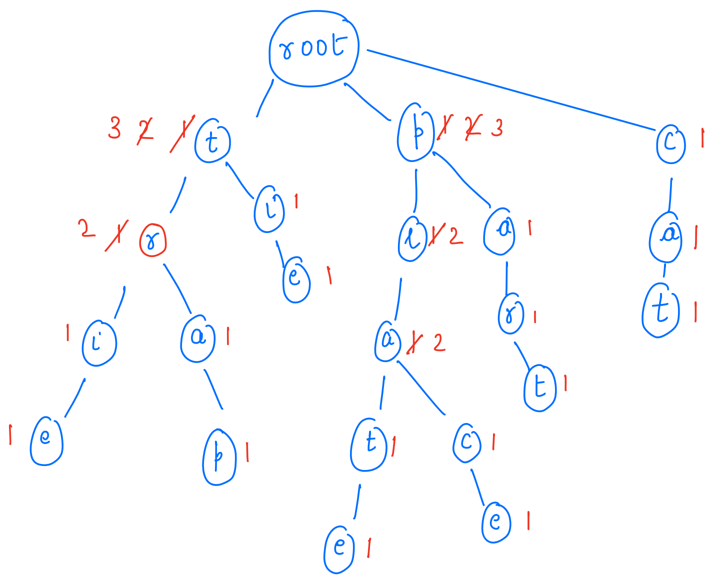
(y) . [e] = null

Qu **Shortest unique prefix**

[trie, trap, plate, cat, part, place, tie]

↓

tri, tra, plat, c, pa, plac, ti



```
class Node {
    char data;
    Node[] children;
    boolean isEnd;
    int pc;
    Node(x) {
        data = x;
        children = new Node[26];
        isEnd = false;
        pc = 0;
    }
}
```

Insert

```
void insert(root, word) {
    curr = root;
    n = word.length;
    for(i=0; i<n; i++) {
        ch = word.charAt(i);
        idx = ch - 'a';
        if(curr.children[idx] == null) {
            Node nn = new Node(ch);
            curr.children[idx] = nn;
        }
        curr = curr.children[idx];
        curr.pf += 1;
    }
    curr.isEnd = true;
}
```

search shortest unique prefix

```
String shortestUniquePrefix(root, word) {

    curr = root;
    n = word.length;
    ans = "";
    for(i=0; i<n; i++) {

        ch = word.charAt(i);

        idx = ch - 'a';

        if(curr.children[idx] == null) {

            return "" / null;  → Depends on question
        }
        if(curr.children[idx].pf > 1) {
            ans += ch;
        } else {    → curr.children[idx].pf == 1
            ans += ch;
            return ans;
        }

        curr = curr.children[idx];
    }

    return ans;

}
```

Thankyou