① Magic dictionary (SD) → LLD
② Search Suggestion System (SD)
③ Stream of characters
④ Prefix & Suffix Search Manacher's Algo

☆ Bit Manipulation Basics

① Magic dictionary

Change exactly one char to match any other word in dict

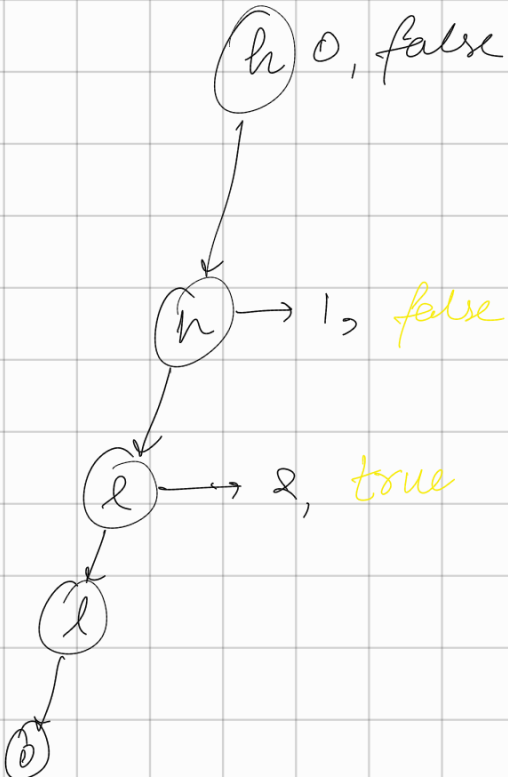
[hello, leetcode]

- ✗ hello — leetcoded → ✗
- ✓ hillo — leetcode → ✓
- ✗ hell — leetcod → ✗

Search Parameters { String word , int idx , Node root or curr , boolean change }

Eg: for

	h	h	l	l	o
idx	0	1	2	3	4
change	false				
prefix	✓				



If character needs to be replaced

Case 1: already one replace
→ return false

Case 2: first change
→ make change & move ahead checking for all children

for all char a to z

if char → replace [curr → contains(ch)]

if change == true

return false

else

return {idx+1, curr.contains(ch), true}

* Character cannot be replaced by itself. *

CODE:

```
search(word, idx, curr, changed){
  if(idx == word.length()){
    if(changed == true && curr.getEnd() == true)
      return true;
    return false;
  }

  char ch = word.charAt(idx);
  if(changed == true) return false;

  for(char c='a'; c<='z'; c++){
    if(c != ch && curr.contains(c) && search(word, idx+1,
                                    curr.get(c), true))
      return true;
  }
  return false
}
```

② Search Suggestion System { System Design }

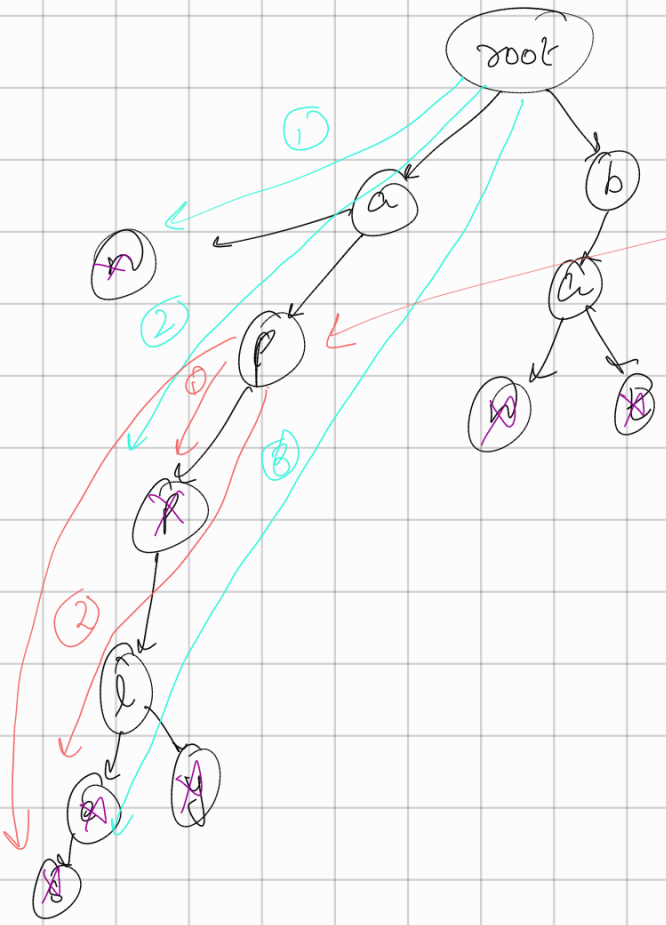
given an array of strings "products" & a string "search word"
Implement SSS

Eg: products = ["mobile", "mouse", "moneypot", "monitor", "mousepad"]
search word = "mouse"

Output: m → mobile, moneypot, monitor
mo → "
mou → mouse, mousepad
mous → "
mouse → "

☆ Practically, the child nodes with highest prof sum can occur first

Eg: app, apple, apples, apps, apply, ban, bat, an



when searching for (a)
→ an, app, apple

when searching for (ap)
→ app, apple, apples

here
apples < apply
because lexicographical order
NOT SORTED ORDER

Similarly on searching for "appl"

→ apple , apples , apply

↳ ∵ lexicographically e < y
& length is second preference

Preorder DFS on Tries is by default lexicographical.

CODE:

```
public void DFS (Node root, String ssf, List<String> ans, int k){
    if(ans.size()==k) return;

    if(root.getEnd()==true)
        ans.add(ssf)

    for(char c='a'; c<='z'; c++){
        if(root.containing(c)==true){
            DFS(root.getch), ssf+c, ans, k);
        }
    }
}
```

Suggested products {

insert

new AL - - -

call DFS for all substrings. - -

return ans;

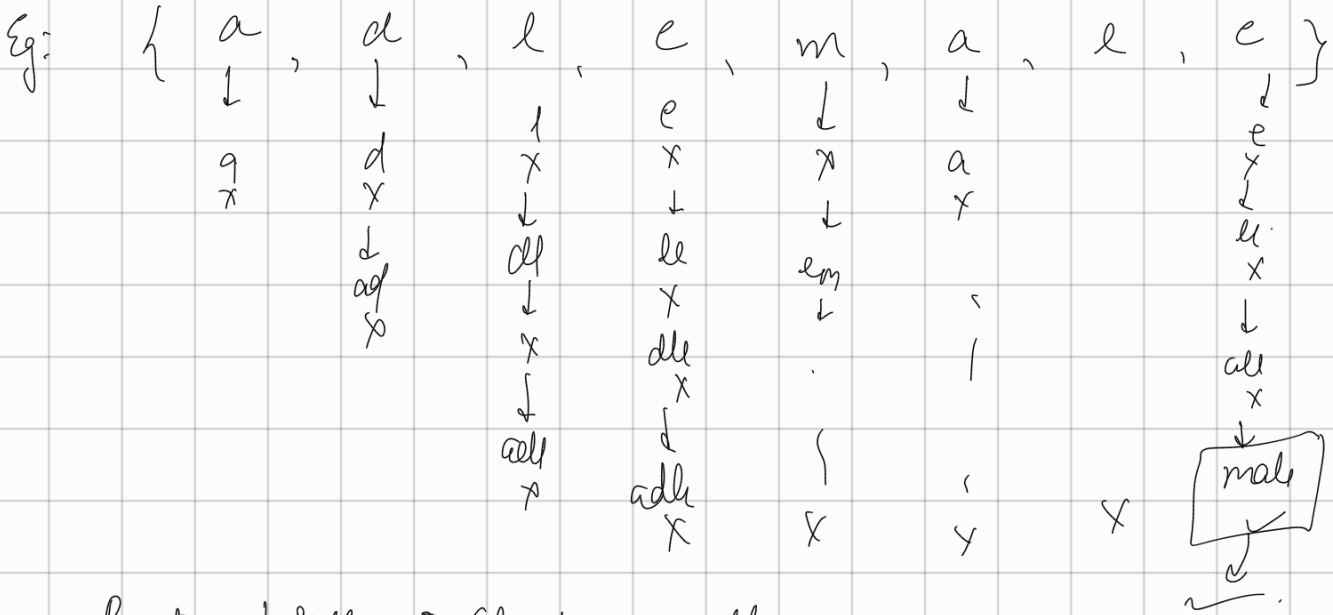
}

③ Stream of characters

{"apple", "male"} → Dictionary

Online ⇒ Stream of characters

Eg: { a , d , l , e , m , a , l , e }



Brute force → Check for all

TC X

Solution: Insert (R to L) → reverse word

Search (R to L) → Suffix ⇒ Prefix

for apple → insert ellpa

for male → insert elam

& Search using StringBuilder

(4) Palindrome pairs

given a list of unique words return all the pairs of the distinct indices (i, j) in given list, so that the concatenation of the two words is a palindrome

Eg: words = ["abcd", "dcba", "lls", "s", "sssll"]

Output = [[0, 1], [1, 0], [3, 2], [2, 4]]

- [0, 1] → abcddcba
- [1, 0] → dcbaabcd
- [3, 2] → llssll
- [2, 4] → sllssssll

Approach: for x+y = palindrome → 3 case

Case (1) x = reverse(y) → abcd = reverse(dcba)

both x+y & y+x Palindrome

Always Palindrome

Case (2) Suffix of y = Reverse of x

Eg: x = abc, y = zxcba

reverse suffix

remaining characters should also be a palindrome

Case (3) Prefix of x = Reverse of y

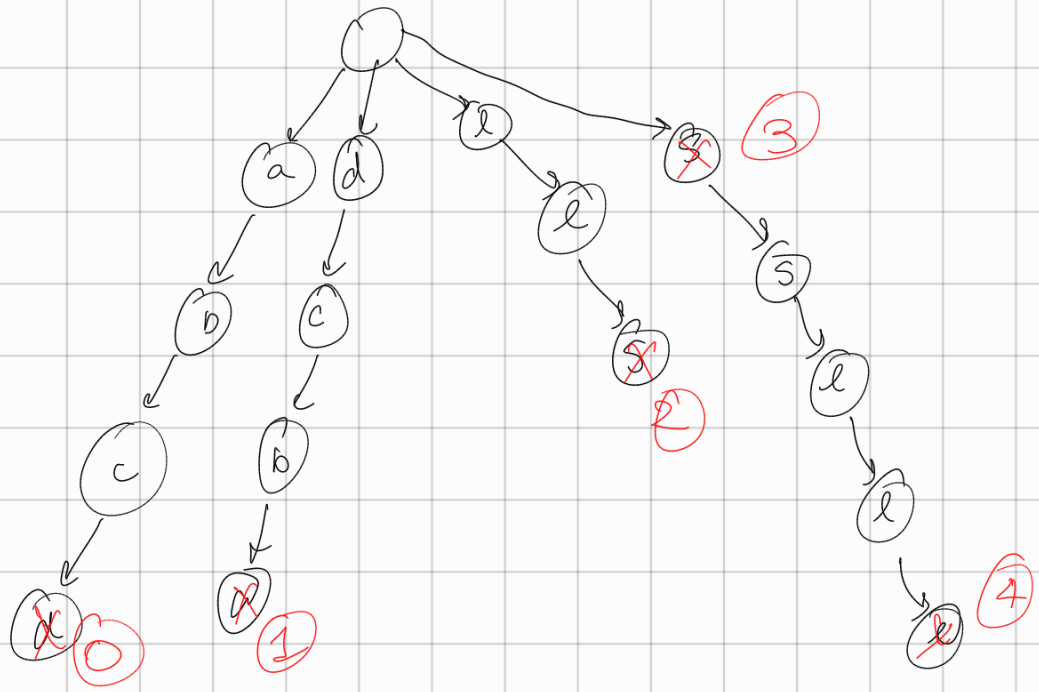
Eg: x = llss, y = sll

prefix reverse

remaining characters should also be a palindrome.

In Case (2) & (3) remaining part should also be a palindrome

Trie



(*) → is End = true → instead of is End, mark index of string for not is End store -1.

Queries: for $x = abcd$, $x + y$ = Palindrome

Case ① → y = reverse of x

Case ② → reverse suffix of x & rem. palin

Case ③ → reverse prefix of y & rem palin

OR

for $x + y$ = Palin, make y = corr idx word & check for x