Trie

208. Implement Trie (Prefix Tree)

Medium 6798 90 Add to List Share

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

HashSet

String → hashcode

O(L) L → length

Insertion, Search, Delete

worst → O(N*L) {for N strings}
case

HashSet is not a good DS → coz of inc- collision
when the set of Strings is huge.

Trie

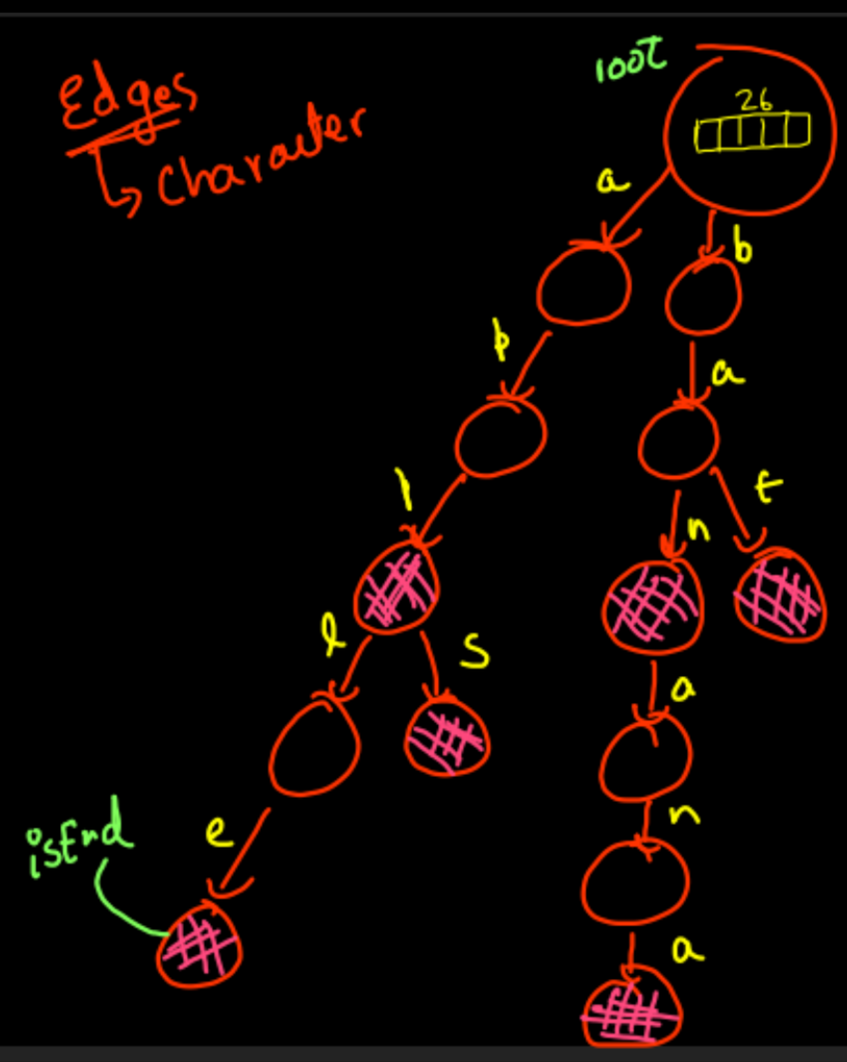
- insertion
- deletion
- find

} O(L)

→ We can also search for prefixes.

Drawback is that it uses a lot of extra space.

* Trie will always have a root node depicting that there is one string i.e empty string.



insert ("apple")
insert ("banana")
insert ("bat")
insert ("ban")
insert ("app")
insert ("apps")
Search ("apple")
↳ travel then isEnd
f Search ("bank")
f search ("ap")
T brefix ("ap")

```java
public static class Node {

    private Node[] children = new Node[26];
    private boolean isEnd = false;

    public boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    public Node get(char ch) {
        return children[ch-'a'];
    }

    public void set(char ch) {
        children[ch-'a'] = new Node();
    }

    public boolean getEnd() {
        return isEnd;
    }

    public void setEnd() {
        isEnd = true;
    }
}
```

```java
Node root;
public Trie() {
    root = new Node();
}
```

```java
public void insert(String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false)
            curr.set(ch);

        curr = curr.get(ch);
    }

    curr.setEnd();
}
```

```java
public boolean search(String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) return false;
        curr = curr.get(ch);
    }

    return curr.getEnd();
}
```

```java
public boolean startsWith(String prefix) {
    Node curr = root;

    for(int i=0;i<prefix.length();i++) {
        char ch = prefix.charAt(i);
        if(curr.contains(ch) == false) return false;
        curr = curr.get(ch);
    }

    return true;
}
```

Worst Case:
Insertion, Deletion, Search for 1 String } O(L) Tc

SC (Worst case for entire trie) O(N*L)

211. Design Add and Search Words Data Structure

Medium 4752 198 Add to List Share

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the `WordDictionary` class:

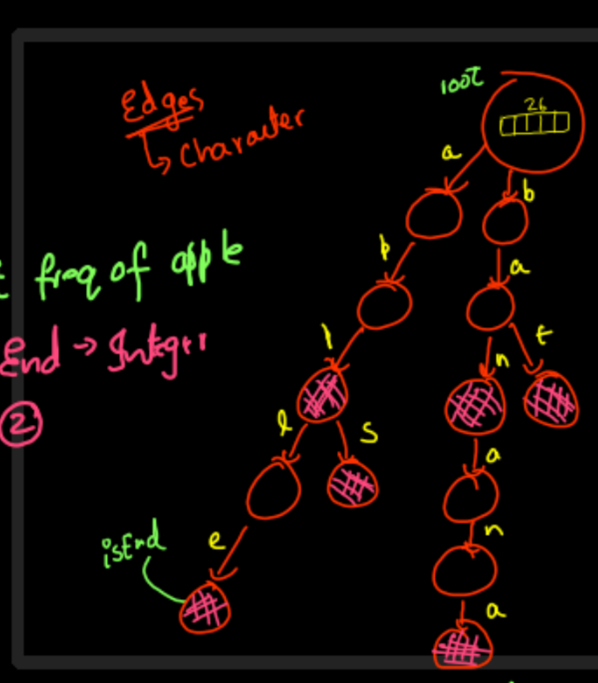
- `WordDictionary()` Initializes the object.
- `void addWord(word)` Adds `word` to the data structure, it can be matched later.
- `bool search(word)` Returns `true` if there is any string in the data structure that matches `word` or `false` otherwise. `word` may contain dots `'.'` where dots can be matched with any letter.

Logic for Node and adding/insert into the trie will remain the same.

```java
public boolean search(String word, int idx, Node curr) {
    if(idx == word.length()) {
        return curr.getEnd();
    }

    char ch = word.charAt(idx);
    if(ch != '.') {
        if(curr.contains(ch) == false) return false;
        return search(word,idx + 1,curr.get(ch));
    }

    for(char chn='a';chn<='z';chn++) {
        if(curr.contains(chn) == false) continue;
        if(search(word,idx + 1, curr.get(chn)) == true) return true;

    }

    return false;
}

public boolean search(String word) {
    return search(word,0,root);
}
```

Only search is diff.
WC TC : 26^L all dots

Implement Trie- II (Codestudio (N)



count freq of apple
↳ isEnd → Integer
of K②

insert ("apple")
insert ("banana")
insert ("bat")
insert ("ban")
insert ("app")
insert ("apps")
Search ("apple")
↳ travel then isEnd
Search ("bank")
search ("ap")
Prefix ("ap")

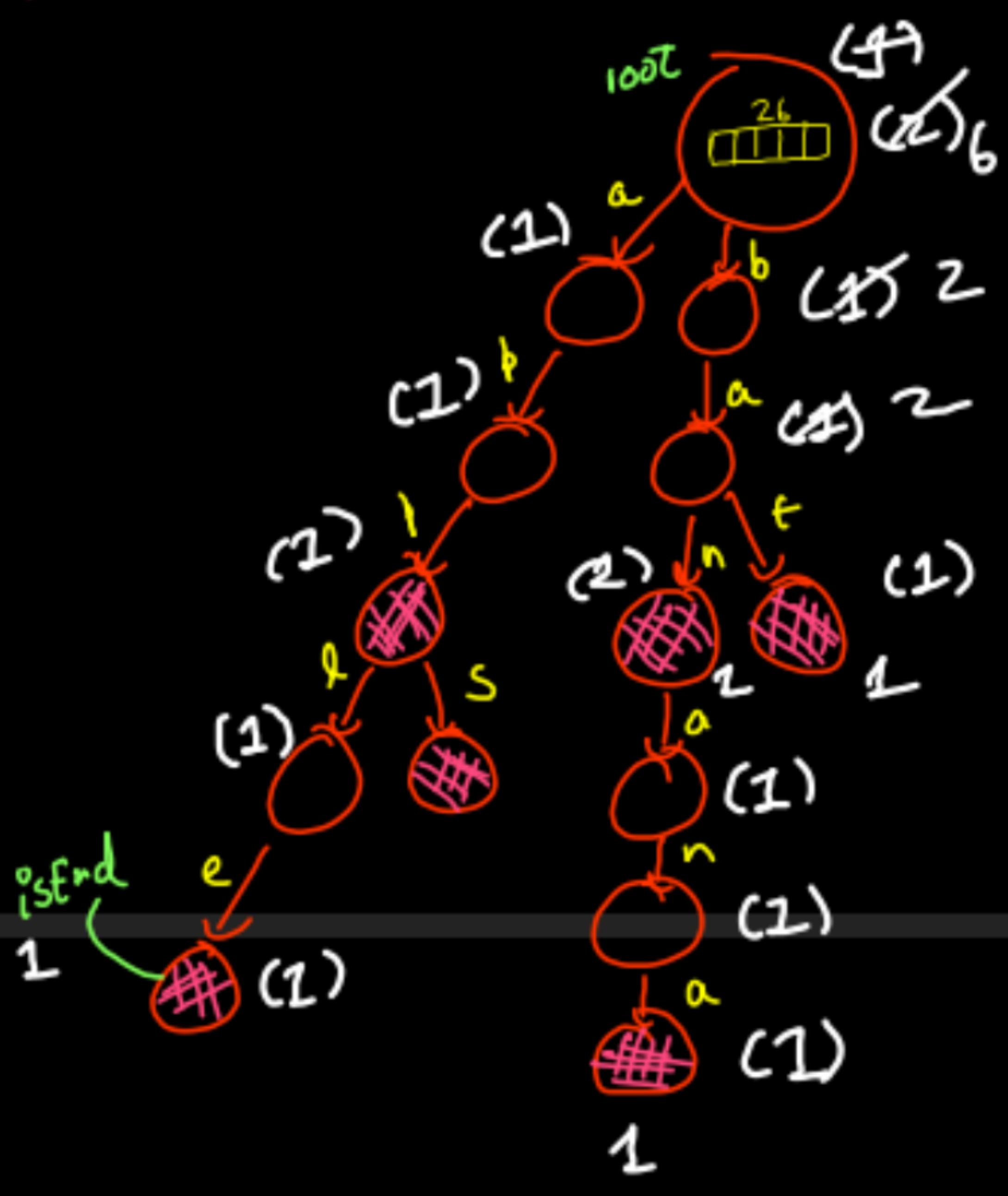
count no of strings starting with 'ap'
↳ Reach ap → apply DFS & count += isEnd

Delete "ban"

freq ≥ 2 → isEnd--

freq = 1
- Subtree → this means it is a prefix. So isEnd = 0
- No subtree → delete it

Optimization (Maintain extra variable)



at every node, we have a variable named prefixCount which tells us the number of words that are there with that prefix.

1 → isEnd
(1) → prefix count } in diag

prefix count of root will tell us the total no of words in the trie.

```java
public static class Node {
    private int isEnd = 0;
    private int prefixCount = 0;
    private Node[] children = new Node[26];

    public boolean contains(char ch) {
        if(children[ch-'a'] == null) return false;
        return true;
    }

    public Node get(char ch) {
        return children[ch-'a'];
    }

    public void set(char ch) {
        children[ch-'a'] = new Node();
    }

    public int getEndFreq() {
        return isEnd;
    }

    public int getPrefCount() {
        return prefixCount;
    }

    public void increaseEndFreq() {
        isEnd++;
    }

    public void decreaseEndFreq() {
        isEnd--;
    }

    public void increasePrefCount() {
        prefixCount++;
    }

    public void decreasePrefCount() {
        prefixCount--;
    }
```

```java
Node root;

public Trie() {
    // Write your code here.
    root = new Node();
}

public void insert(String word) {
    // Write your code here.
    Node curr = root;
    for(int i=0;i<word.length();i++) {
        curr.increasePrefCount();
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }

        curr = curr.get(ch);
    }
    curr.increasePrefCount();
    curr.increaseEndFreq();
}

public int countWordsEqualTo(String word) {
    Node curr = root;
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            return 0;
        }

        curr = curr.get(ch);
    }

    return curr.getEndFreq();
}
```

```java
public int countWordsStartingWith(String word) {
    // Write your code here
    Node curr = root;
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            return 0;
        }
        curr = curr.get(ch);
    }

    return curr.getPrefCount();
}

public void erase(String word) {
    if(countWordsEqualTo(word) <= 0) {
        return;
    }

    Node curr = root;
    for(int i=0;i<word.length();i++) {
        curr.decreasePrefCount();
        char ch = word.charAt(i);
        curr = curr.get(ch);
    }

    curr.decreasePrefCount();
    curr.decreaseEndFreq();
}
```

Weighted Prefix Search

677. Map Sum Pairs

Medium 1203 123 Add to List Share

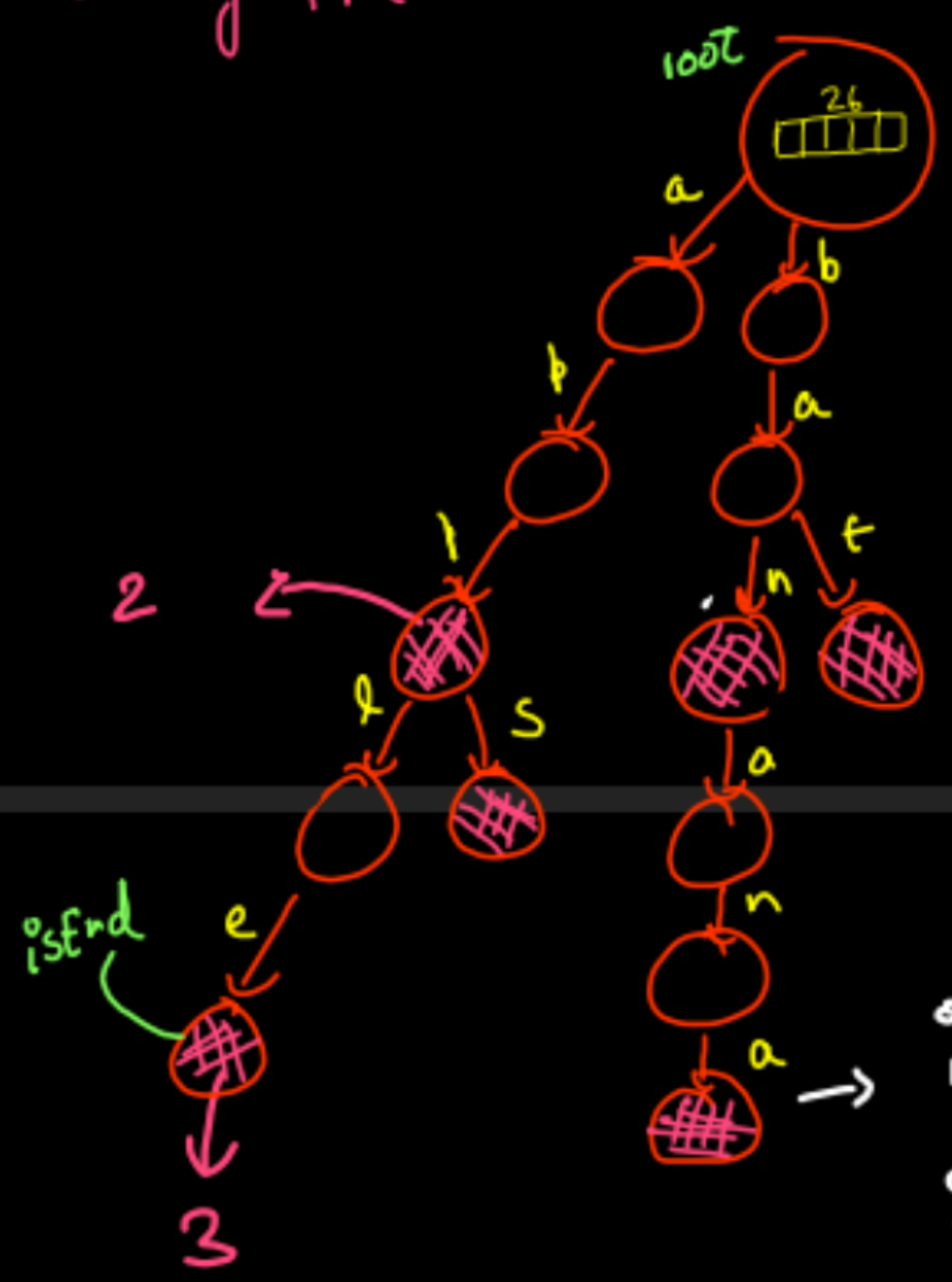
Design a map that allows you to do the following:

- Maps a string key to a given value.
- Returns the sum of the values that have a key with a prefix equal to a given string.

Implement the `MapSum` class:

- `MapSum()` Initializes the `MapSum` object.
- `void insert(String key, int val)` Inserts the `key-val` pair into the map. If the `key` already existed, the original `key-value` pair will be overridden to the new one.
- `int sum(string prefix)` Returns the sum of all the pairs' value whose `key` starts with the `prefix`.

Kind of designing hashmap using trie.



apple - 3

app - 2

Σ of all values with prefix ap

↳ store Σ of values in prefix count.

→ instead of storing freq, store the value

Using Dfs

```
static class Node {
    private Node[] children = new Node[26];
    private int val = 0;

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    void setVal(int val) {
        this.val = val;
    }

    int getVal() {
        return val;
    }
}
```

```
public void insert(String key, int val) {
    Node curr = root;

    for(int i=0;i<key.length();i++) {
        char ch = key.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }

        curr = curr.get(ch);
    }

    curr.val = val;
}
```

```
private int DFS(Node curr) {
    int ans = 0;

    for(char ch='a';ch<='z';ch++) {
        if(curr.contains(ch) == true) {
            ans += DFS(curr.get(ch));
        }
    }

    ans += curr.getVal();
    return ans;
}

public int sum(String prefix) {
    Node curr = root;

    for(int i=0;i<prefix.length();i++) {
        char ch = prefix.charAt(i);
        if(curr.contains(ch) == false) return 0;
        curr = curr.get(ch);
    }

    return DFS(curr);
}
```

Using Prefix at each node

```
static class Node {
    private Node[] children = new Node[26];
    private int val = 0;
    public int pref = 0;

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    void setVal(int val) {
        this.val = val;
    }

    int getVal() {
        return val;
    }
}
```

```
Node root;

public MapSum() {
    root = new Node();
}


public int search(String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) return 0; //if does not exist return old val=0
        curr = curr.get(ch);
    }

    return curr.getVal();
}
```

```
public void insert(String key, int val) {
    int oldVal = search(key);
    Node curr = root;

    for(int i=0;i<key.length();i++) {
        curr.pref += (val-oldVal);
        char ch = key.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }

        curr = curr.get(ch);
    }

    curr.pref += (val-oldVal);
    curr.val = val;
}
```

```
public int sum(String prefix) {
    Node curr = root;

    for(int i=0;i<prefix.length();i++) {
        char ch = prefix.charAt(i);
        if(curr.contains(ch) == false) return 0;
        curr = curr.get(ch);
    }

    return curr.pref;
}
```

720. Longest Word in Dictionary

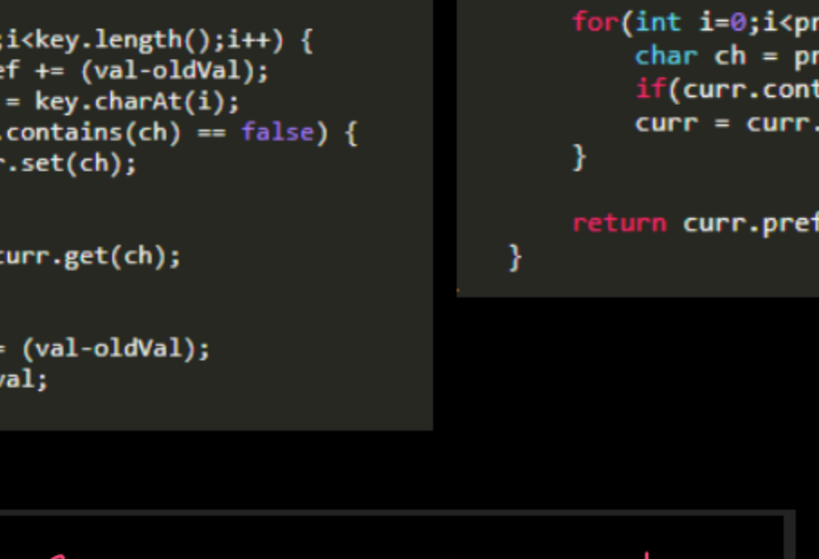
Medium 1316 1286 Add to List Share

Given an array of strings `words` representing an English Dictionary, return *the longest word in* `words` *that can be built one character at a time by other words in* `words`.

If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

{a, ban, app, appl, ap, apply, apples}

↓ Preprocessing (trie)



Apply dfs on shaded nodes (isEnd = true)

* Dfs handles lexicographical order itself.

```java
static class Node {
    private Node[] children = new Node[26];
    private boolean isEnd = false;

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    void setEnd() {
        isEnd = true;
    }

    boolean getEnd() {
        return isEnd;
    }
}
```

```java
public void insert(Node root, String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr = curr.get(ch);
    }

    curr.setEnd();
}
```

```java
String res = "";

public void DFS(Node curr,String asf) {

    if(curr.getEnd() == false) return;

    if(asf.length() > res.length()) {
        res = asf;
    }

    for(char ch='a';ch<='z';ch++) {
        if(curr.contains(ch) == true) {
            DFS(curr.get(ch),asf + ch);
        }
    }

}

public String longestWord(String[] words) {
    Node root = new Node();
    for(String word : words) insert(root,word);

    root.setEnd();
    DFS(root,"");
    return res;
}
```

648. Replace Words

Medium 1483 152 Add to List Share

In English, we have a concept called **root**, which can be followed by some other word to form another longer word - let's call this word **successor**. For example, when the **root** `"an"` is followed by the **successor** word `"other"`, we can form a new word `"another"`.

Given a `dictionary` consisting of many **roots** and a `sentence` consisting of words separated by spaces, replace all the **successors** in the sentence with the **root** forming it. If a **successor** can be replaced by more than one **root**, replace it with the **root** that has **the shortest length**.

Return *the* `sentence` after the replacement.

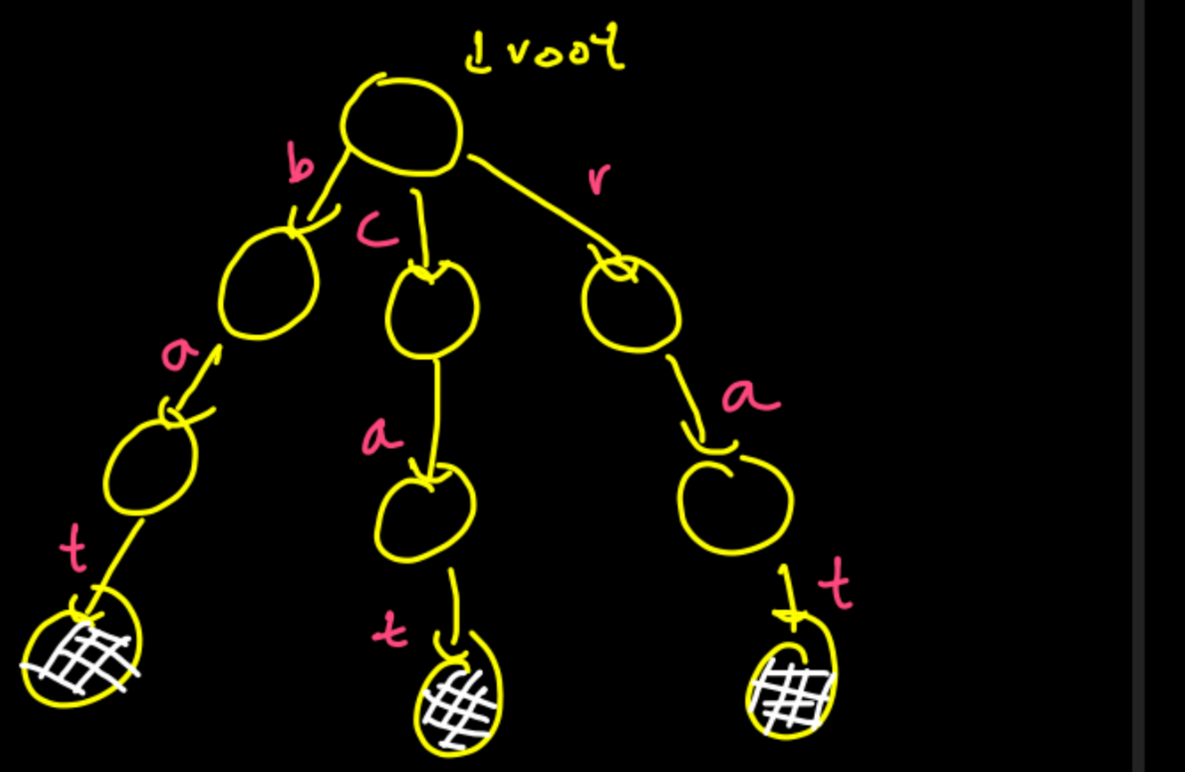
Example 1:

```
Input: dictionary = ["cat","bat","rat"], sentence
= "the cattle was rattled by the battery"
Output: "the cat was rat by the bat"
```

- Every two consecutive words in `sentence` will be separated by exactly one space.
- `sentence` does not have leading or trailing spaces.

} constraint

Replace words with Shortest Prefix



the cattle was rattled by the battery
↓ ↓ ↓ ↓ ↓ ↓ ↓
the cat was rat by the bat

```java
static class Node {
    private Node[] children = new Node[26];
    private boolean isEnd = false;

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    void setEnd() {
        isEnd = true;
    }

    boolean getEnd() {
        return isEnd;
    }
}
```

```java
public void insert(Node root, String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr = curr.get(ch);
    }

    curr.setEnd();
}
```

```java
public String search(Node curr, String word) {
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);

        //shortest length subtring for a word
        if(curr.getEnd() == true) return word.substring(0,i);

        //word with no prefix will itself be returned
        if(curr.contains(ch) == false) return word;

        curr = curr.get(ch);

    }

    //entire word is found but no prefix
    return word;
}
```

```java
public String replaceWords(List<String> dictionary, String sentence) {
    Node root = new Node();
    for(String word : dictionary) {
        insert(root,word);
    }

    StringBuilder res = new StringBuilder("");
    for(String word : sentence.split(" ")) {
        if(res.length() > 0) res.append(" ");
        res.append(search(root,word));
    }

    return res.toString();
}
```

14. Longest Common Prefix

Easy 7624 2927 Add to List Share

Write a function to find the longest common prefix string amongst an array of strings.

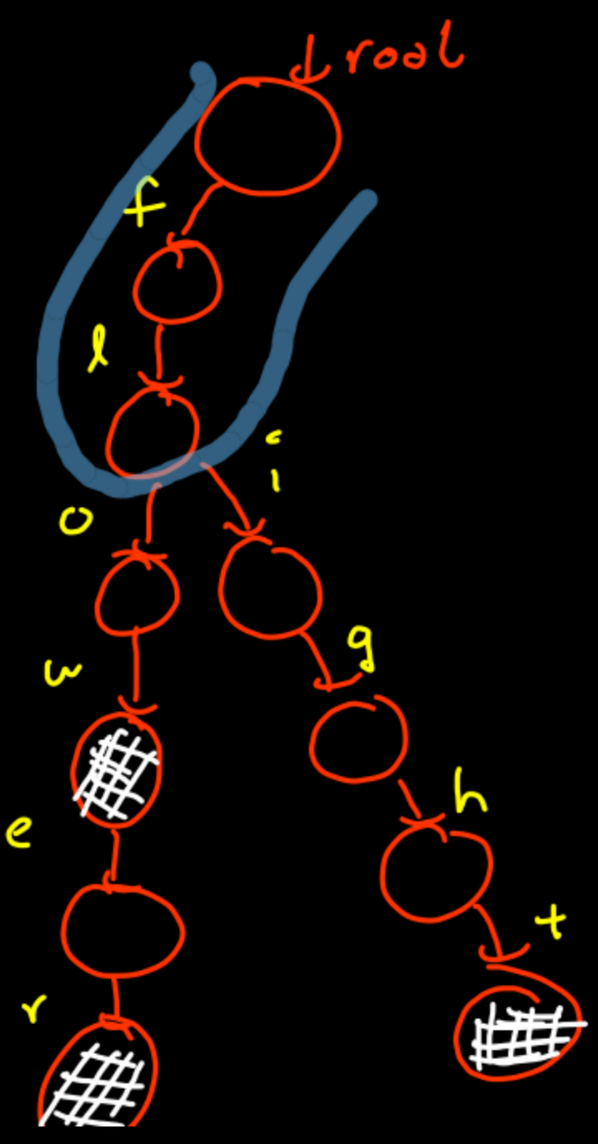
If there is no common prefix, return an empty string `""`.

Example 1:

```
Input: strs = ["flower","flow","flight"]
Output: "fl"
```

Example 2:

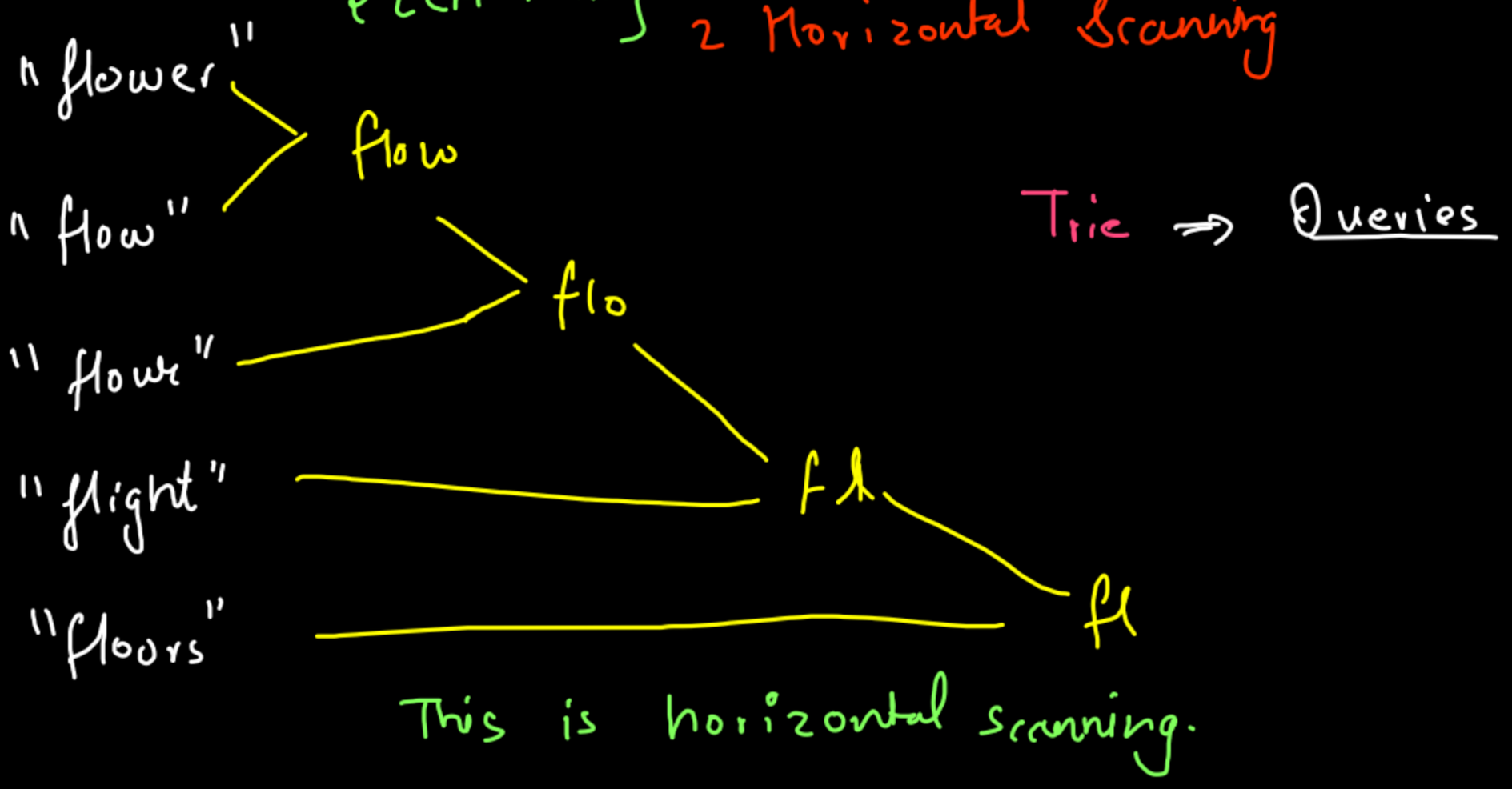
```
Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```



Longest Common Prefix {LCA → Tree}

1 Vertical Scanning

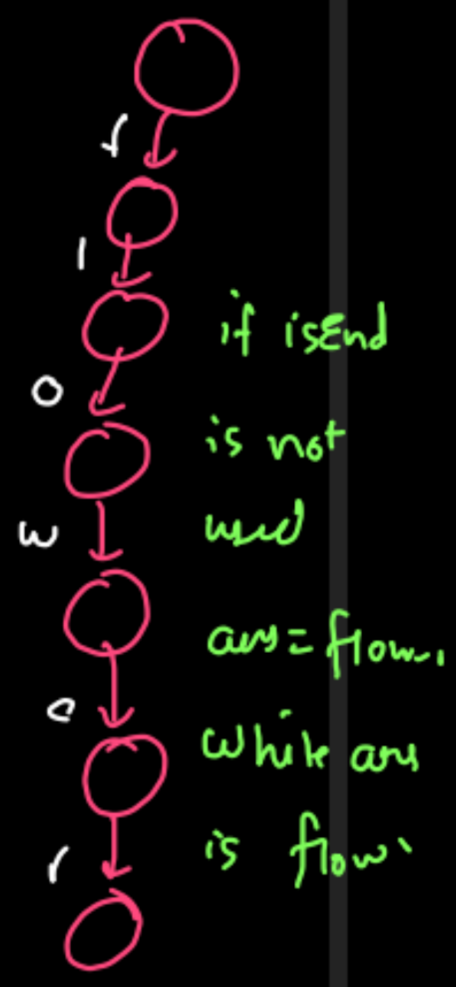
2 Horizontal Scanning




```java
static class Node {
    private Node[] children = new Node[26];
    private boolean isEnd = false;

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    void setEnd() {
        isEnd = true;
    }

    boolean getEnd() {
        return isEnd;
    }
}
```

```java
String res = "";
public void dfs(Node curr,String asf) {

    int count = 0;
    char c = '0';

    if(asf.length() > res.length()) res = asf;

    if(curr.getEnd() == true) return;

    for(char ch='a';ch<='z';ch++) {
        if(curr.contains(ch) == true) {
            c = ch;
            count++;
        }
    }

    if(count == 1)
        dfs(curr.get(c),asf + c);

}
```

important




```java
public void insert(Node root, String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr = curr.get(ch);
    }

    curr.setEnd();
}
```

```java
public String longestCommonPrefix(String[] strs) {
    Node root = new Node();
    for(String str: strs) {
        insert(root,str);
    }
    dfs(root,"");
    return res;
}
}
```

Shortest Unique Prefix (1B)

Find shortest unique prefix to represent each word in the list.

Example:

```
Input: [zebra, dog, duck, dove]
Output: {z, dog, du, dov}
where we can see that
zebra = z
dog = dog
duck = du
dove = dov
```

NOTE: Assume that no word is prefix of another. In other words, the representation is always possible.

zebra, dog, duck, dove

z dog du dov

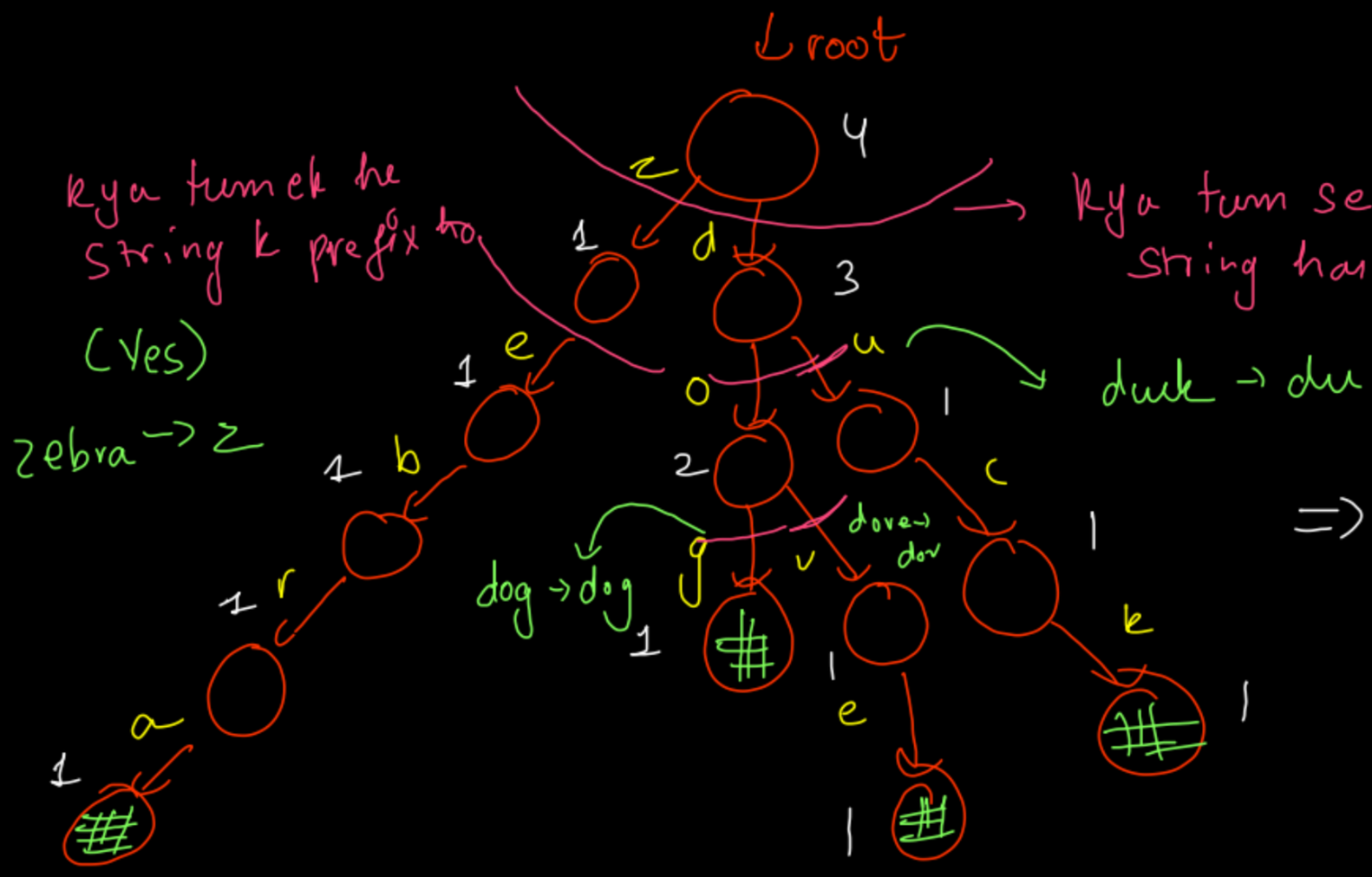
find the shortest prefix for each string ie not a prefix of any other string in the array.

⇒ Ex: dog and dogs will never be there in the array as dog is a prefix for dogs.

If this happens, dog will have no pr such that it will not be anu other

zebra, dog, duck, dove

Use Prefix count



Kya tumke he string k prefix ho (Yes)

zebra → z

Kya tum se nikalne vali ek he string hai (No)

⇒ jaha prefix count pehli baar 1 mila vahi ans hai

```
public static class Node {
    Node[] children = new Node[26];
    int prefCount = 0;
    int isEnd = 0;

    Node get(char ch) {
        return children[ch-'a'];
    }

    void set(char ch) {
        children[ch-'a'] = new Node();
    }

    boolean contains(char ch) {
        if(children[ch-'a'] != null) return true;
        return false;
    }
}
```

```
public void insert(Node root, String word) {
    Node curr = root;
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr.prefCount++;
        curr = curr.get(ch);
    }

    curr.prefCount++;
    root.isEnd++;
}
```

Shortest Unique Prefix

Medium 77 0 Add to favorites

Asked In: GOOGLE

```
public String search(Node root, String word) {
    Node curr = root;
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);

        if(curr.prefCount == 1) return word.substring(0,i);
        curr = curr.get(ch);
    }

    return word;
}
```

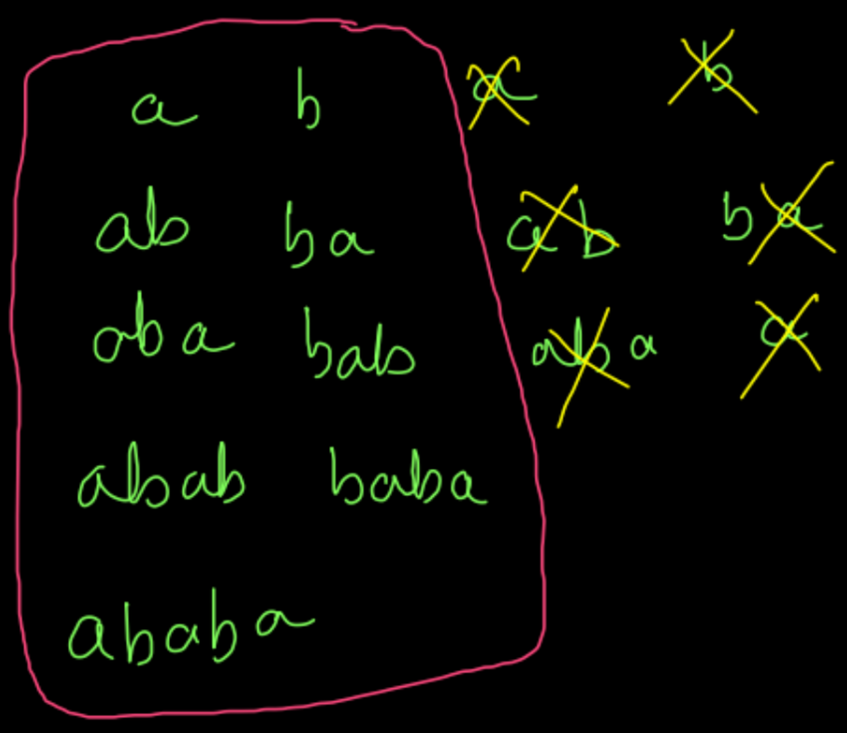
```
public String[] prefix(String[] A) {
    Node root = new Node();
    for(String str : A) {
        insert(root,str);
    }

    String[] res = new String[A.length];
    for(int i=0;i<res.length;i++) {
        res[i] = search(root,A[i]);
    }

    return res;
}
}
```

Count Distinct Substrings

{LC: 1698 - Locked} {GfG}

{Saare topics padhne ke baad is qus me dimag me nahi aaega ki tries use hona hai}

"ababa"



HashMap will give TLE

$n^2 * n \rightarrow O(n^3)$

↓ Substrings → String length

Using HashSet gives TLE

```
public static int countDistinctSubstring(String st)
{
    //your code here
    HashSet<String> hs = new HashSet<>();

    for(int i=0;i<st.length();i++) {
        StringBuilder sb = new StringBuilder("");
        for(int j=i;j<st.length();j++) {
            sb.append(st.charAt(j));
            hs.add(sb.toString());
        }
    }

    hs.add("");
    return hs.size();

}
```

N^3 L

$N^3 * L \approx O(N^4)$

↓ WC = N ↓ WorstCase

Best case $O(N^3)$

$(2N^3 * T)$ → test case

$O(N^2)$

```
public static class Node {
    Node[] children = new Node[26];

    public Node get(char ch) {
        return children[ch-'a'];
    }

    public void set(char ch) {
        children[ch-'a'] = new Node();
    }
}
```

```
public static int countDistinctSubstring(String st)
{
    //your code here
    Node root = new Node();
    int count = 1;

    for(int i=0;i<st.length();i++) {
        Node curr = root;
        for(int j=i;j<st.length();j++) {
            if(curr.get(st.charAt(j)) == null) {
                curr.set(st.charAt(j));
                count++;
            }
            curr = curr.get(st.charAt(j));
        }
    }

    return count;

}
```

Magic Dictionary

676. Implement Magic Dictionary

Medium 1006 179 Add to List Share

Design a data structure that is initialized with a list of **different** words. Provided a string, you should determine if you can change exactly one character in this string to match any word in the data structure.

Implement the `MagicDictionary` class:

- `MagicDictionary()` Initializes the object.
- `void buildDict(String[] dictionary)` Sets the data structure with an array of distinct strings `dictionary`.
- `bool search(String searchWord)` Returns `true` if you can change **exactly one character** in `searchWord` to match any string in the data structure, otherwise returns `false`.

Input
["MagicDictionary", "buildDict", "search", "search", "search", "search"]
[[], [["hello", "leetcode"]], ["hello"], ["hhllo"], ["hell"], ["leetcoded"]]
Output
[null, null, false, true, false, false]

Trie

"hello"
"leetcode"

hello → ✗ (Can't Replace any char)

hhllo → hello ✓

hell → ✗ Length is small

leetcoded → ✗ Length is greater

lutcodz → leetcode ✓

Recursion parameters → { word, idx, Node curr, boolean change }

hhllo → root, 0, false

word, curr, idx, change

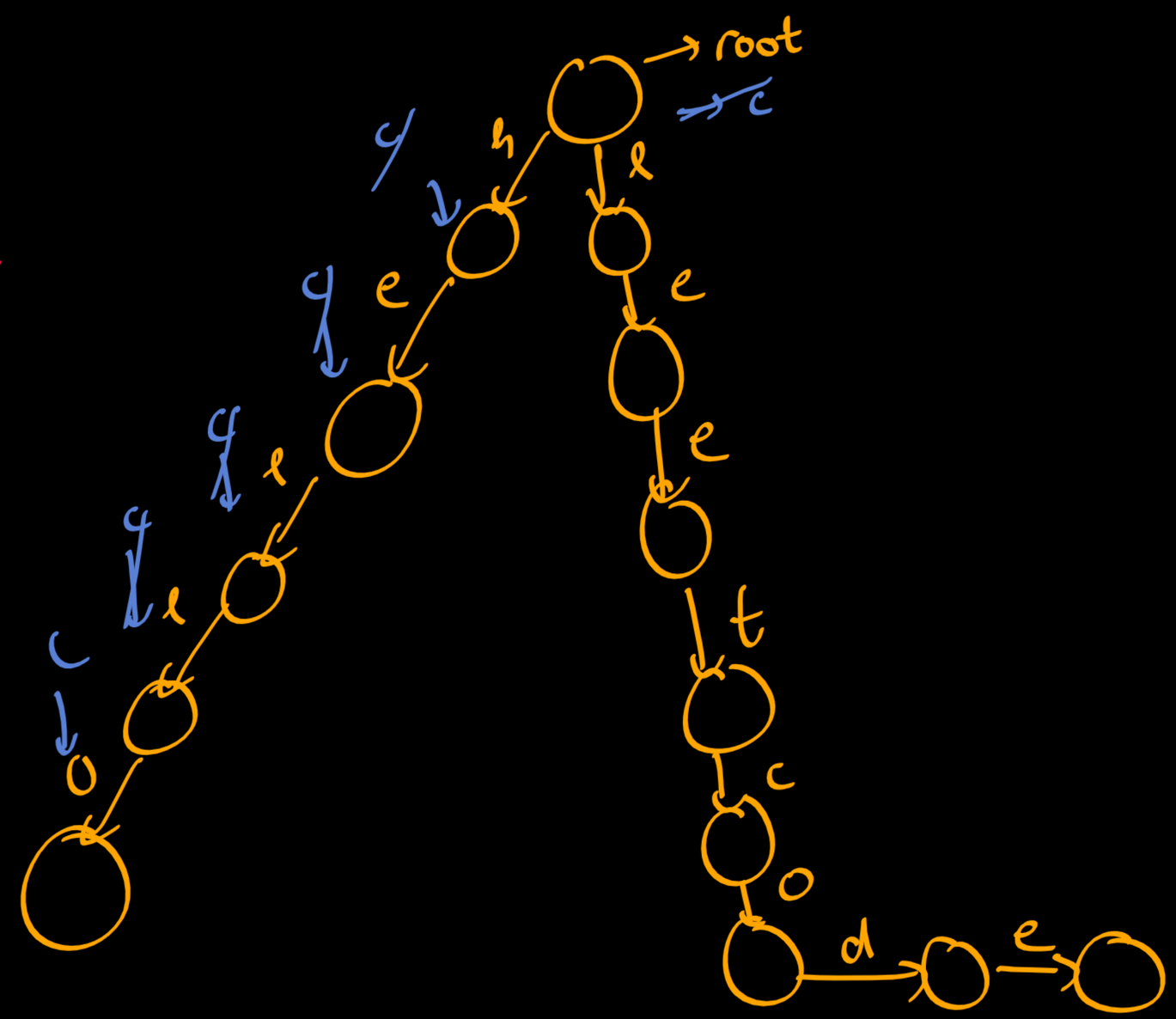
hhllo, (h), 1, false

hhllo, (e), 2, true

hhllo, (l), 3, true

hhllo, (l), 4, true

hhllo, (o), 5, true



↳ if(isEnd == true && change == true)

```
class MagicDictionary {

    static class Node {
      private Node[] children = new Node[26];
      private boolean isEnd = false;

      boolean contains(char ch) {
          return (children[ch-'a'] != null) ;
      }

      void set(char ch) {
          children[ch-'a'] = new Node();
      }

      Node get(char ch) {
          return children[ch-'a'];
      }

      void setEnd() {
          isEnd = true;
      }

      boolean getEnd() {
          return isEnd;
      }
  }

  Node root;
```

} Same Node class.

```java
Node root;

public MagicDictionary() {
    root = new Node();
}

private void insert(String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr = curr.get(ch);
    }

    curr.setEnd();
}

public void buildDict(String[] dictionary) {

    for(String word : dictionary) {
        insert(word);
    }
}
```

```java
public boolean search(String word,int idx,Node curr, boolean change) {

    if(idx == word.length()) {
        if(curr.getEnd() == true && change == true) return true;
        return false;
    }

    char ch = word.charAt(idx);
    //apna character same hai
    //aage se ans bhi true aa raha hai
    //aage se ans tabhi true aaega jab ek replacement ki hogi
    //and apna character match yaani humne replacement nahi kari
    //this means humne nahi ki but aage kahi ek replacement ho k ans true aa raha hai
    //to ye sahi hai
    //hence return true
    if(curr.contains(ch) == true && search(word,idx + 1,curr.get(ch),change) == true) {
        return true;
    }

    if(change == true) return false;

    for(char chn ='a';chn<='z';chn++) {
        if(chn == ch) continue;
        //same character ko usi se replace kara doge to galat ans aaega

        if(curr.contains(chn) && search(word,idx + 1,curr.get(chn),true)) {
            return true;
        }
    }

    return false;
}

public boolean search(String searchWord) {
    return search(searchWord,0,root,false);
}
}
```

→ current char bina replace kare ans milta hai kya

→ current char replace karne par ans milta hai kya

Search Suggestion System → System Design type question

1268. Search Suggestions System

Medium 2236 133 Add to List Share

You are given an array of strings `products` and a string `searchWord`.

Design a system that suggests at most three product names from `products` after each character of `searchWord` is typed. Suggested products should have common prefix with `searchWord`. If there are more than three products with a common prefix return the three lexicographically minimums products.

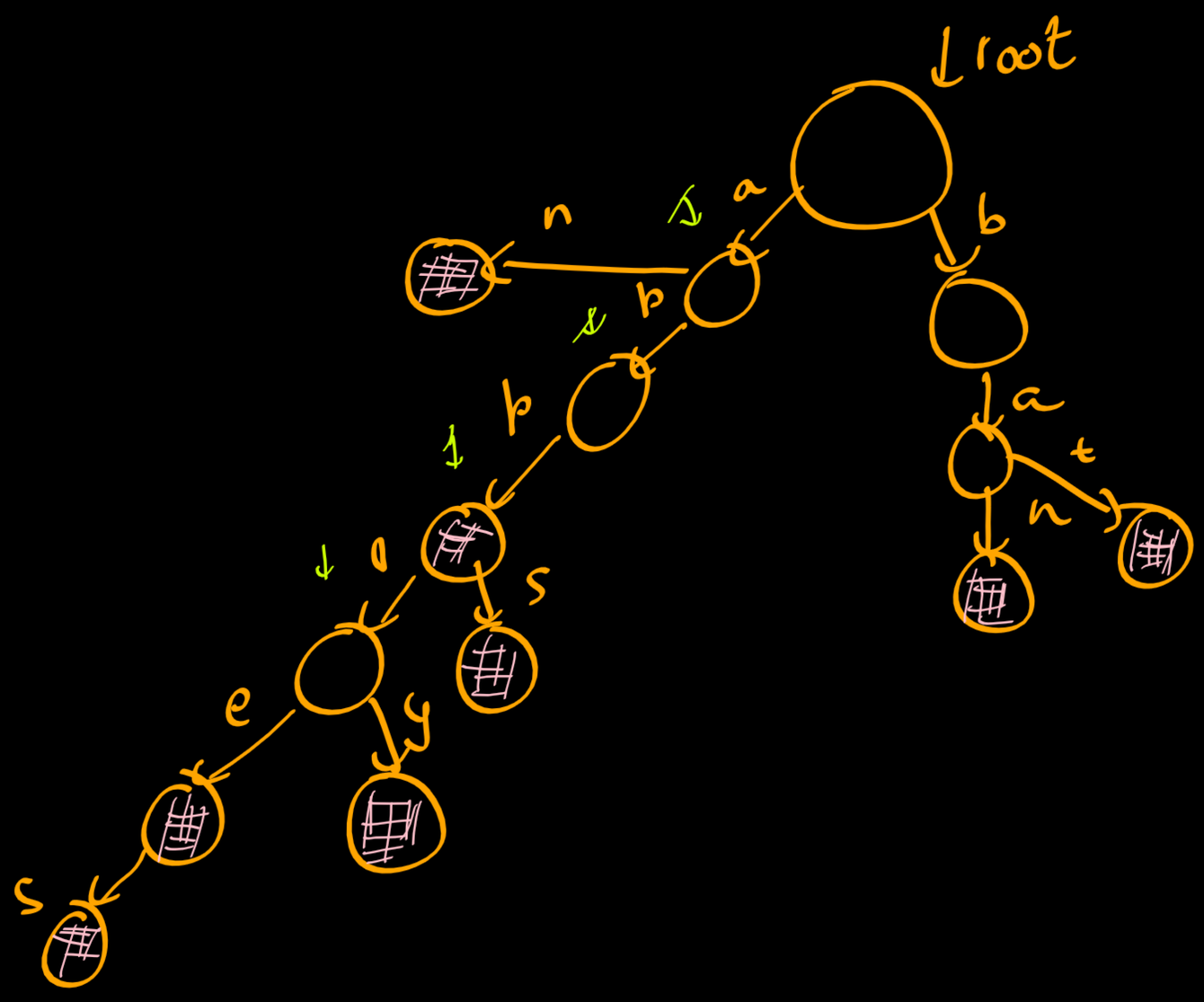
Return *a list of lists of the suggested products after each character of* `searchWord` *is typed.*

Let us say our DB has following words.

app, apple, apples, apps, apply, ban, bat, an.

Say, we are searching for the word "application"

app, apple, apples, apps, apply, ban, bat, an.




```
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 ch='a';ch<='z';ch++) {
        if(root.contains(ch) == true) {
            DFS(root.get(ch),ssf + ch,ans,k);
        }
    }
}
```

```
public List<List<String>> suggestedProducts(String[] products, String searchWord) {
    Node root = new Node();
    for(String word : products) {
        insert(root,word);
    }

    List<List<String>> res = new ArrayList<>();
    for(int i=0;i<searchWord.length();i++) {
        char ch = searchWord.charAt(i);
        if(root.contains(ch) == true) {
            root = root.get(ch);
            List<String> ans = new ArrayList<>();
            DFS(root,searchWord.substring(0,i+ 1),ans,3);
            res.add(ans);
        } else break;
    }

    while(res.size() < searchWord.length()) {
        res.add(new ArrayList<>());
    }

    return res;
}
```

application

Dfs → preorder 3 words

{an, app, apple}

application

{app, apple, apples}

application

{app, apple, apples}

application

{apple, apples, apply}

application

{ }

→ Given in ques to give atmost 3 lexicographically minimum products.

Stream of Characters | Leetcode - 1032 Hard

1032. Stream of Characters

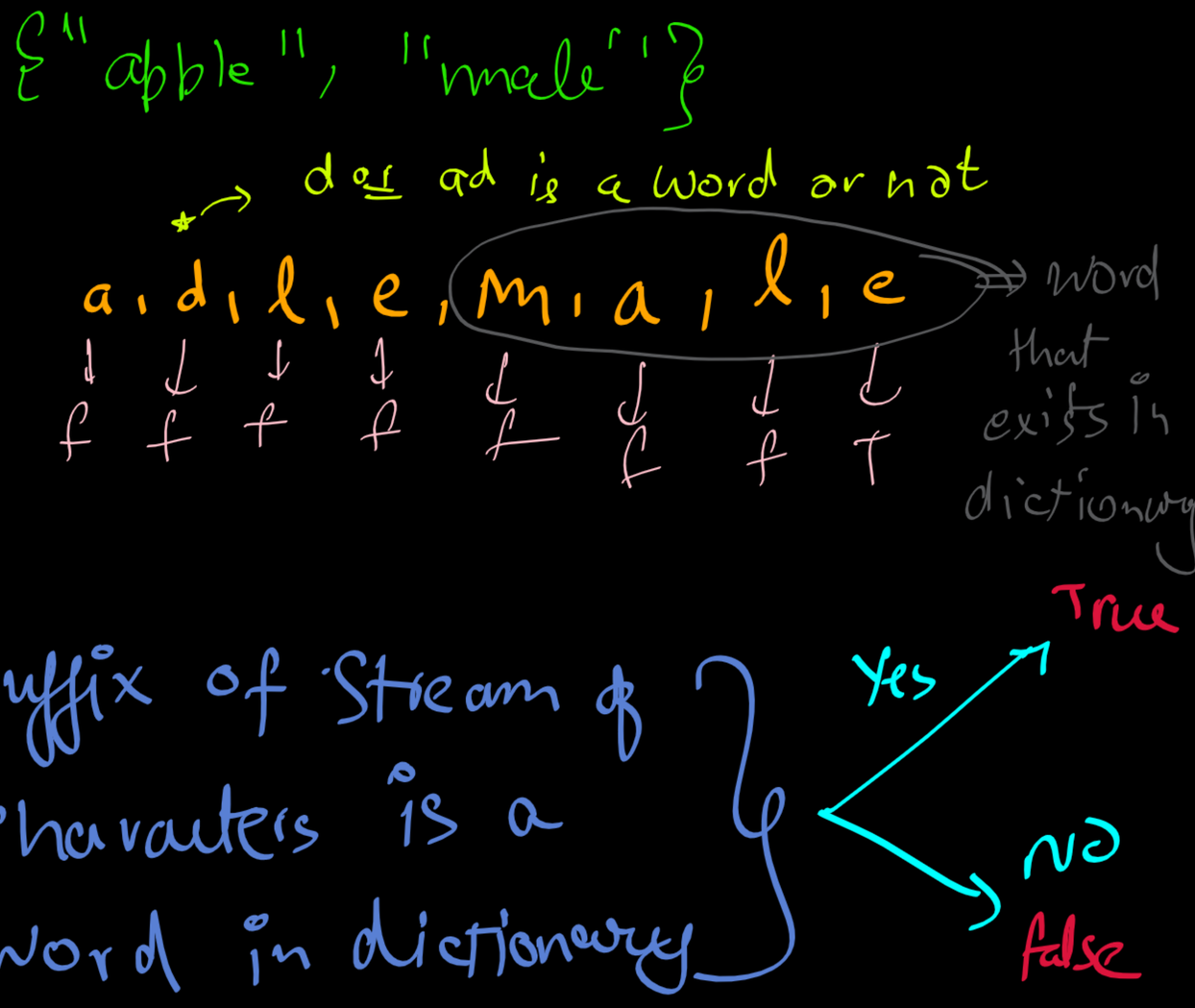
Hard 1510 169 Add to List Share

Design an algorithm that accepts a stream of characters and checks if a suffix of these characters is a string of a given array of strings `words`.

For example, if `words = ["abc", "xyz"]` and the stream added the four characters (one by one) `'a'`, `'x'`, `'y'`, and `'z'`, your algorithm should detect that the suffix `"xyz"` of the characters `"axyz"` matches `"xyz"` from `words`.

Implement the `StreamChecker` class:

- `StreamChecker(String[] words)` Initializes the object with the strings array `words`.
- `boolean query(char letter)` Accepts a new character from the stream and returns `true` if any non-empty suffix from the stream forms a word that is in `words`.



insert → R to L
search → R to L
{ Both searching and insertion will be done in reverse direc }

```
Node root;
int maxLen = 0;
public StreamChecker(String[] words) {
    root = new Node();
    for(String word : words) {
        insert(root,word);
        maxLen = Math.max(word.length(),maxLen);
    }
}
```

```
StringBuilder sb = new StringBuilder("");
public boolean query(char letter) {
    sb.append(letter);

    Node curr = root;
    //here sb.length() - maxLen -1 is the length of the prefix
    //this means when prefix length is greater than the length of any word
    // then we have to stop
    for(int i=sb.length()-1;i>=Math.max(0,sb.length()-maxLen-1);i--) {
        if(curr.getEnd() == true) return true;
        char ch = sb.charAt(i);
        if(curr.contains(ch) == false) return false;
        curr = curr.get(ch);
    }
    if(curr.getEnd() == true) return true;
    return false;
  }
}
```

Palindrome Pairs & LC- 336 Hard &

336. Palindrome Pairs

Hard 2639 248 Add to List Share

Given a list of **unique** words, return all the pairs of the ***distinct*** indices `(i, j)` in the given list, so that the concatenation of the two words `words[i] + words[j]` is a palindrome.

Example 1:

```
Input: words = ["abcd","dcba","lls","s","sssll"]
Output: [[0,1],[1,0],[3,2],[2,4]]
Explanation: The palindromes are
["dcbaabcd","abcddcba","slls","llssssll"]
```

["abcd", "dcba", "lls", "s", "sssll"]
0 1 2 3 4

0 1 } Both are palindromes — abcddcba, dcbaabcd
1 0 }

3,2 sll s ✓

2,3 α llss ✗

2,4 → llsssll

1,2 α sssll s

x+y ⇒ palindrome
- x is reverse of y → (x,y) and (y,x) } Both will be palindromes
- suffix of y = reverse of x
- prefix of x = reverse of y

suffix of y = reverse of x

2,4
lls — sssll ⇒ suffix of y
reverse → sll

suffix of y is reverse of x }

for this to be palindrome, the remaining string (y) should also be a palindrome.

Say x = llsss y = sll
prefix of x is reverse of y

Again, remaining string (x) should be a palindrome.

Input: words = ["abcd","dcba","lls","s","sssll"]
Output: [[0,1],[1,0],[3,2],[2,4]]
Explanation: The palindromes are ["dcbaabcd","abcddcba","slls","llssssll"]

Make "abcd" as y

Start searching for suffixes of abcd

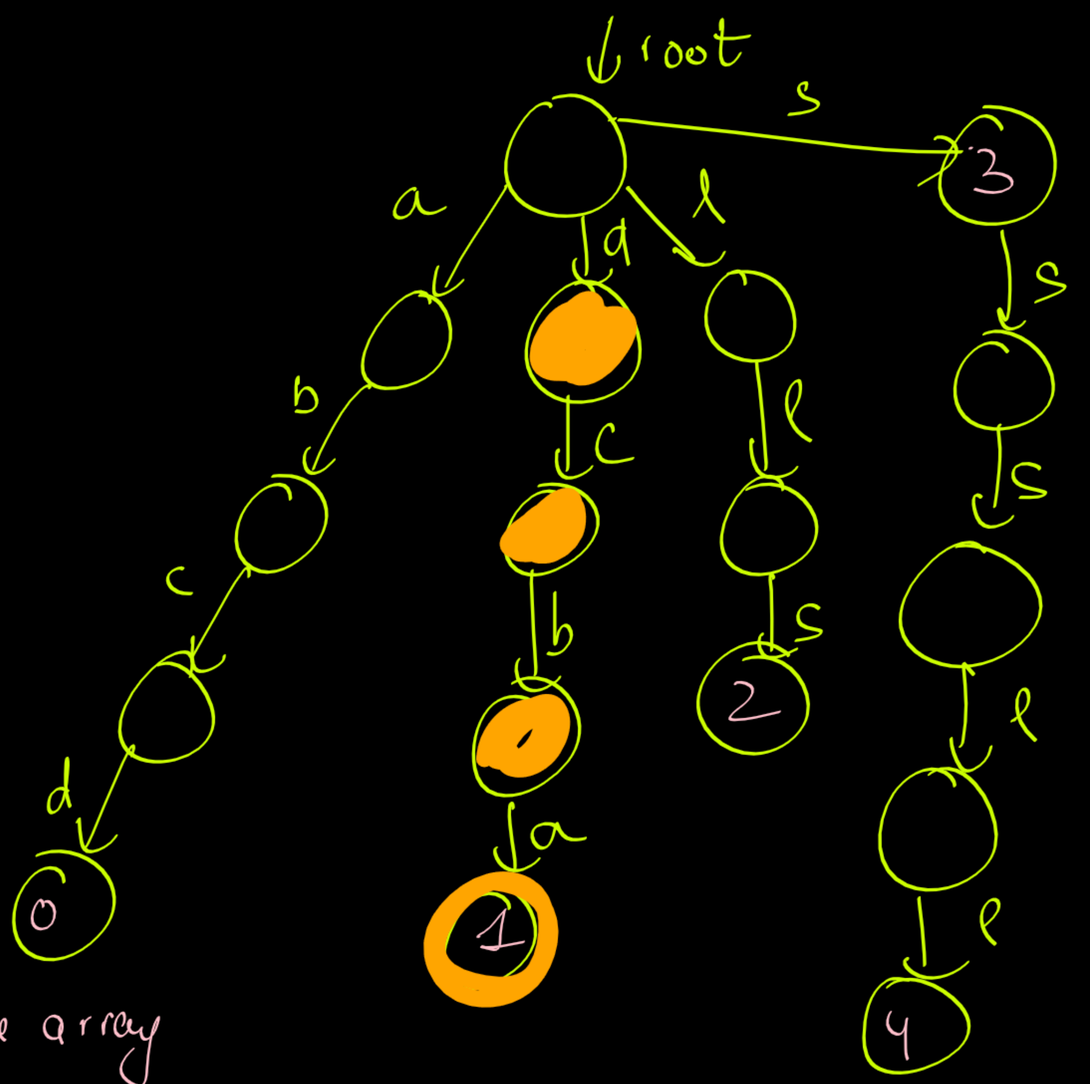
x + y

abcd

d + abc | d

↳ If we had 'd' in the trie. We do not have d currently. Its just a prefix not a word.

end contains index of string in the array →



dcba + abcd

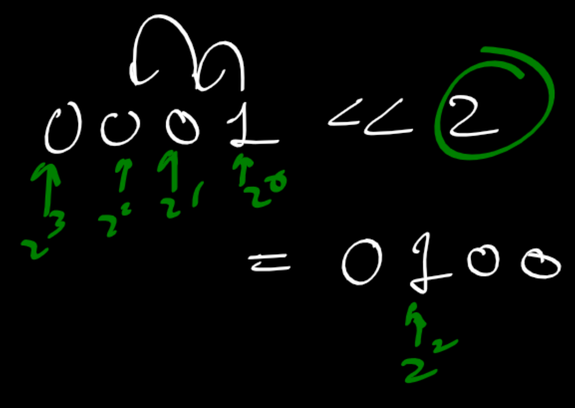
x y

Since x is reverse of y both (x,y) & (y,x) will be Palindromes. But, we will currently add only (x,y)(1,0)

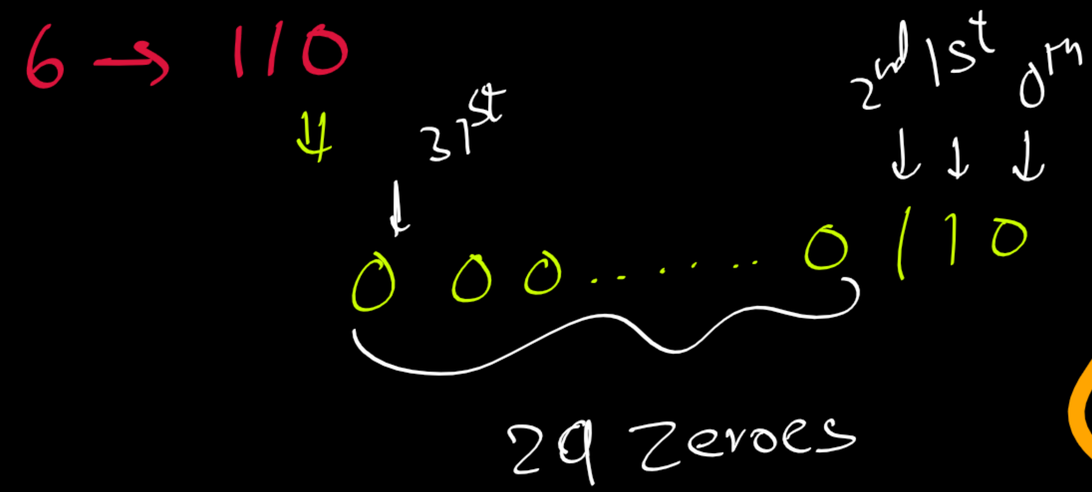
⇒ In the next step, take string from the array i.e "dcba" & make it y.

Bit Manipulation Basics

Integer [4 bytes ⇒ 32 bits]



Left shift by 2 multiplied the no by 2^2.



So, Num << N ⇒ Num * 2^N

Left shift = 01101001 << 2 = 10100101 00

Right Shift

$0101 >> 2 = 0001$ (5)

$0101 >> 1 = 0010$

$x >> N \Rightarrow x = x / 2^N$

Get, Set and Unset

get k^{th} bit

Set k^{th} bit is 0, make it 1, else leave it

If k^{th} bit is 1, make it 0, else leave as it is.

toggle : $1 \to 0$ & $0 \to 1$

Say the no is 0101

and k = 2nd Bit

Say, mask = 0001

Since k = 2nd Bit, Mask << 2 = 0100

⇒ get Number & Mask

```
  0 1 0 1
& 0 1 0 0
---------
  0 1 0 0
---------
  ↓
  kth bit = 1
```

No & 0 = 0

No & 1 = No } at k^{th} bit, we get value of k^{th} bit 0/1 rest all bits will be 0}

if ((Num & (1<<k)) != 0) Bit was set i.e. 1 else 0.

Set

$x = 5$

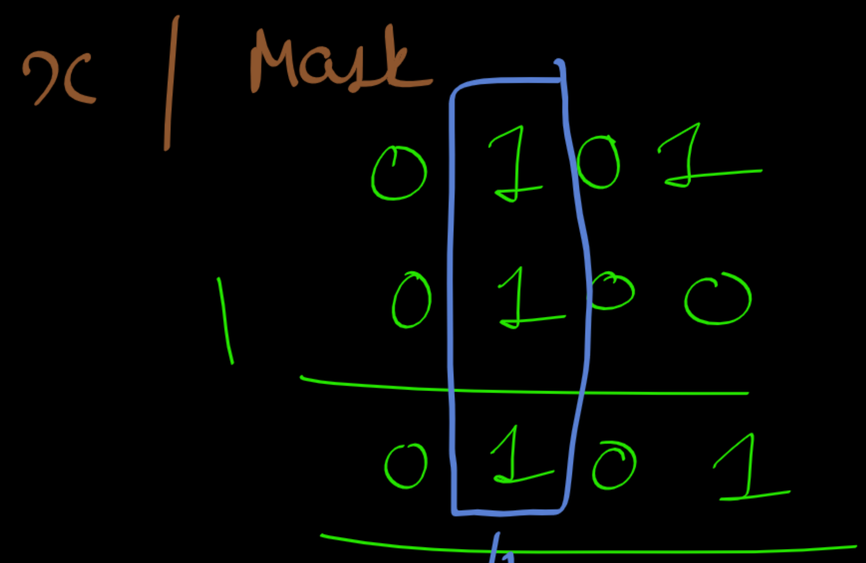
0 1 0 1

2nd Bit

Mask = 0 0 0 1

Mask << 2 = 0 1 0 0

x | Mask



This bit is set (It was already set)

x | 0 = x

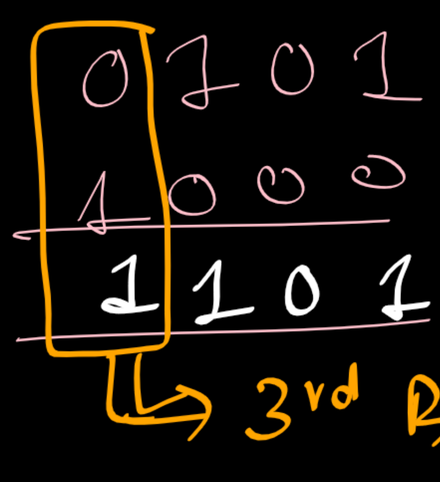
x | 1 = 1

for 3rd Bit

Mask = 0 0 0 1

Mask << 3 = 1 0 0 0

x V Mask



3rd Bit is now set.

To set a bit

→ create an appropriate mask

→ x | Mask

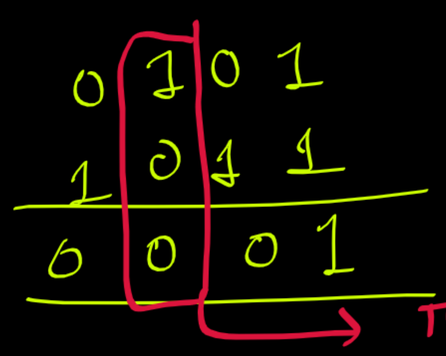
Unset

x = 5 = 0 1 0 1 & k = 2nd Bit

Mask = 0 0 0 1

Mask << 2 = 0 1 0 0

~(Mask) = 1 0 1 1



This bit is now unset.

→ Generate Mask as discussed in prev qs.

→ Negate the mask

→ x & Mask

Toggle

$x = 0101$ and $k = 2^{nd}$ Bit

mask = 0001

Mask << 2 = 0100

$1 \wedge 0 = 1$

$0 \wedge 1 = 1$

$1 \wedge 1 = 0$

$0 \wedge 0 = 0$

XOR → $\wedge$

```
  0 1 0 1
^ 0 1 0 0
---------
  0 0 0 1
```

↳ Bit is toggled

for k = 3

```
  0 1 0 1   → x
^ 1 0 0 0   → mask
-----------
  1 1 0 1
```

↳ Bit is toggled

XOR Properties

① $x \wedge 0 = x$

② $x \wedge x = 0$

if $a \wedge b = c$
then $a \wedge c = b$
$b \wedge c = a$

Maximum XOR

421. Maximum XOR of Two Numbers in an Array

Medium 3855 325 Add to List Share

Given an integer array `nums`, return *the maximum result of* `nums[i] XOR nums[j]`, where `0 <= i <= j < n`.

Example 1:

```
Input: nums = [3,10,5,25,2,8]
Output: 28
Explanation: The maximum result is 5 XOR 25 = 28.
```

Brute force = $O(N^2)$

5, 9, 1, 3, 7

0101 1001 0001 0011 0111

5 ^ 5 = 0

5 ^ 9 = 1100 = 12

5 ^ 1 = 0100 = 4

5 ^ 3 = 0110 = 6

5 ^ 7 = 0010 = 2

9 ^ 9 = 0

9 ^ 1 = 1000 = 8

9 ^ 3 = 1010 = 10

9 ^ 7 = 1110 = 14 → max

1 ^ 1 = 0

1 ^ 3 = 0010 = 2

1 ^ 7 = 0110 = 6

3 ^ 3 = 0

3 ^ 7 = 0100 = 4

7 ^ 7 = 0

5, 9, 1, 3, 7

0101 1001 0001 0011 0111

We don't need end Marker as all nos are of equal length (4 bit in ex, 32 bit in code)

0 → Left

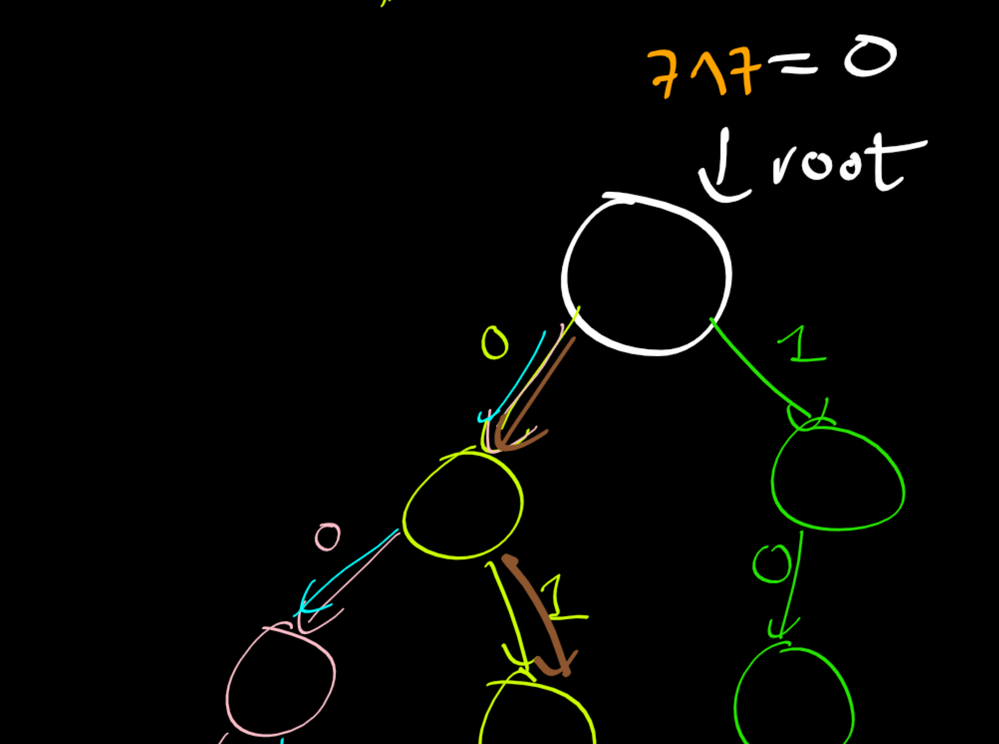
1 → Right

Node {

Node left;

Node right;

}



5, 9, 1, 3, 7

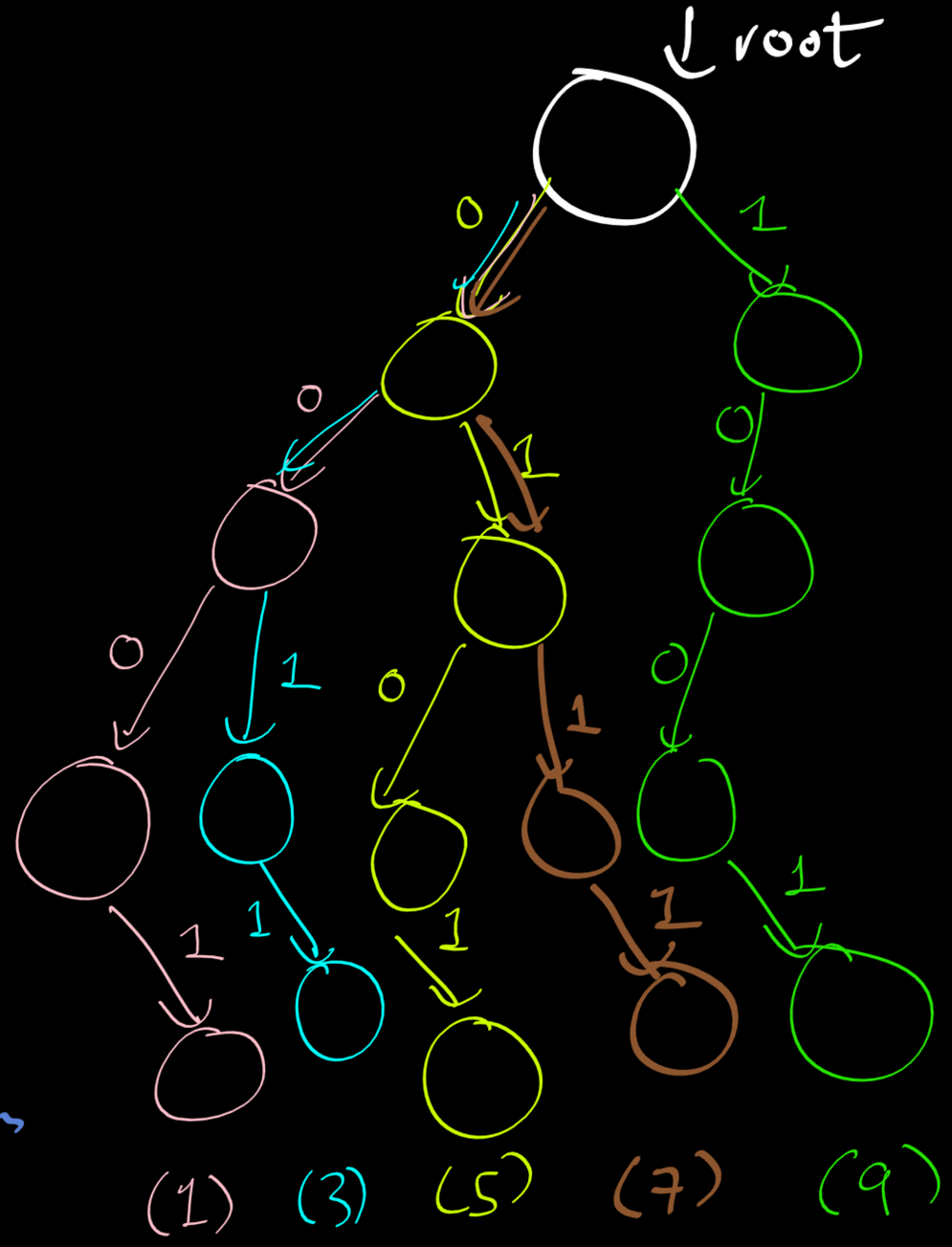
0101 1001 0001 0011 0111

0101

1010 ⇒ In ideal case, XOR with comp gives max no

1111

If this no is not present, we try to find the closest No. Closest No will have MSBs close to the no we want. Hence, we will look for a closer No from L → R.



0101

0010 1101

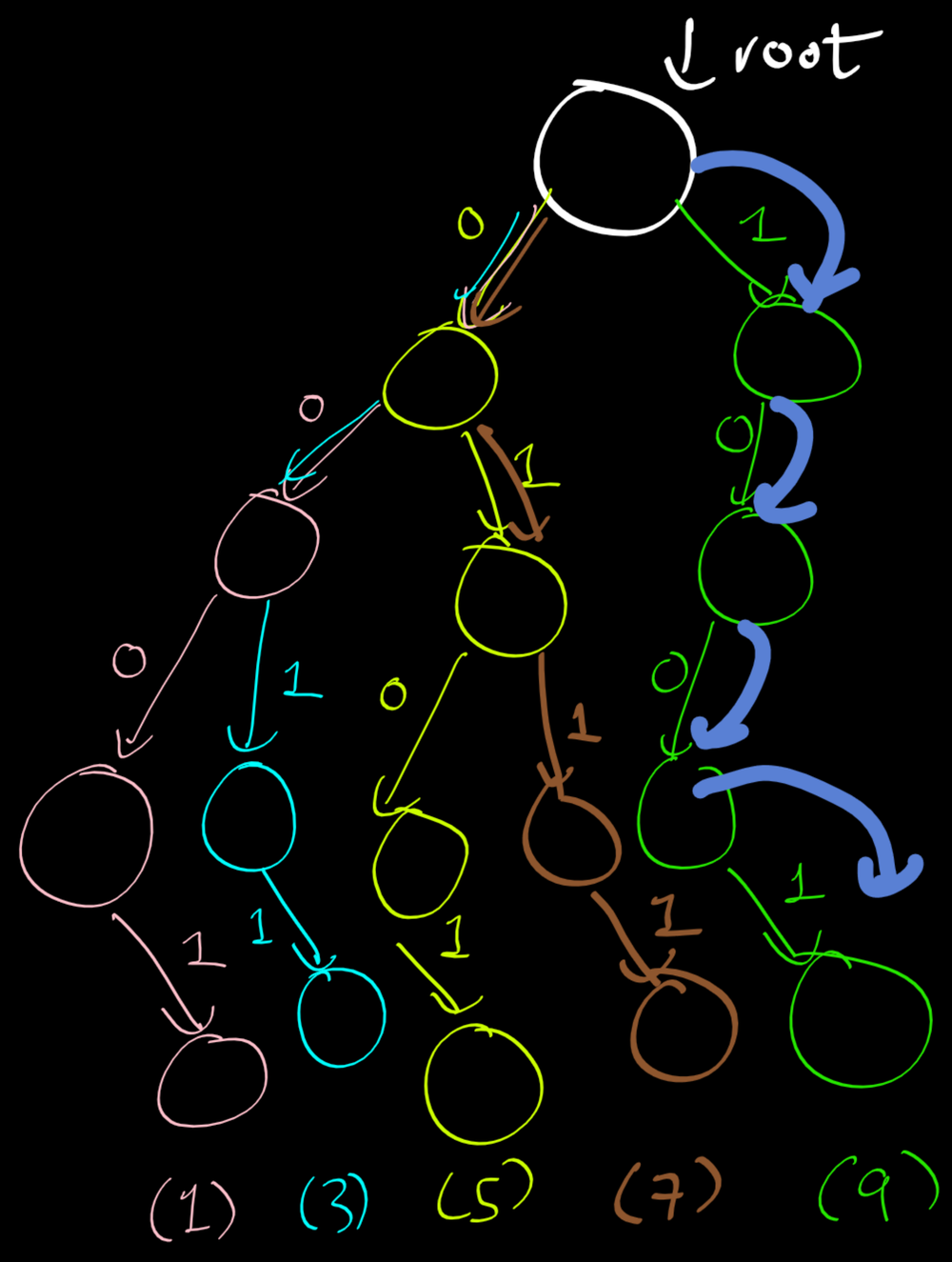
0111 < 1000

We will see the prefix. If XOR with prefix gives the bit = 1, we will move to that bit. Hence trie comes into picture.

① 0101

```
  1 1 1 1
    0 1 0 1
  1 0 0 1
No ---------
  1 1 0 0
XOR ---------
```

5 ^ 9 = 12



② 1001

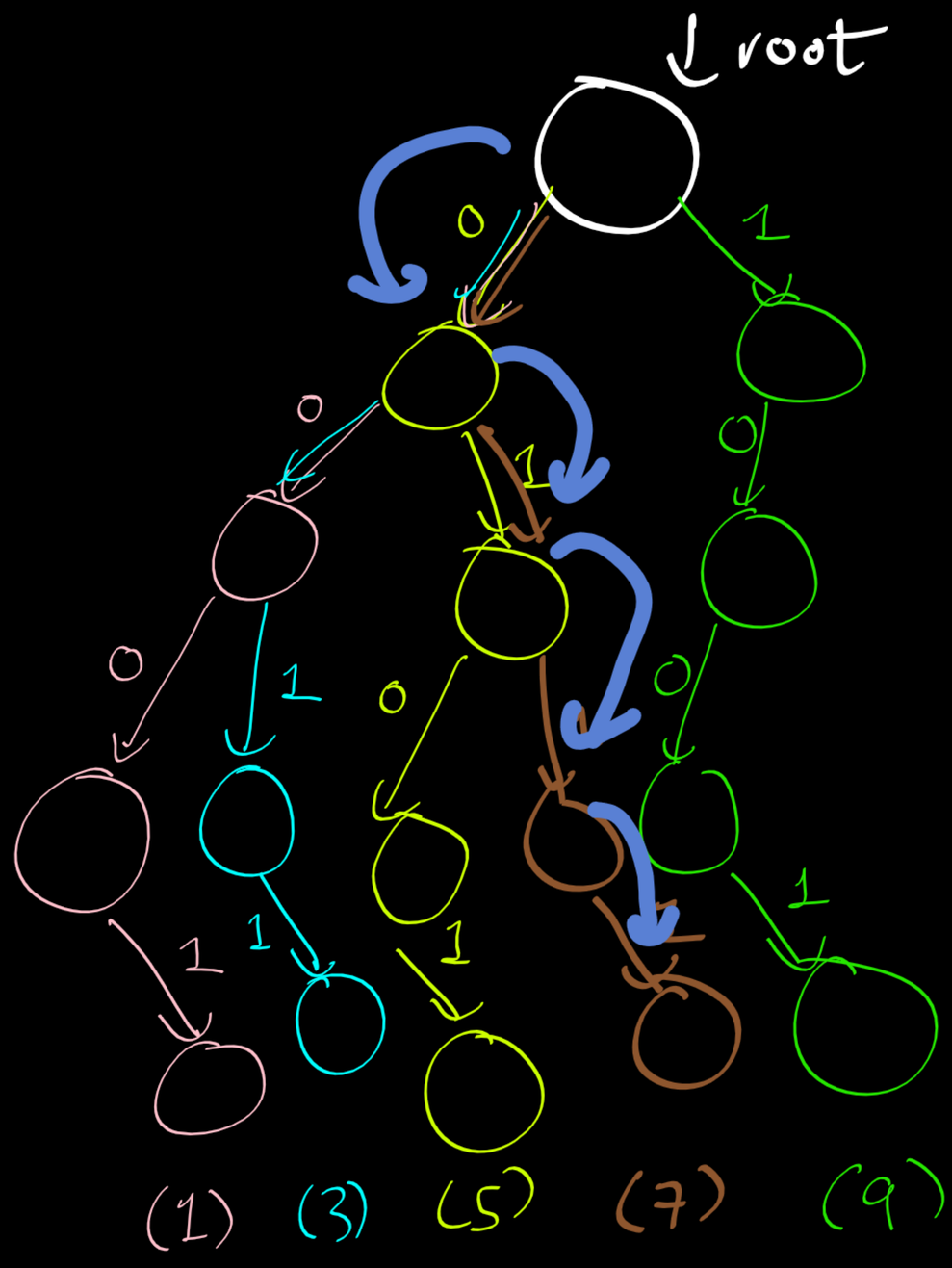
1 1 0 1

1 0 0 1

No 0 1 1 1

XOR 1 1 1 0

9 ^ 7 = (14)



So, we can do the same procedure for remaining nos also.

O(32 * N)

↓ insertion ↓ Search ↓ extra space

Slight optimization: We calculate 5^7, no need to calculate 7^5. So, do not insert all nos first. Insert one no, find max XOR; insert second no, now find max XOR and so on.

↳ Optimized because no of elements (Nos) in trie will be less.

```
class Node {
    Node left;
    Node right;
}

public void insert(Node root, int val) {
    for(int i=31;i>=0;i--) {
        int bit = val & (1 << i);

        if(bit == 0) {
            if(root.left == null)
                root.left = new Node();
            root = root.left;
        } else {
            if(root.right == null)
                root.right = new Node();
            root = root.right;
        }
    }
}
```

```
public int search(Node root,int val) {
    int maxXor = 0;

    for(int i=31;i>=0;i--) {
        int bit = val & (1 << i);
        if(bit == 0) {
            //pair exists with ith bit 1 so that we get XOR=1
            if(root.right != null) {
                root = root.right;
                maxXor = maxXor | (1 << i); //setting the ith bit
            } else {
                root = root.left;
            }
        } else {
            if(root.left != null) {
                root = root.left;
                maxXor = maxXor | (1 << i);
            } else {
                root = root.right;
            }
        }
    }

    return maxXor;
}
```

```
public int findMaximumXOR(int[] nums) {
    Node root = new Node();

    int ans = 0;
    for(int val : nums) {
        insert(root,val);
        ans = Math.max(ans,search(root,val));
    }

    return ans;
}
```

Slight optimization i.e. insert one elt & search.

Unique Rows in Boolean Matrix - GfG.

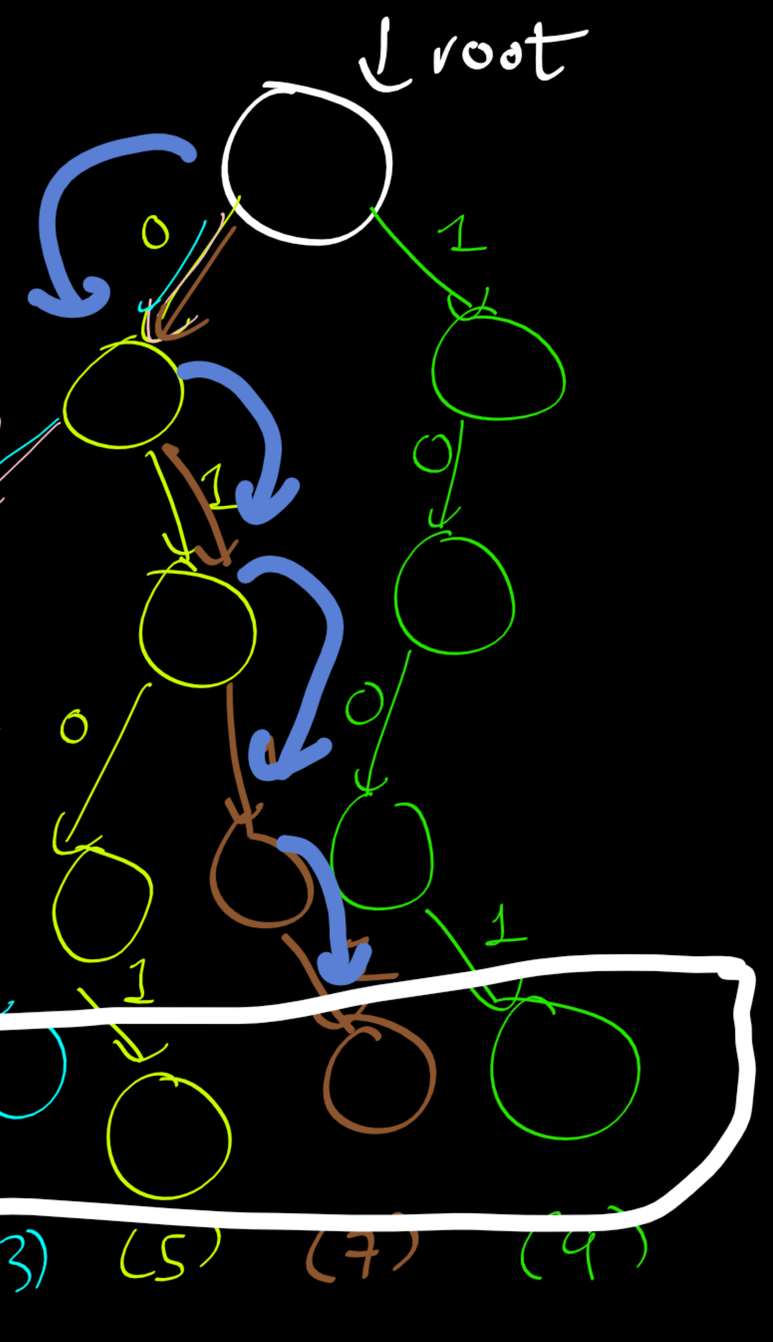
Matrix

→ 1 1 0 1
→ 1 0 0 1
→ 1 1 0 1
→ 1 0 0 1
→ 0 1 1 1
→ 1 0 0 0
→ 0 1 1 1

1 1 0 1
1 0 0 1
0 1 1 1
1 0 0 0

No of unique rows = 4.

Count No of Nodes after inserting in the deepest level.



Max XOR Pair - II {Max XOR with an ele from Array LC - 1707 Hard}

if we apply online queries
(process in same order as given)

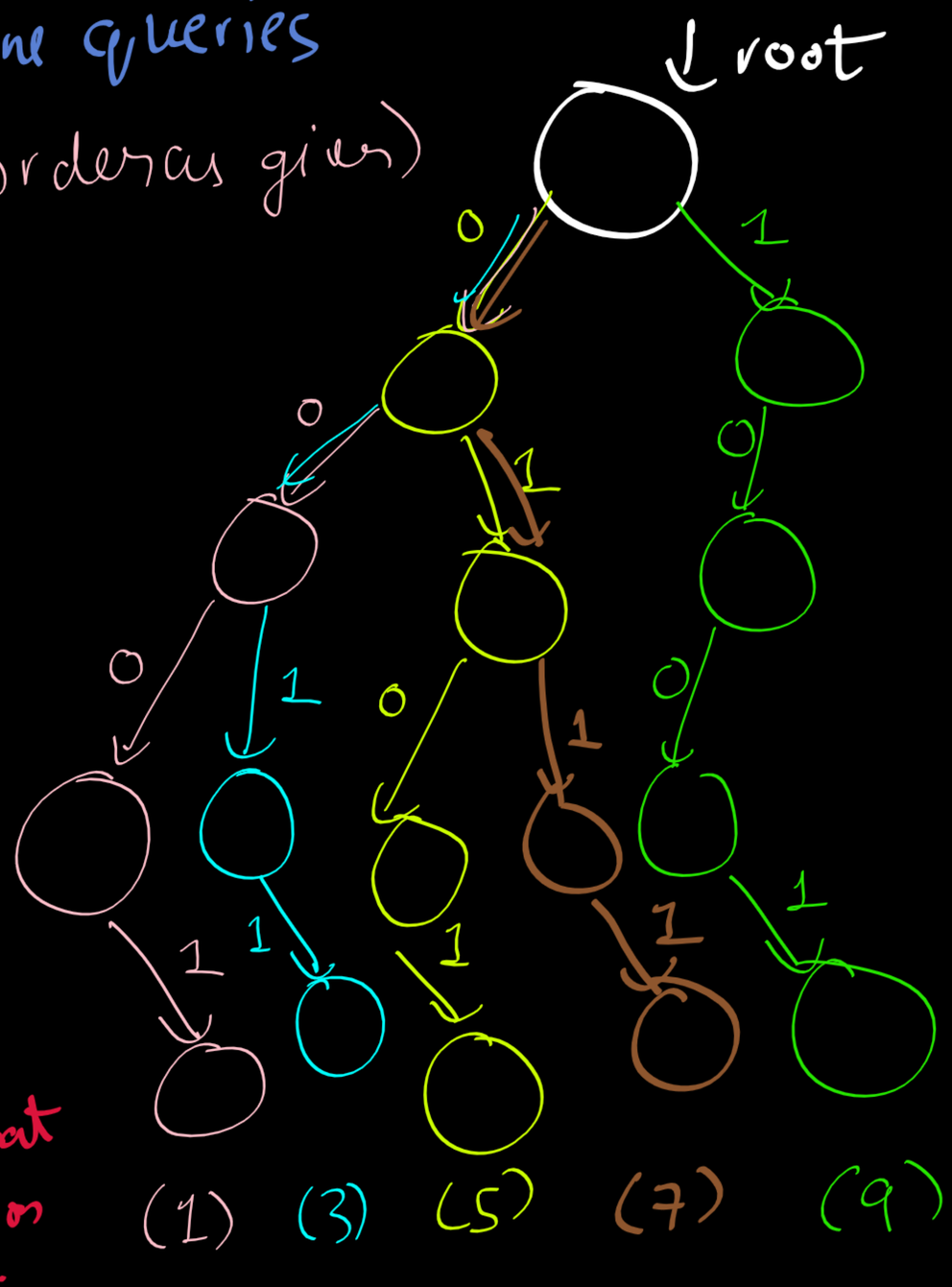
search (root, 6)
- ⑨

search (root, 4)
- ⑦, - ⑨

search (root, 8)

search (root, 7)
- ⑤, - ⑦, - ⑨

change the order such that we don't have to del from trie.



{6, 7} ③
{4, 5} ②
{8, 9} ④
{7, 3} ①

Keep inserting only those eles that lie in range

Max Xor Pair for 7 & subs that val taken from arr is ≤ 3.

offline queries

Why do we need offline queries ie. why do we need to change the order?

Ans: If we perform online queries, we might have deleted the numbers in some previous query that we need for this query. If we don't delete, it is impossible to know which numbers we don't have to include & which nos we have to. Here, we need offline queries so that we can change the order.

Offline queries: The word offline itself suggests that it is not real-time. The data will be available to us prior. So, we can change the order of queries only if the queries are offline i.e they are pre-known to us.

① ↓ {7,3} Actual order → 3rd

0 1 1 1
No 0 0 0 1
xor 0 1 1 0
7 ^ 1

② ↓ {4,5} 1st

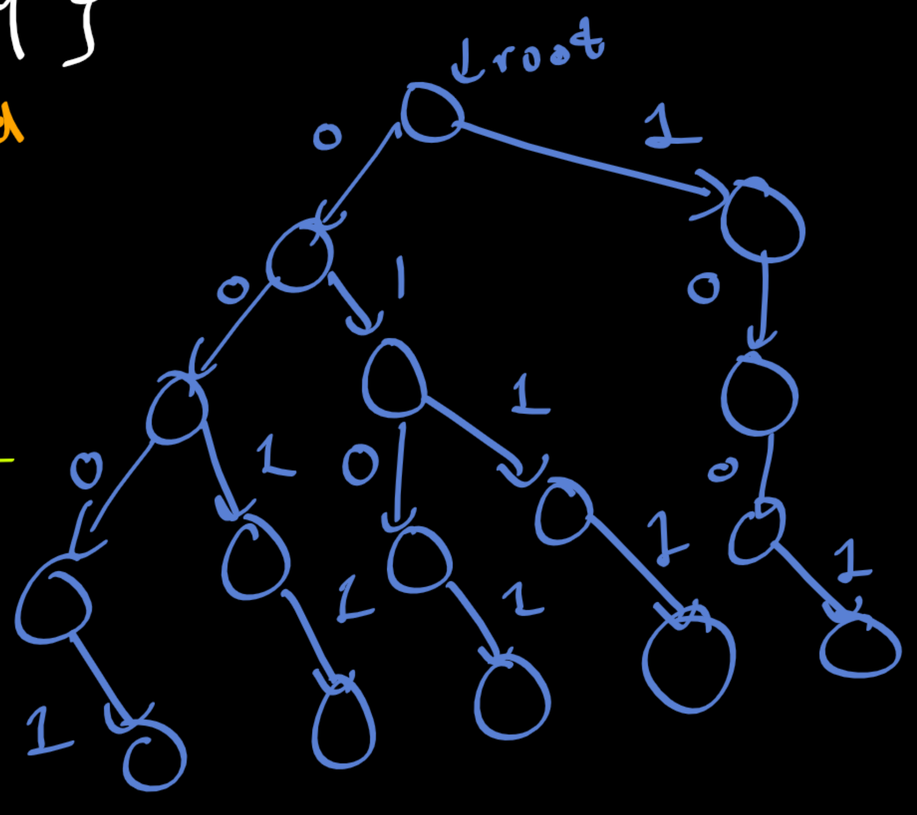
0 1 0 0
No 0 0 1 1
xor 0 1 1 1
4 ^ 3

③ ↓ {6,7} 0th

0 1 1 0
No 0 0 0 1
xor 0 1 1 1
6 ^ 1

④ ↓ {8,9} 2nd

1 0 0 0
No 0 1 1 1
1 1 1 1
8 ^ 7



{1, 3, 5, 7, 9} ↓ outer loop ↓ queries

0001 0011 0101 0111 1001

{ 0 | 1 | 2 | 3 }
6^1 | 4^3 | 8^7 | 7^1

↳ final Ans.

```java
public int[] maximizeXor(int[] nums, int[][] queries) {
    int[][] q = new int[queries.length][3];
    for (int i = 0; i < q.length; i++) {
        q[i] = new int[] { queries[i][0], queries[i][1], i };
    }

    Arrays.sort(q, (a, b) -> (a[1] - b[1]));
    Arrays.sort(nums);

    int[] res = new int[queries.length];
    int idx = 0;
    Node root = new Node();

    for (int i = 0; i < queries.length; i++) {
        // Insertion
        while (idx < nums.length && nums[idx] <= q[i][1]) {
            insert(root, nums[idx]);
            idx++;
        }

        if (idx == 0)
            res[q[i][2]] = -1;
        else
            res[q[i][2]] = search(root, q[i][0]);
    }
    return res;
}
```

Code for Max XOR Pair-II

{ search, insert & node are same as prev questions }

Prefix and Suffix Search {LC: 745 Hard}

Dict: {"apple", "banana"}

745. Prefix and Suffix Search

Hard 1046 330 Add to List Share

Design a special dictionary with some words that searchs the words in it by a prefix and a suffix.

Implement the `WordFilter` class:

- `WordFilter(string[] words)` Initializes the object with the `words` in the dictionary.
- `f(string prefix, string suffix)` Returns *the index of the word in the dictionary,* which has the prefix `prefix` and the suffix `suffix`. If there is more than one valid index, return **the largest** of them. If there is no such word in the dictionary, return `-1`.

query("ap", "na") → false

query("ban", "ple") → false

query("ap", "el") → false

query("ap", "le") → True

Approach is inspired from Manachers Algorithm.

"apple" Store every combination of suffix # prefix for every word in the trie. for "apple" it is shown

prefix	suffix
""	"apple"
"a"	"pple"
"ap"	"ple"
"app"	"le"
"appl"	"e"
"apple"	""

Now search for the same i.e q(pre, suf) → suf # pre

suffix # prefix

apple#

pple#a

ple#ap

le#app

e#appl

#apple

Stored in trie

```java
static class Node {
    private Node[] children = new Node[27];
    private int idx;

    boolean contains(char ch) {
        if(ch == '#') return children[26] != null;
        return (children[ch-'a'] != null) ;
    }

    void set(char ch) {
        if(ch == '#') children[26] = new Node();
        else children[ch-'a'] = new Node();
    }

    Node get(char ch) {
        if(ch == '#') return children[26];
        return children[ch-'a'];
    }

    void setIdx(int idx) {
        this.idx = idx;
    }

    int getIdx() {
        return idx;
    }
}

Node root;
```

```java
public void insert(Node root,String word,int idx) {
    Node curr = root;
    root.setIdx(idx);
    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            curr.set(ch);
        }
        curr = curr.get(ch);
        curr.setIdx(idx);
    }

}
```

```java
public int search(Node root,String word) {
    Node curr = root;

    for(int i=0;i<word.length();i++) {
        char ch = word.charAt(i);
        if(curr.contains(ch) == false) {
            return -1;
        }
        curr = curr.get(ch);
    }

    return curr.getIdx();
}
```

```java
public WordFilter(String[] words) {
    root = new Node();

    for(int idx=0;idx<words.length;idx++) {
        String word = words[idx];
        for(int i=0;i<word.length();i++) {
            insert(root,word.substring(i) + "#" + word,idx);
        }

        insert(root,"#" + word,idx);
    }
}

public int f(String prefix, String suffix) {
    return search(root,suffix + "#" + prefix);
}
```

Why we not did

insert(root, word.substring(i)+"#"+word.substring(0,i), idx)

We included the complete word as prefix because we can have overlapping prefix & suffix in query. for instance, we have the word "apple" & query is

query("appl", "pple");

prefix

suffix

Here, we have to store pple#apple for overlapping suffixes & prefixes.

If we insert only prefix then in trie, we will have suffix # prefix

pple # a ⇒ This will give WA

Count Pairs with XOR in a Range {LC: 1803 Hard}

1803. Count Pairs With XOR in a Range

Hard 278 11 Add to List Share

Given a **(0-indexed)** integer array `nums` and two integers `low` and `high`, return *the number of* ***nice pairs***.

A **nice pair** is a pair `(i, j)` where `0 <= i < j < nums.length` and `low <= (nums[i] XOR nums[j]) <= high`.

Example 1:

```
Input: nums = [1,4,2,7], low = 2, high = 6
Output: 6
Explanation: All nice pairs (i, j) are as follows:
    - (0, 1): nums[0] XOR nums[1] = 5
    - (0, 2): nums[0] XOR nums[2] = 3
    - (0, 3): nums[0] XOR nums[3] = 6
    - (1, 2): nums[1] XOR nums[2] = 6
    - (1, 3): nums[1] XOR nums[3] = 3
    - (2, 3): nums[2] XOR nums[3] = 5
```

Do later