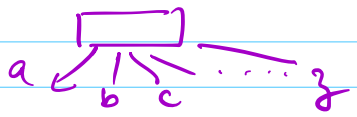
9th Aug 2023

# TRIES - 1 (TRYS)

**Tries →** Tree like DS that stores data from top to bottom.

**Trie of characters (a-z)**



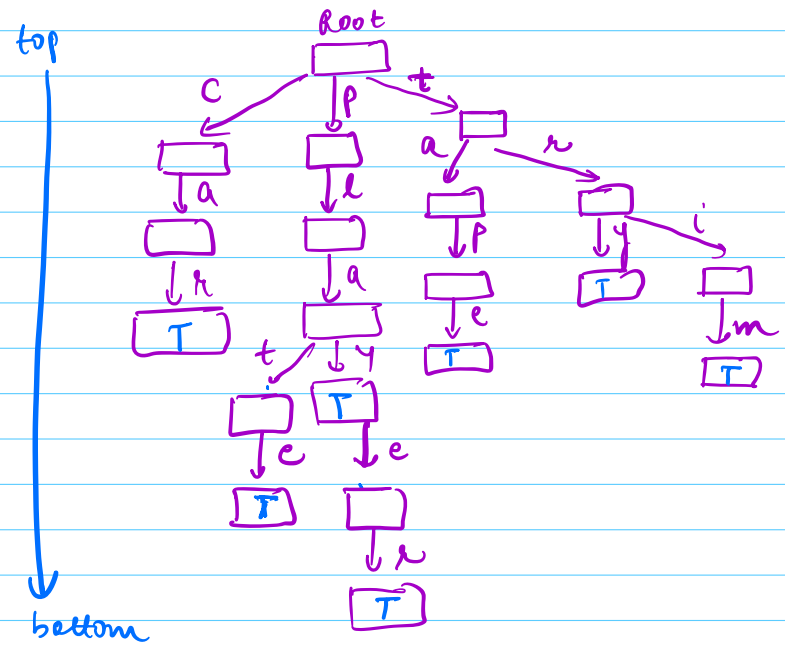

```
class Node {
    Node[26] child   // a -> 0
                     // b -> 1
                     // :
    boolean isEnd

}
// initially if i child[i] = null
    default isEnd = false.
```

**Real life use cases**

1. Auto complete
2. Word search
3. Spelling checker
4. Internet routing
5. Encryption.

**List of words**

1) tape
2) try
3) trie
4) play
5) plate
6) car
7) player.



## Search

check("cat") ⇒ false (not present)

If all characters of the word are not travelled ⇒ not present.

check("try") ⇒ true.

check("tri") ⇒ false

If all characters of the word are travelled then, it is either present OR it is a prefix of any word present.

```
boolean search(root, word) {
    temp = root;

    for i → 0 to (word.length - 1) {

        ch = word[i]
        if (temp.child[ch - 'a'] == null)
            return false;

        temp = temp.child[ch - 'a'];
    }

    return temp.isEnd;
}
```

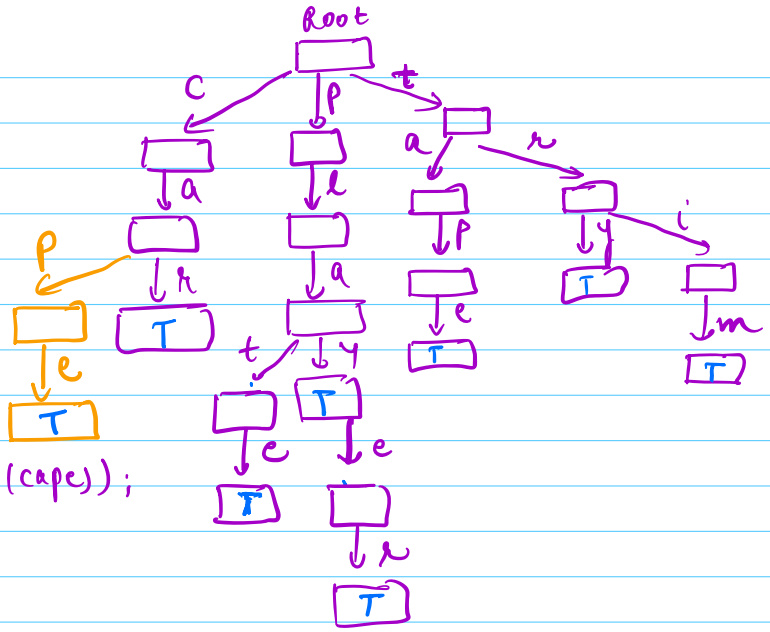
TC: O(word length)

Hashset → complete word is required to check.

Trie → can check as we type characters → faster.

## List of words

1) tape
2) try
3) trim
4) play
5) plate
6) car
7) player
8) cape (insert (cape));




```
void insert (root, word) {
    temp = root;

    for i → 0 to (word.length-1) {

        ch = word[i]
        if (temp.child[ch-'a'] == null)
            temp.child[ch-'a'] = new Node()

        temp = temp.child[ch-'a'];
    }

    temp.isEnd = true; // temp.freq++;
}
```

TC: O(N * word length) , SC: O(N * word length)

Keep track if a word is inserted multiple times. i.e. freq

~~boolean isEnd~~ → int freq.

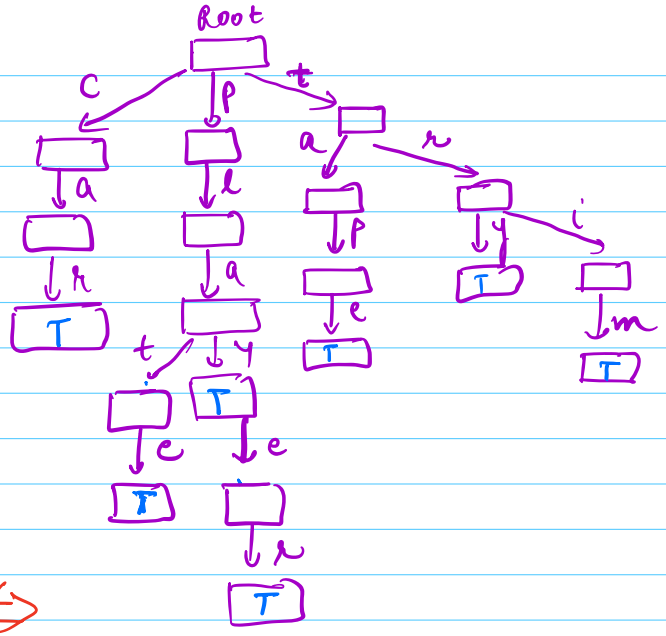
```
class Dict {
    Node root;
    Dict () {
        root = new Node();
    } public Node getRoot(){
        return this.root;
    }
}
Dict d = new Dict();
d.getRoot();
insert(root, "x");
```

8:23 am IST

## Deletion

delete("trim")

Sol1 → Travel to the end of the word & update is End to false.



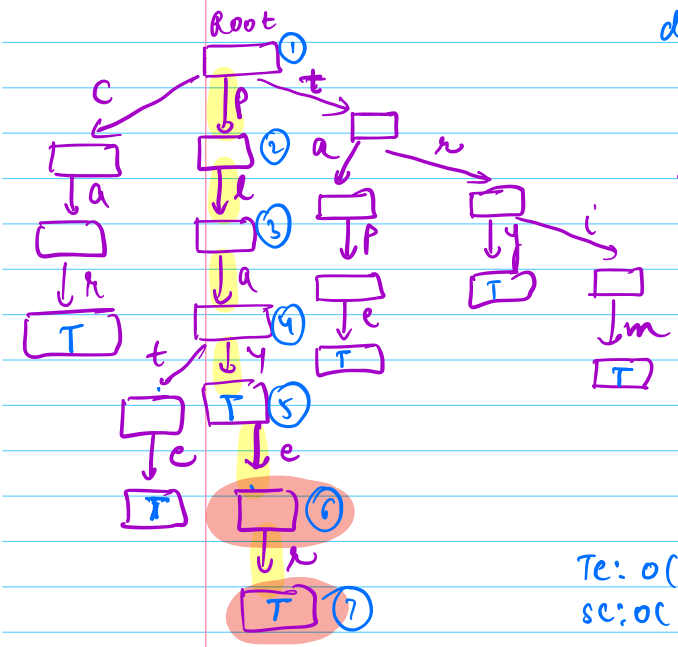

Drawbacks:

a> Travel complete ⇏ valid prefix of any input

b> Space wastage

Sol2 → delete un-utilized nodes

↳ only leaf node.




delete (trim)

1> Travel all nodes & insert in the stack.

2> Update is End = false for last node.

3> Pop elements from stack, if leaf node & is End = false

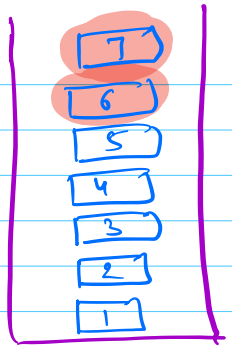
⇒ delete it.

↳ ∀ i child[i] = null.

TC: O(word length)

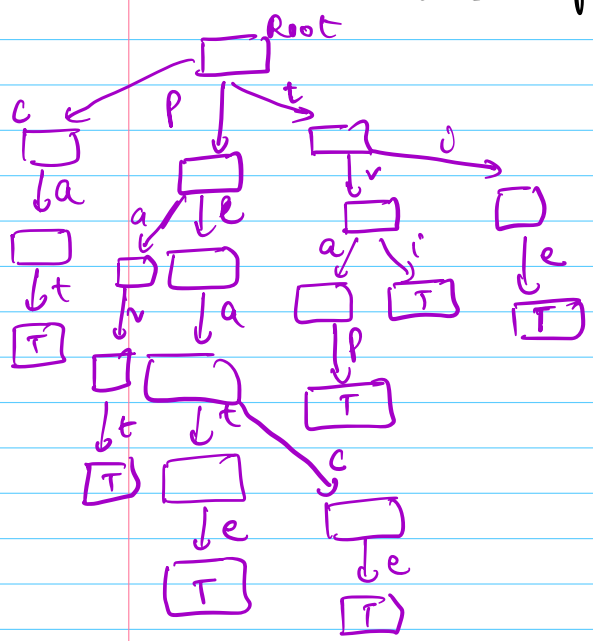
SC: O(word length)

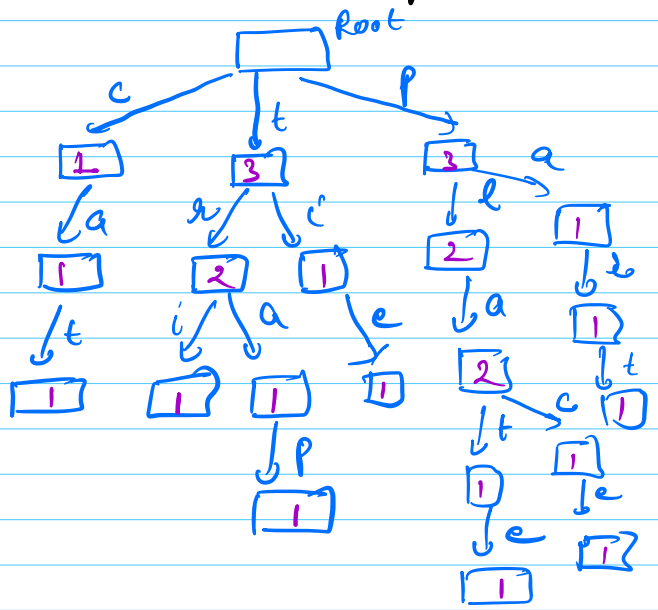
delete player




Q :- Given a list of words, find shortest prefix of each word that uniquely defines the word.
(no word is a prefix of another word)

Eg → ["tri", "trap", "plate", "cat", "part", "player", "tic"]
tri tra plat c pa play ti



1) # nodes → store # times that node is visited.
(while inserting words in the trie)

2) Travel the word till a node with freq = 1 is reached

TC: O(N * word length)
SC: O(N * word length)

```
void insert (root, word) {
    temp = root;

    for i → 0 to (word.length - 1) {

        ch = word[i]
        if (temp.child[ch - 'a'] == null)
            temp.child[ch - 'a'] = new Node()

        temp = temp.child[ch - 'a'];
        temp.freq++;
    }

}
```

```
String search ( root, word ) {
    temp = root;
    String ans = "";
    for i -> 0 to (word.length - 1) {

        ch = word[i]
        temp = temp.child [ch - 'a'];
        ans += ch;
        if (temp.freq == -1)
            return ans;
    }


}

class Node {

    Node [26] child;
    int freq;

}
```