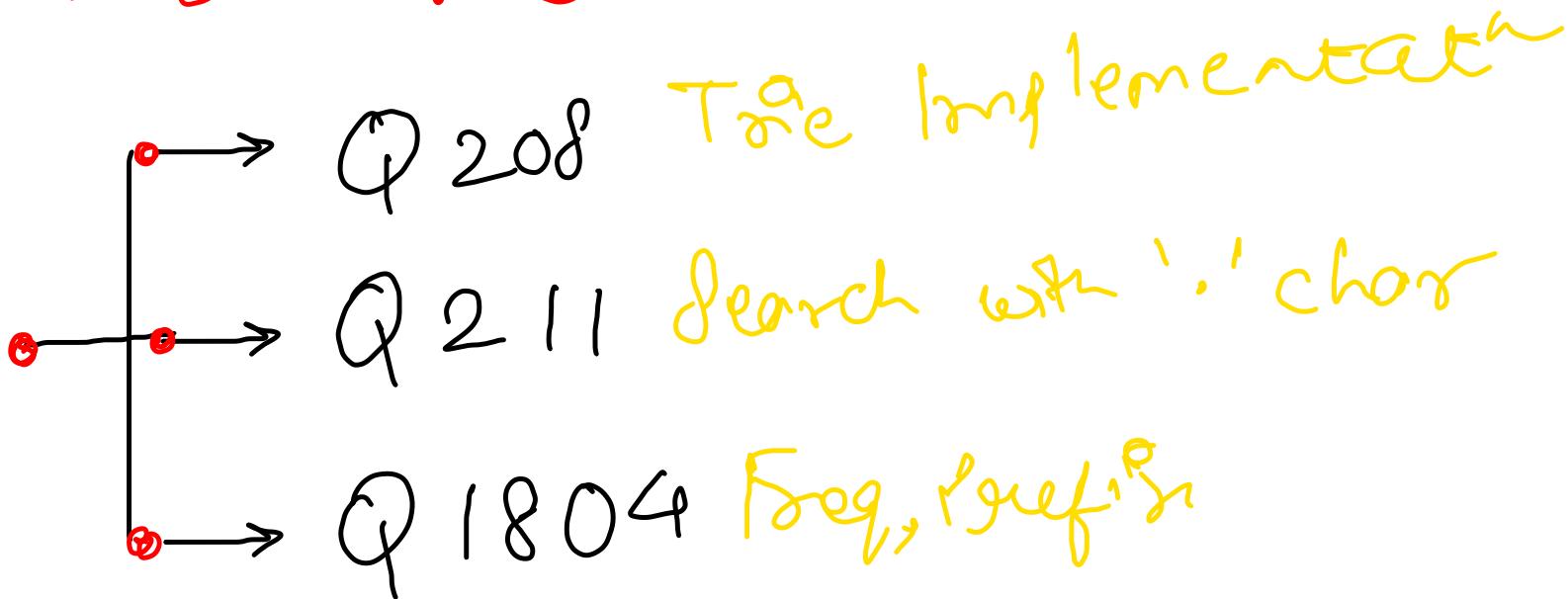


TRIE IMPLEMENTATION



Advantages & Disadvantages
of Trie over Hashing
Applications of Tries

Autocomplete System
Word Suggestion System
Spellchecker / Autocorrect
Prefix Matching
Searching { Phone Directory, Dictionary }

Dictionary Problems

~~Q720~~ longest Word in Dictionary

~~Q677~~ weighted Prefix search

~~Q648~~ Replace Word with Prefix

~~Q14~~ longest Common Prefix

Q676 Magic Dictionary

shortest Unique Prefix

~~GFG~~ (Q1698) Count Distinct Substrings

~~HN~~ GFG Most Frequent Word

Q1268 Search Suggestion Systems (DS)

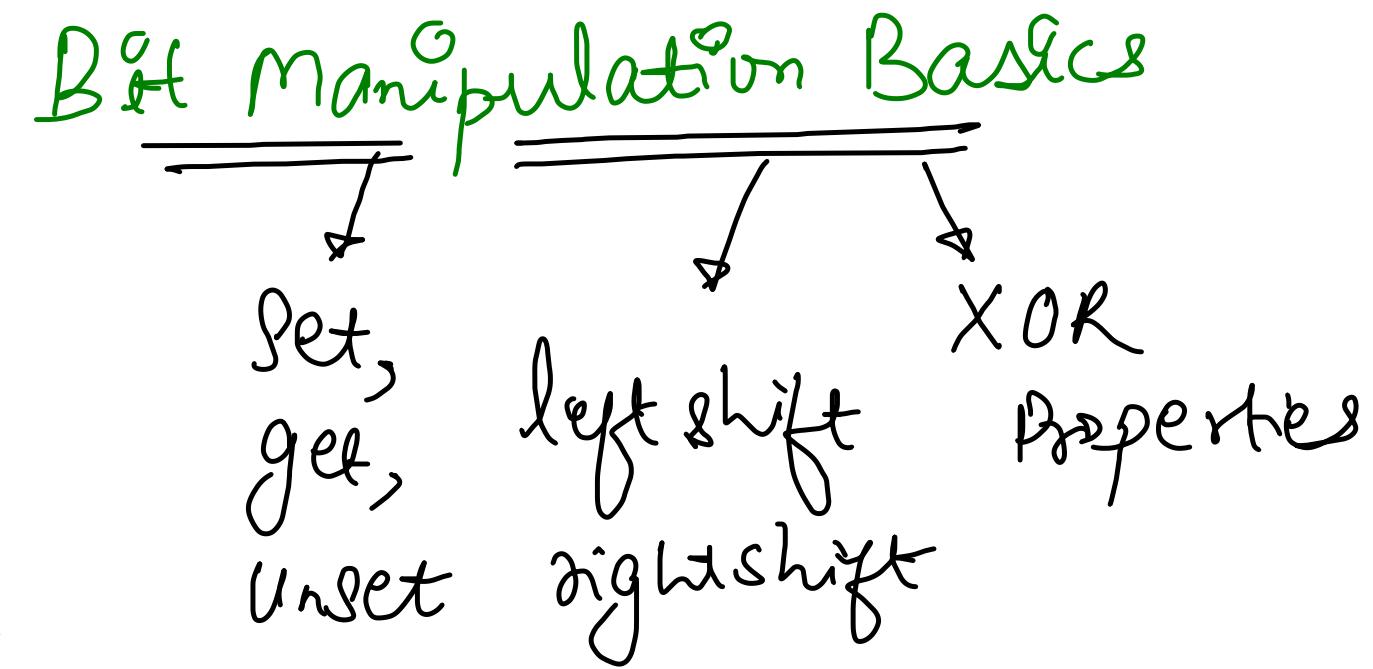
Q1032 Stream of Characters → Hard

~~Q212~~ Word Search - II

Q336 Palindrome Pairs → Hard

Q745 Prefix & Suffix Search → Hard

XOR Problems



Q421 Maximum XOR Pair - I

Q1707 Maximum XOR Pair - II

Q1803 XOR Pairs in Range

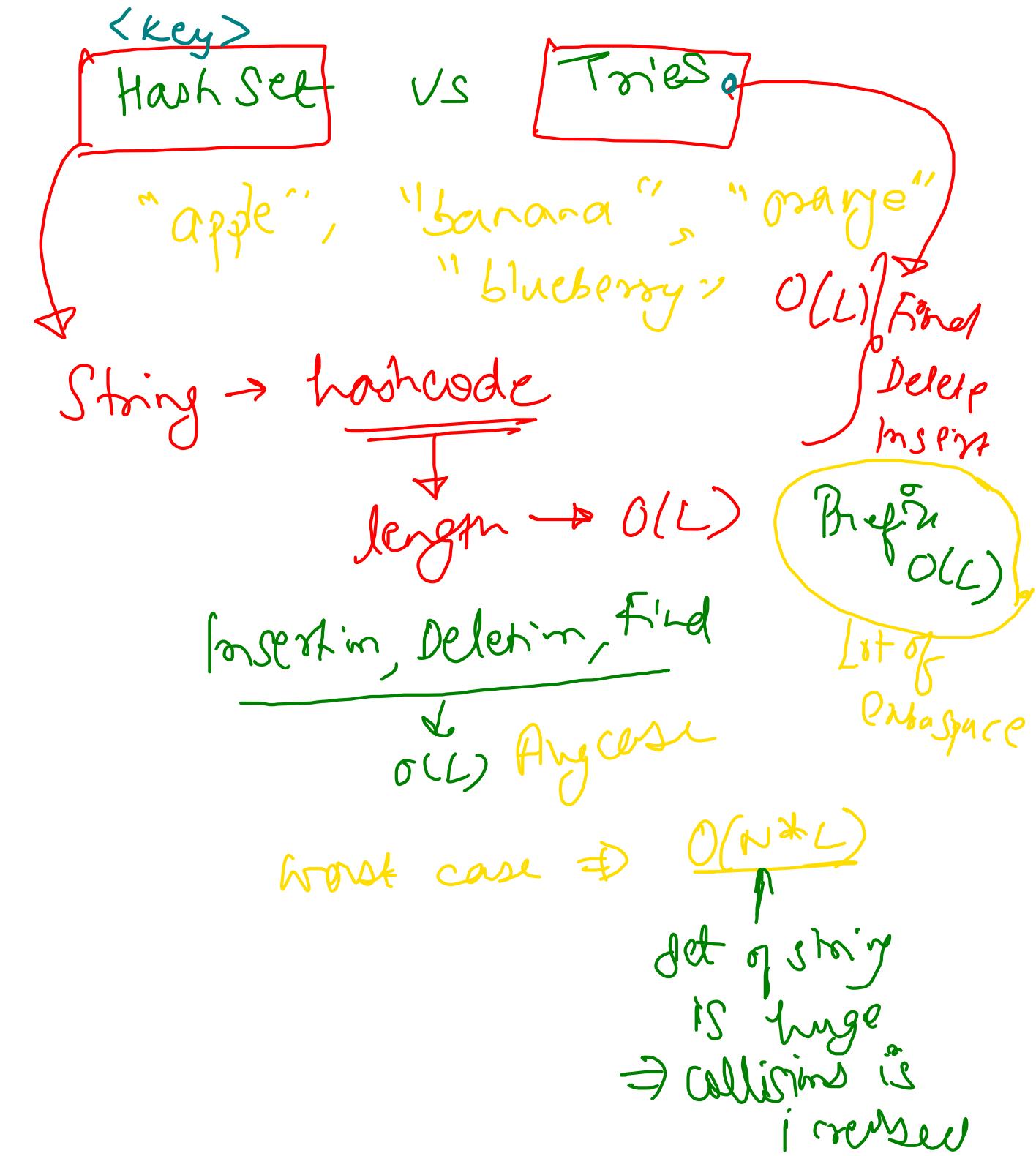
QFG Subarrays with XOR < K

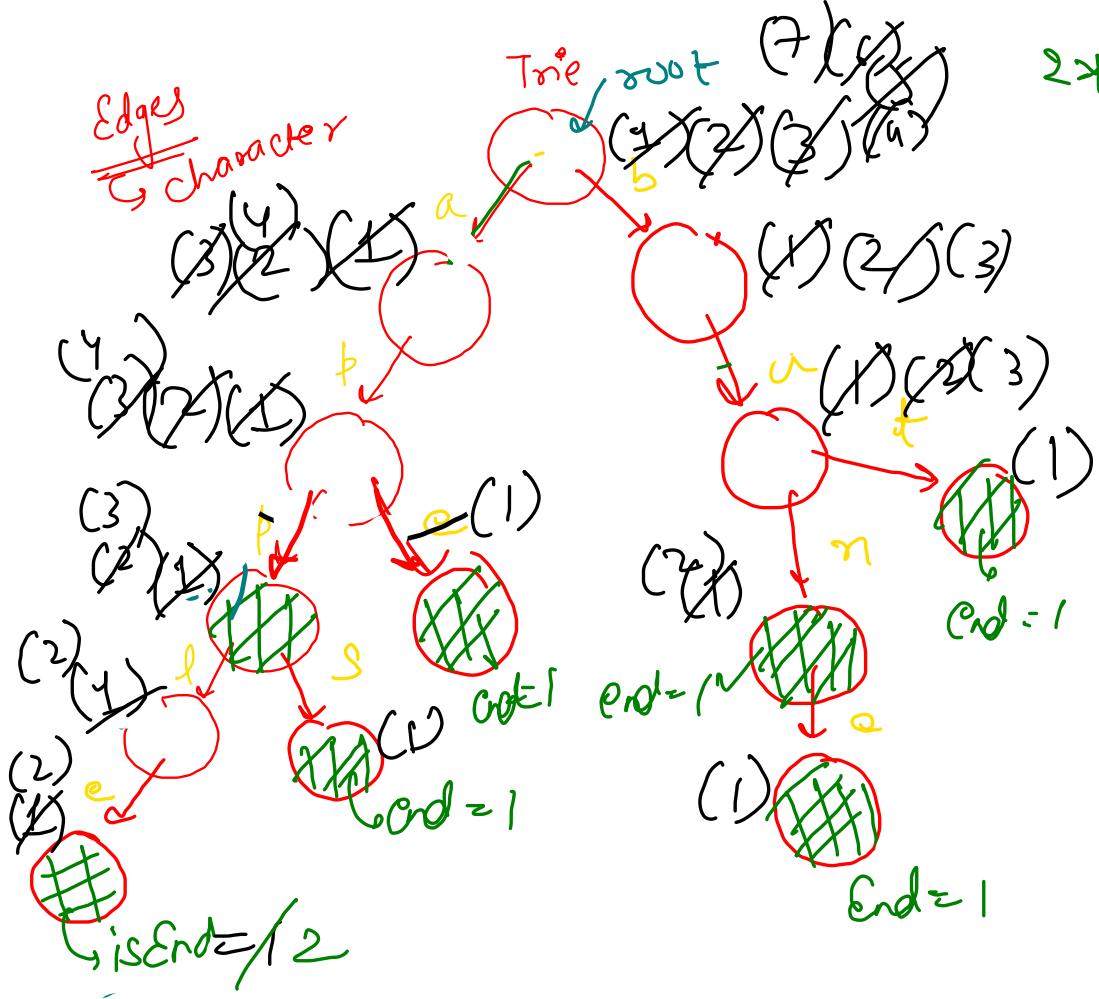
QFG Unique Rows in Boolean Matrix

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.

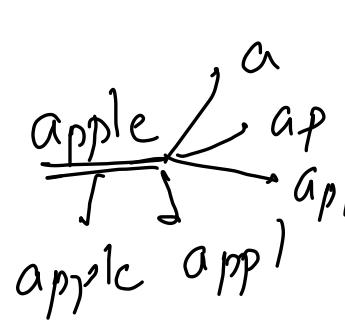




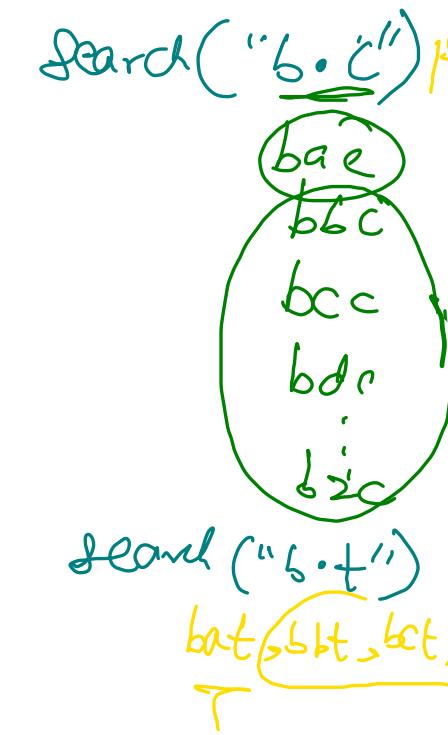
`printprefix("ap")` **DFS**: Breadth

{ ape, app, apple, apls, apps }

2 → insert("apple")
 insert("bank")
 insert("bad")
 insert("ban")
 insert("app")
 insert("apps")
 search("apple") True
 search("bank") False
 search("ap") False
 prefix("apple") True
 prefix("bank") False
 prefix("ap") True
 isEnd("ape")



delete("ban")
 End 2 \Rightarrow decrease End



End = 1
 delete
 Polym
 decrease End
 (End = 0) { Postorder }

No Bef ind { No Suffix }

```

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

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

    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;
    }
}

```

```

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();
}

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();
}

public boolean startswith(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 true;
}

```

Worst case

$T_{\text{ave}} \rightarrow$
 Insert
 Delete
 Find
 Begin
 For 1 String

$\Rightarrow O(L)$

Space for entire Trie
 $\Rightarrow O(NKL)$

~~Q21~~ ~~Leetcode~~

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

worst case

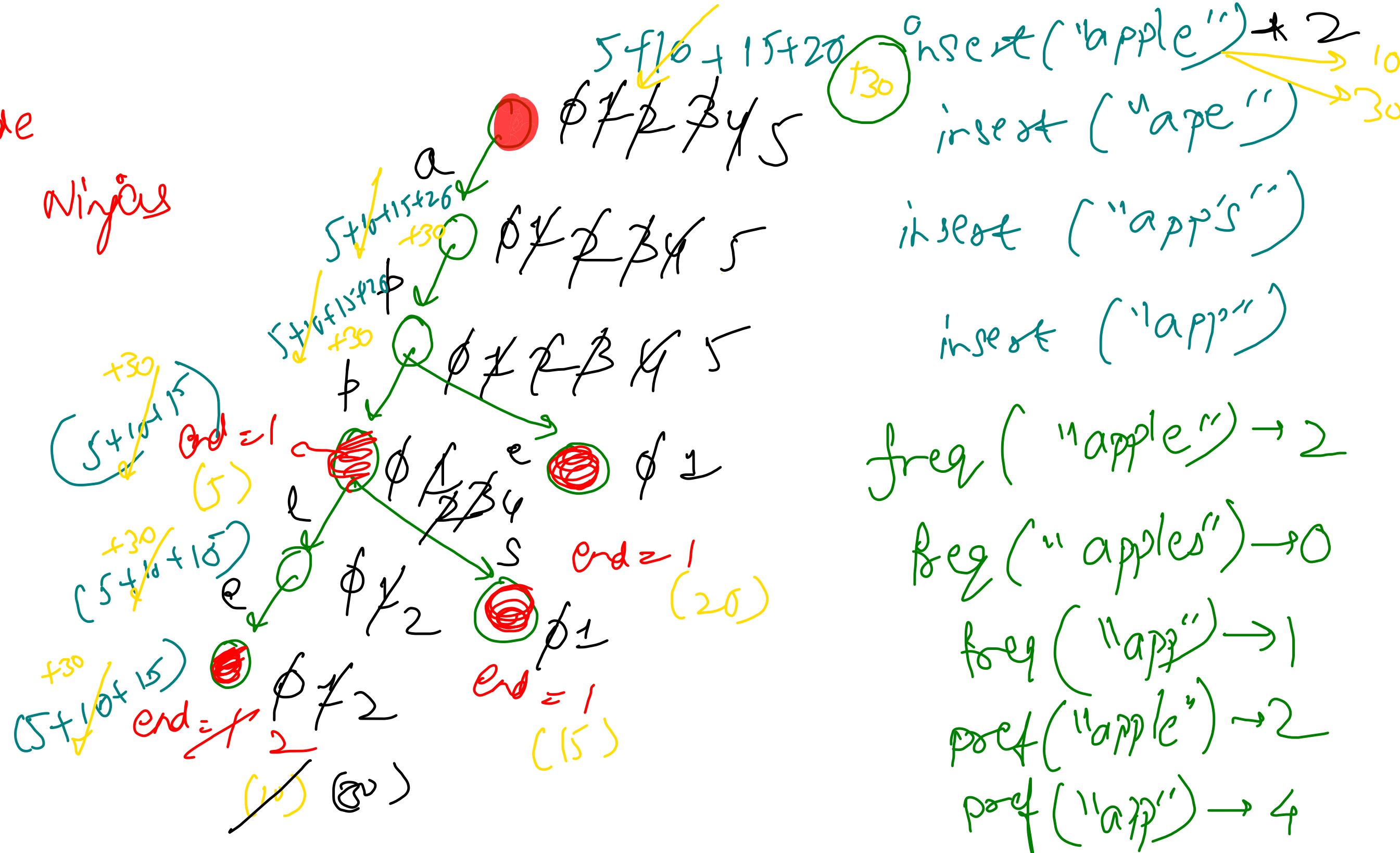
(26)
—
—

All are dots ('.')

~~Q1804~~

Hectcode

→ Coding Ninjas



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

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

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

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

    public int getFreq(){
        return isEnd;
    }

    public int getPref(){
        return prefixCount;
    }

    public void increaseFreq(){
        isEnd++;
    }
}
```

```
public void decreaseFreq(){
    isEnd--;
}

public void increasePref(){
    prefixCount++;
}

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

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

public void insert(String word) {
    Node curr = root;
    for(int i=0; i<word.length(); i++){
        curr.increasePref();

        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            curr.set(ch);

        curr = curr.get(ch);
    }

    curr.increasePref();
    curr.increaseFreq();
}
```

↗(L)

```
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.getFreq();
}
```

↗(L)

```
public int countWordsStartingwith(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.getPref();
}
```

↗(L)

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

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

    curr.decreasePref();
    curr.decreaseFreq();
}
```

~~O(?) MapSum Pair~~

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

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

    public int getVal(){
        return val;
    }

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

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

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

```
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;

        curr = curr.get(ch);
    }
    return curr.getVal();
}
```

```
public void insert(String word, int val) {
    int oldVal = search(word); O(L)

    Node curr = root;
    for(int i=0; i<word.length(); i++){
        curr.pref += (val - oldVal);

        char ch = word.charAt(i);

        if(curr.contains(ch) == false)
            curr.add(ch);

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

SL

Weighted Prefix Search

```
public int sum(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;

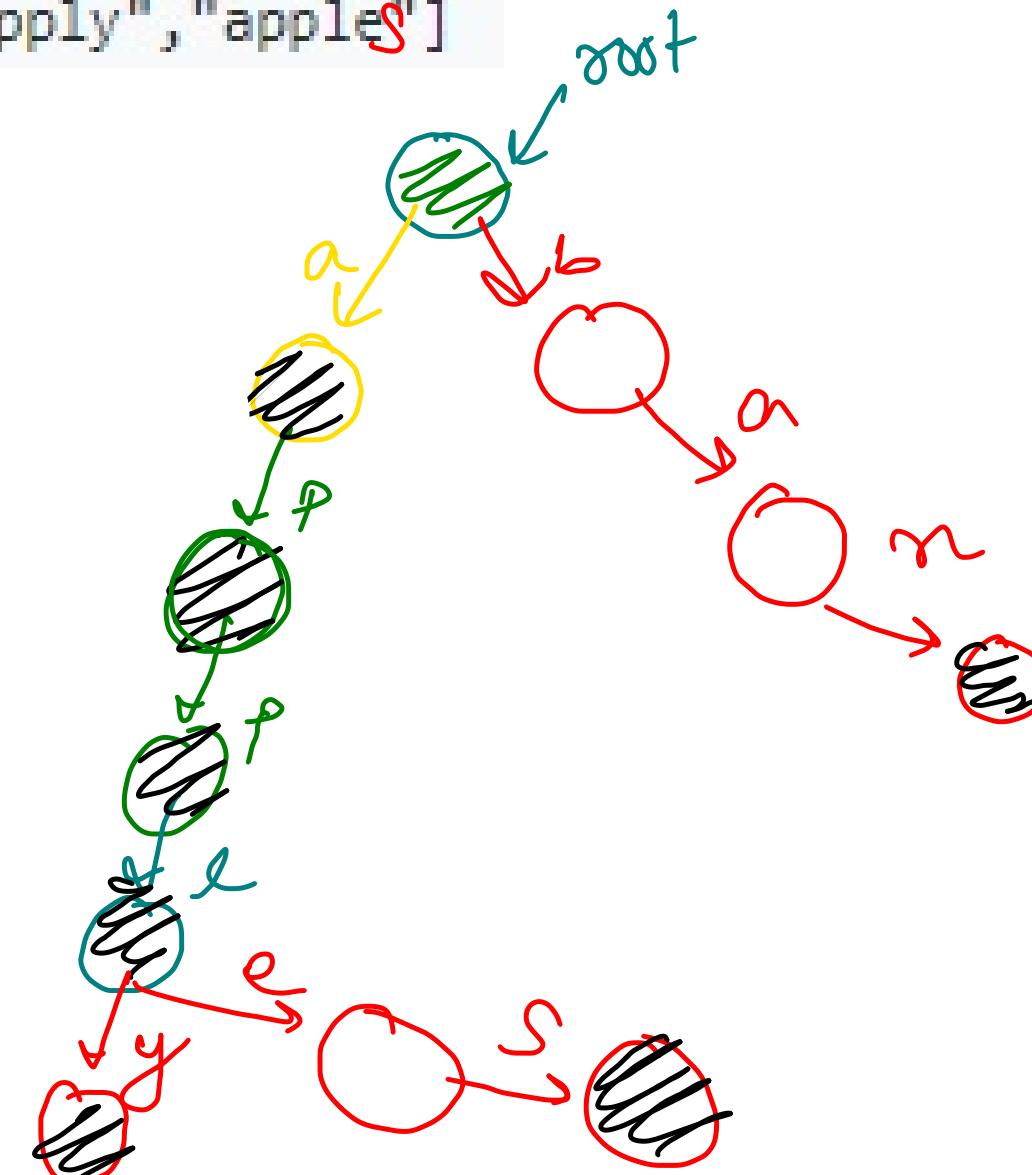
        curr = curr.get(ch);
    }
    return curr.pref;
}
```

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", "banana", "app", "appl", "ap", "apply", "apple"]`

DFS on
shaded nodes only
(end of word = true)
↓
Deepest node



longest word = " " ⑥

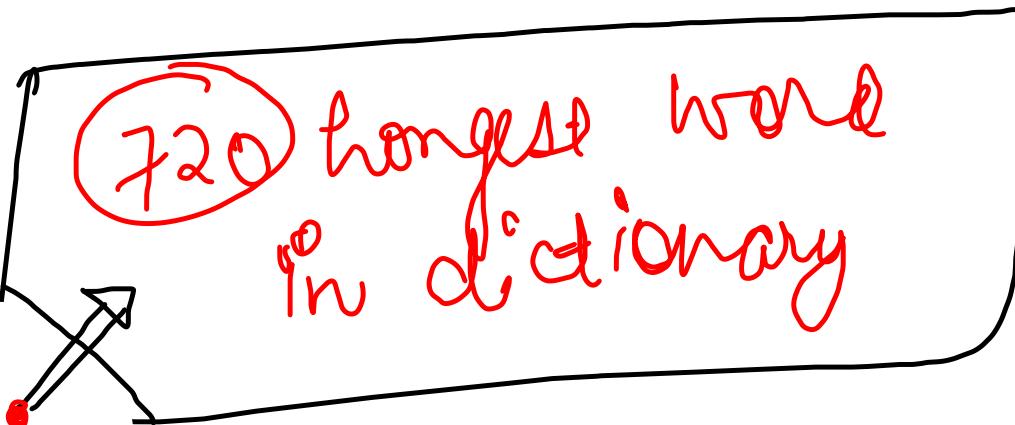
= "a" ①

= "ap" ②

= "app" ③

= "appl" ④

= "apply" ⑤



720 longest word in Dictionary

```
String res = "";
public void DFS(Node root, String anssofar){
    if(root.isTerminal() == false) return;

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

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

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

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

```
public static class Node{
    private Node[] children = new Node[26];
    private boolean isTerminal = false;

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

    public boolean isTerminal(){
        return isTerminal;
    }

    public void setTerminal(){
        isTerminal = true;
    }

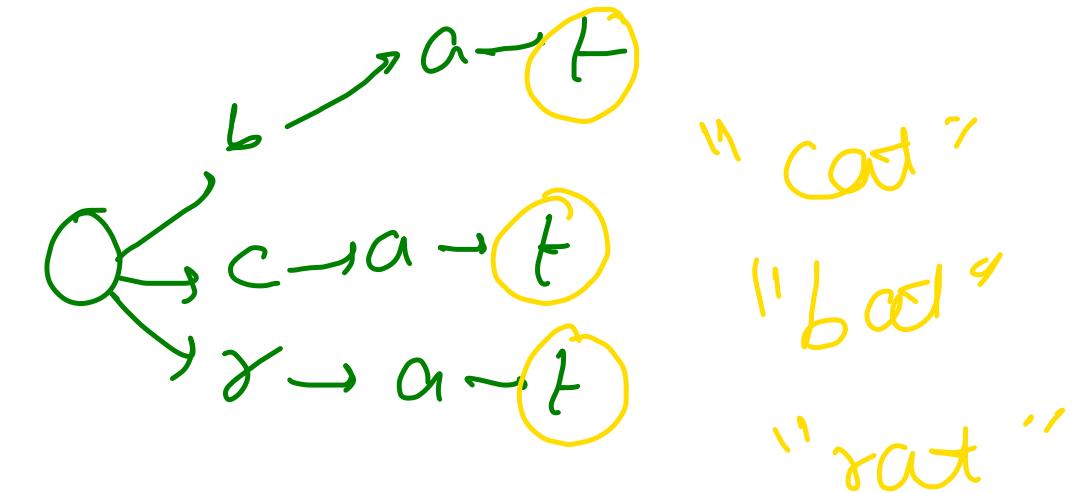
    public void add(char ch){
        children[ch - 'a'] = new Node();
    }

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



Replace words with shortest prefix

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



"the cattle was rattled by the battery"
 ↓ ↓ ↓ ↓ ↓
 "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.

Assumptions

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

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

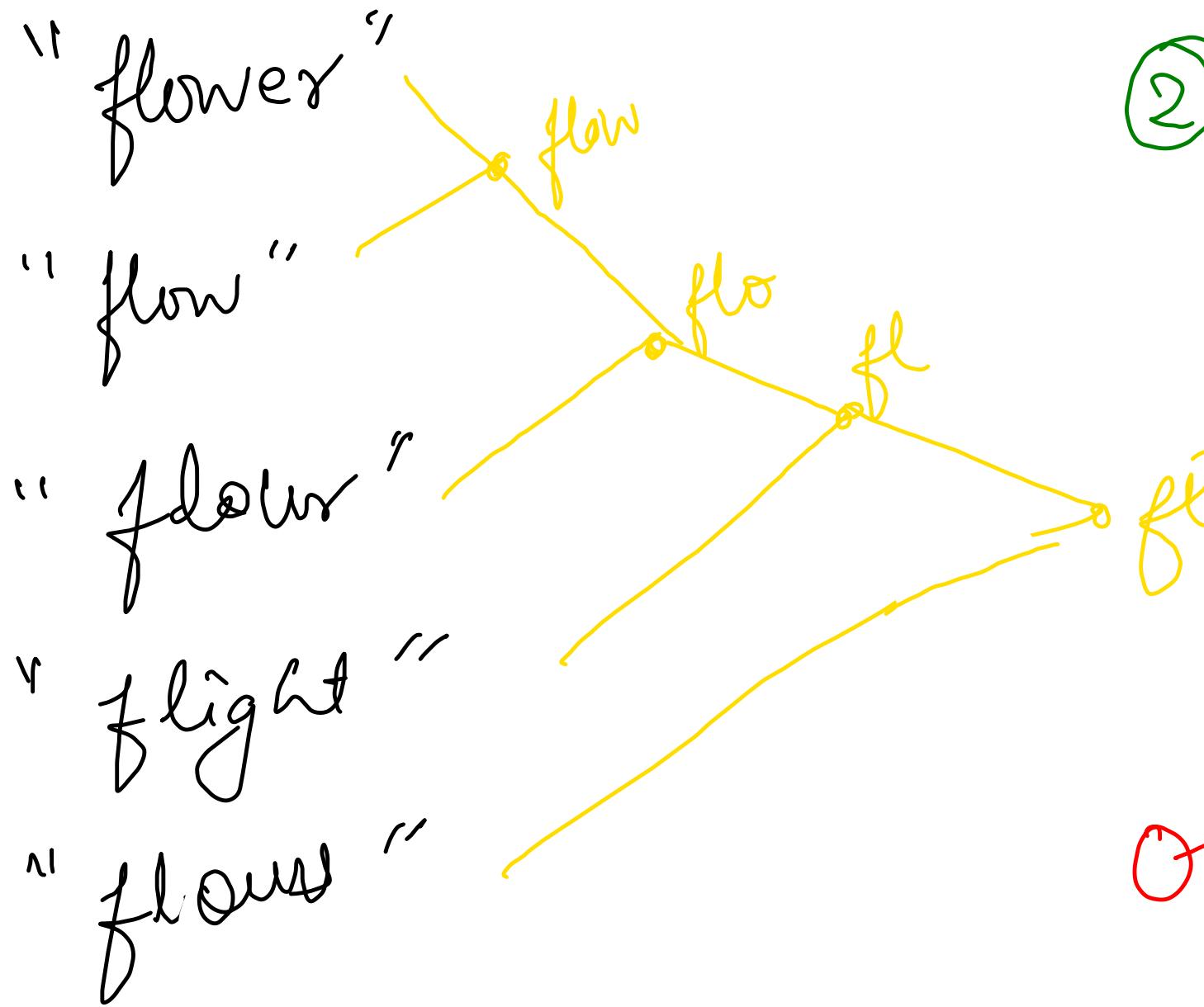
    return res.toString();
}
```

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

        if(curr.isTerminal() == true){
            return word.substring(0, i);
        }

        if(curr.contains(ch) == false)
            return word;
        curr = curr.get(ch);
    }
    return word;
}
```

Highest Common Prefix = {lowest common ancestor of shaded nodes}

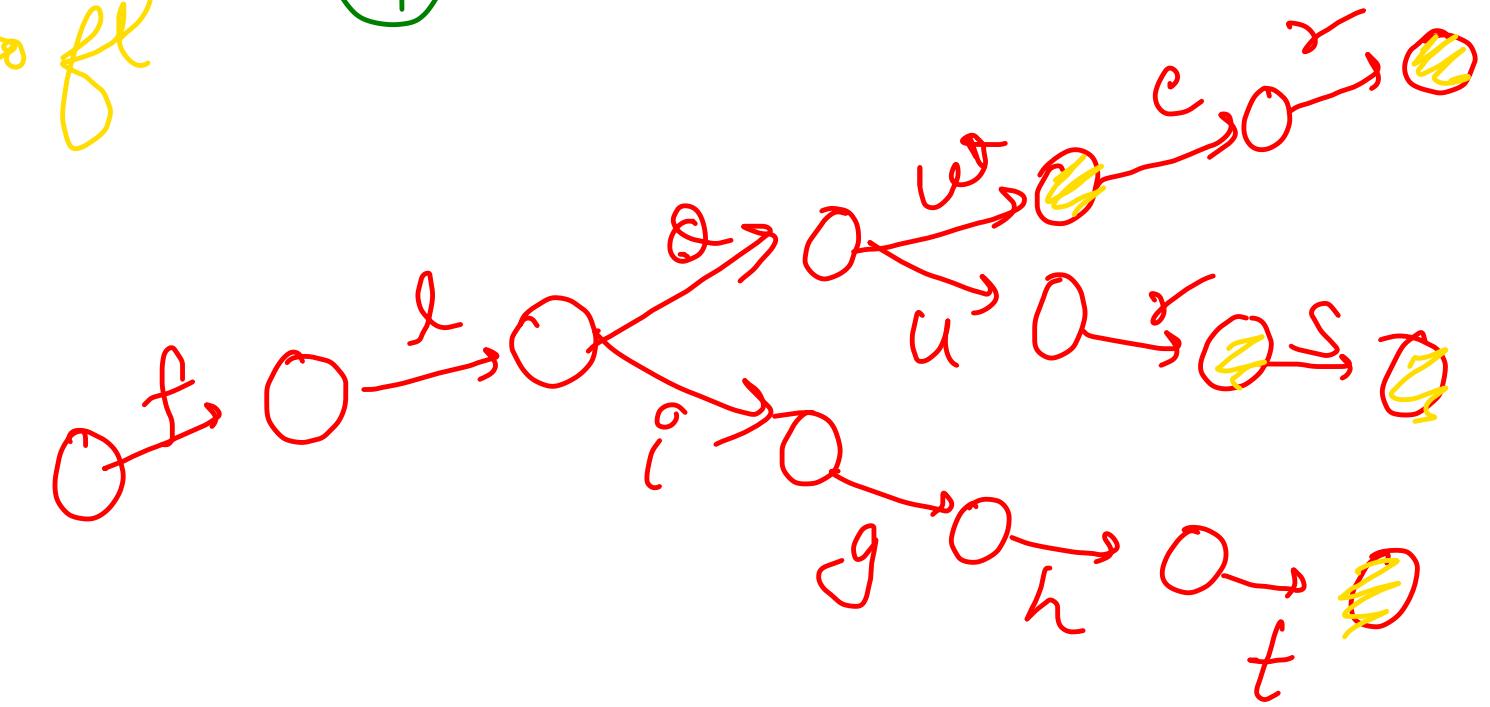


① Vertical scanning

② Horizontal scanning

③ Divide & Conquer

④ Trie \Rightarrow Queries



```

String res = "";
public void DFS(Node curr, String ans){
    int count = 0;

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

    if(curr.isTerminal == true){
        return;
    }

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

    if(count == 1) DFS(curr.get(child), ans + child);
}

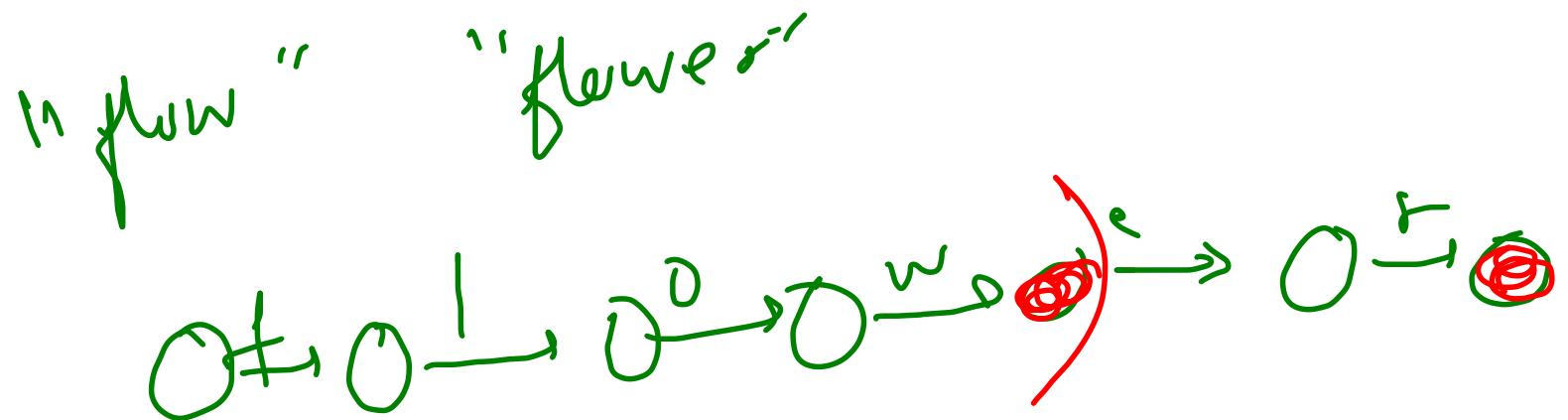
```

```

public String longestCommonPrefix(String[] strs) {
    Node root = new Node();
    for(String str: strs){
        insert(root, str);
    }

    DFS(root, "");
    return res;
}

```



(B)

Shortest Unique Prefix

[zebra, dog, duck, dove]

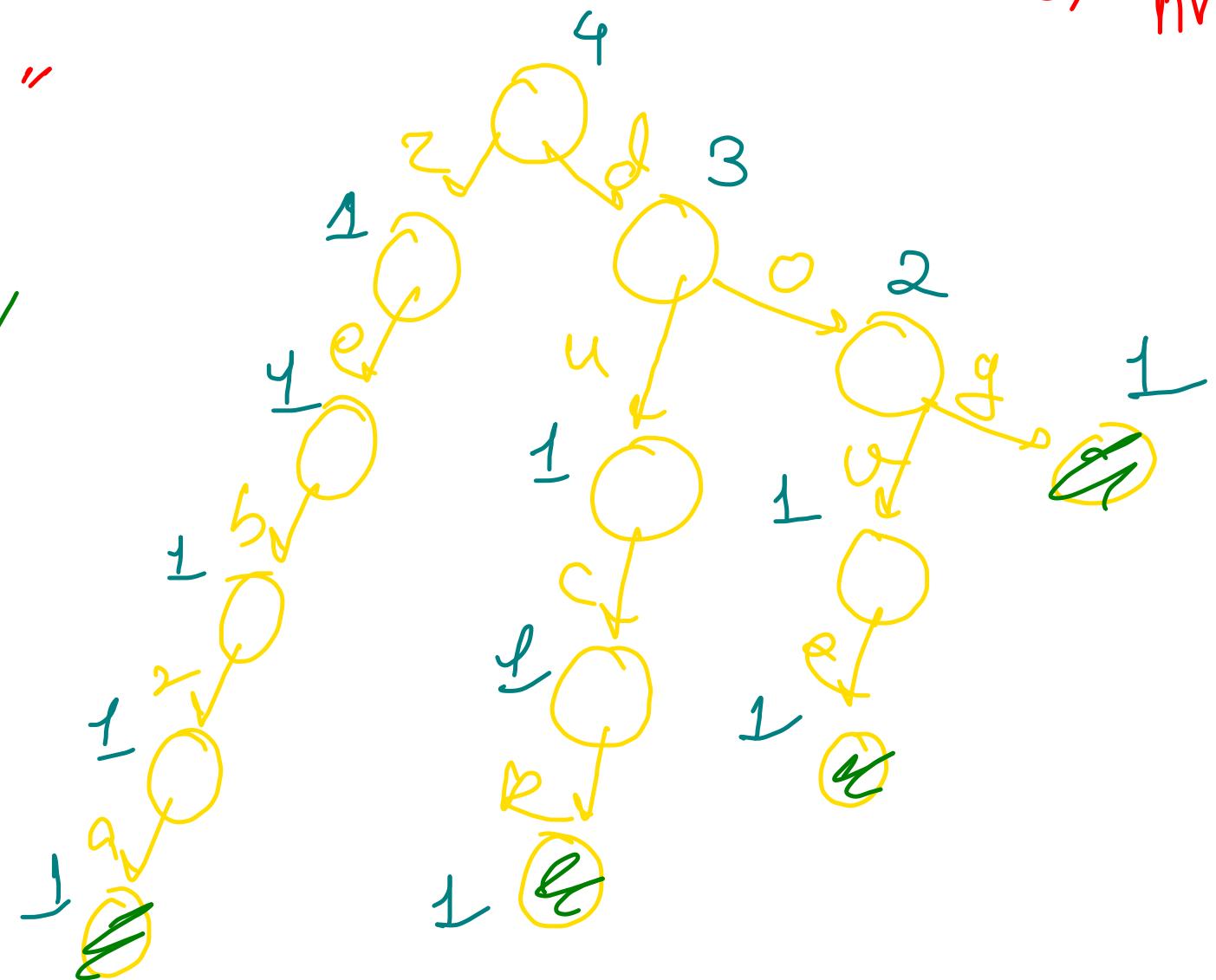
"z"
"d"
"du"
"dov"

Approach
Find first character
which have prefix
Count = 1

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

"dog" vs "dogs"

Answer will
always
exist



```

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

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

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

```

```

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

        if(root.prefCount == 1)
            return word.substring(0, i);

        root = root.get(ch);
    }

    return word;
}

```

```

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

        if(root.get(ch) == null){
            root.set(ch);
        }

        root.prefCount++;
        root = root.get(ch);
    }

    root.prefCount++;
    root.end++;
}

```

```

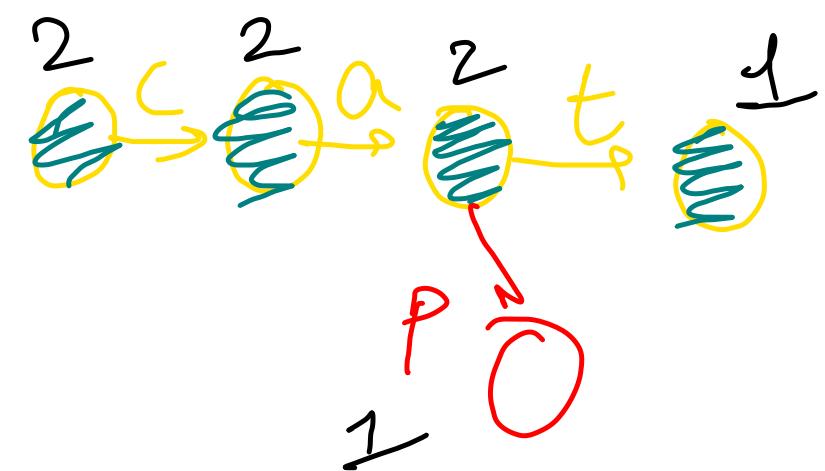
public String[] prefix(String[] A) {
    Node root = new Node();
    for(String str: A){
        insert(root, str);
    }

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

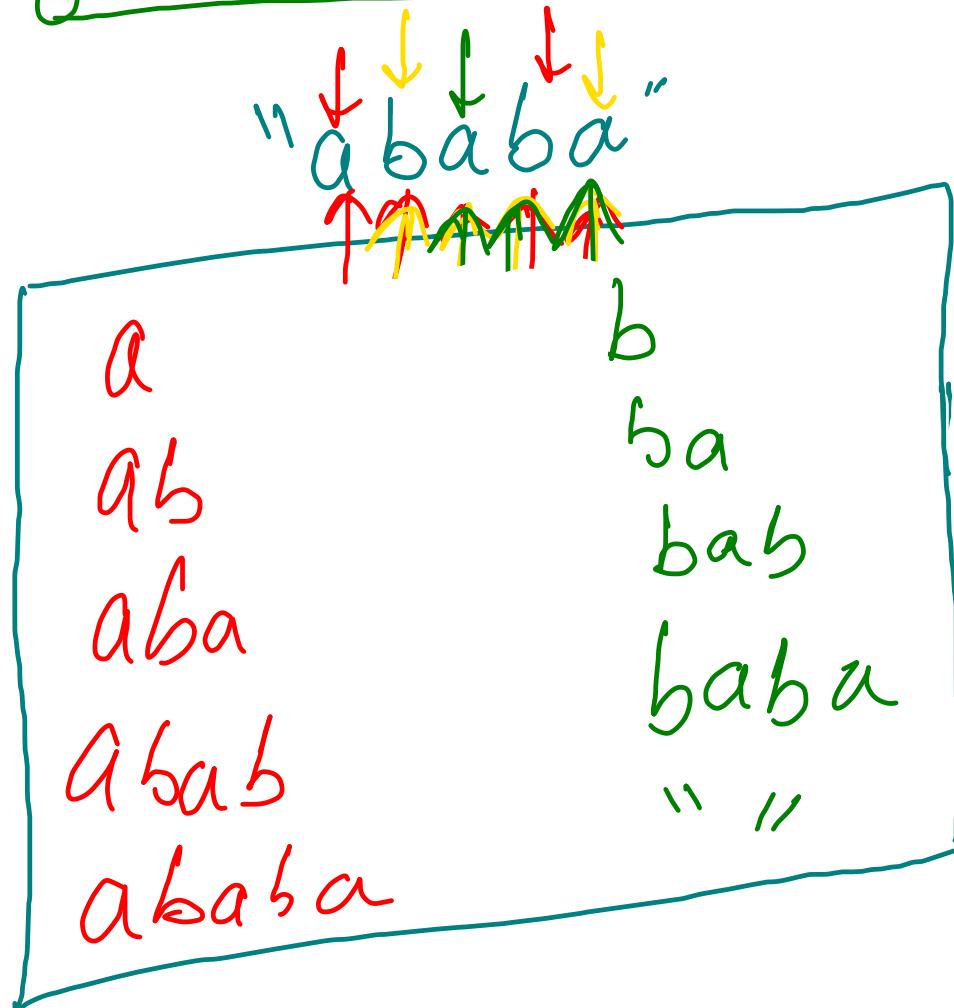
    return res;
}

```

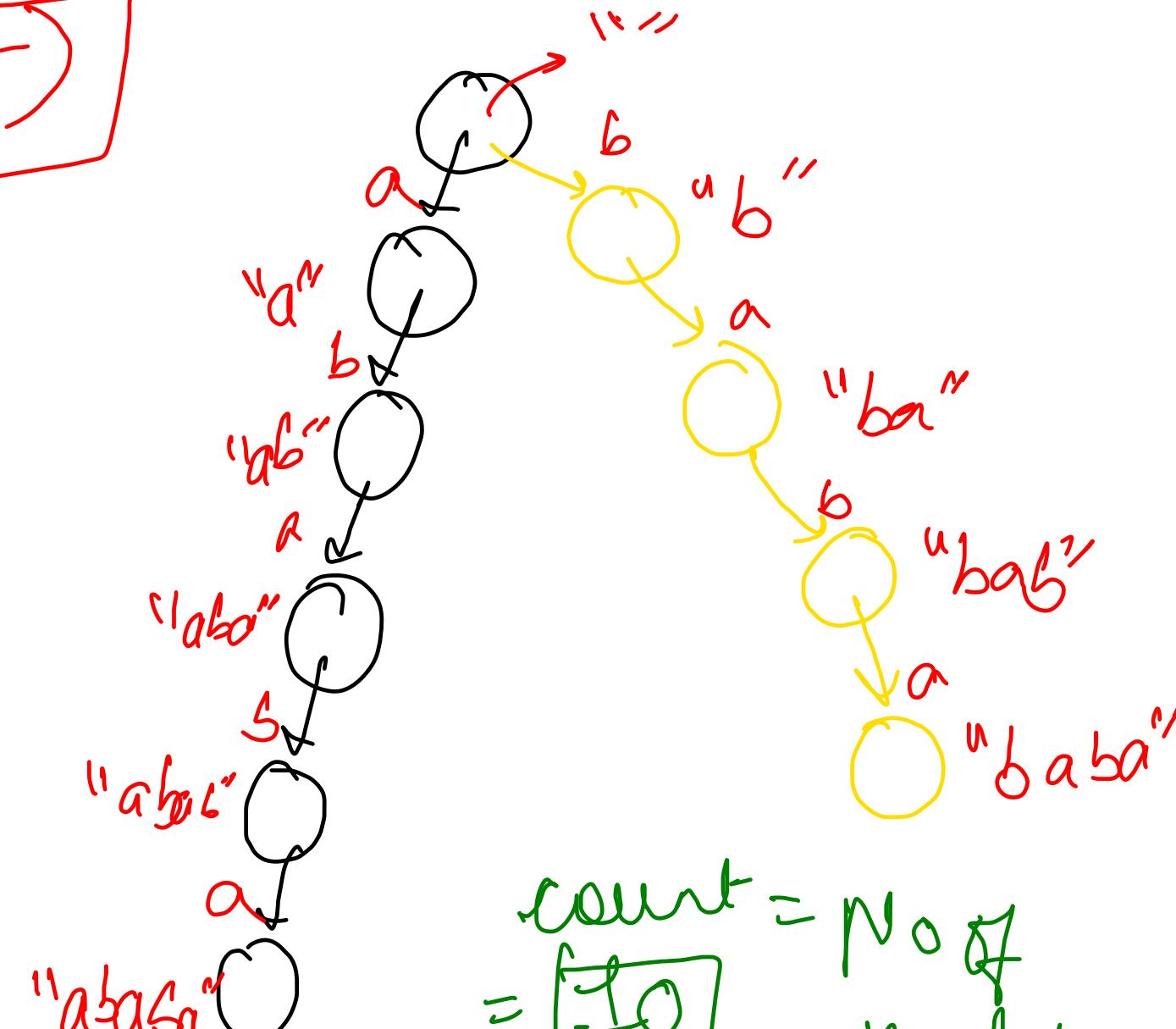
*Shortest Unique
Prefin*



Count Distinct substrings

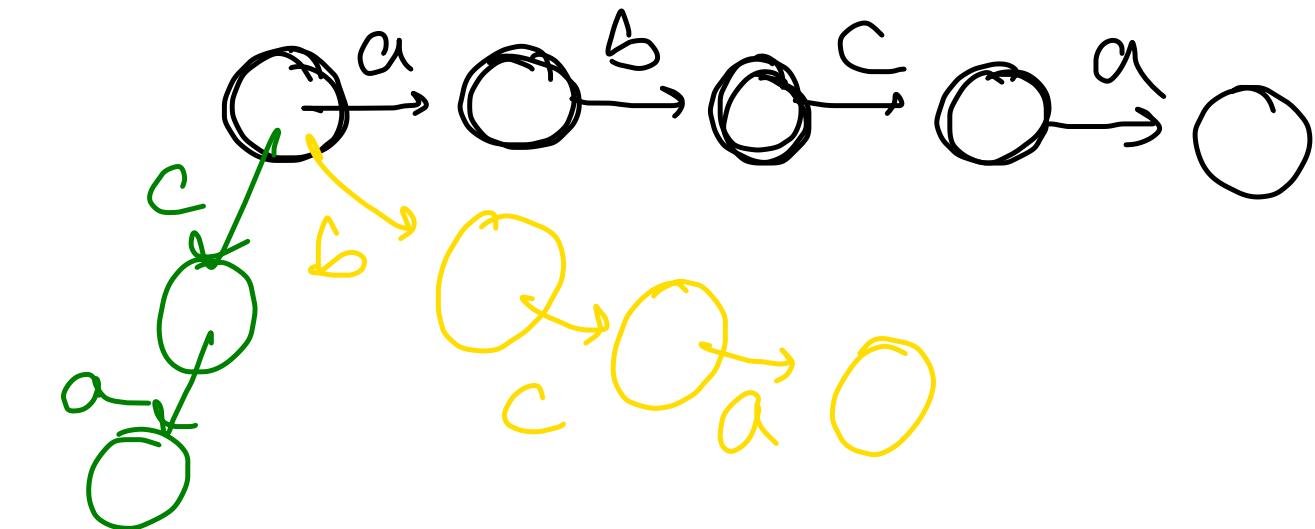


$O(N^2)$



count = No of nodes in tree
= 10

"abca"
 count = 7
~~7~~
~~8~~
~~10~~



$O(N^2)$

```

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 s)
{
    Node root = new Node();
    int count = 1;

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

Word search - II

① Word exists \Rightarrow getEnd()

② Word Prefix \Rightarrow root != null

```

public void exist(int r, int c, char[][] board, String ssof, Node root){
    if(root == null) return; // if word is prefix of at least one word
    if(root.getEnd() == true){
        // if word exists in dictionary or not
        res.add(ssof);
    }

    if(r < 0 || c < 0 || r >= board.length || c >= board[0].length
    || board[r][c] == '0')
        return;

    for(int call=0; call<4; call++){
        char ch = board[r][c];
        board[r][c] = '0';
        exist(r + x[call], c + y[call], board, ssof + ch, root.get(ch));
        board[r][c] = ch;
    }
}

```

```

public static class Node{ }

public void insert(Node curr, String word) {
    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();
}

int[] x = {-1, 1, 0, 0};
int[] y = {0, 0, -1, 1};
HashSet<String> res;

```

```

public List<String> findWords(char[][] board, String[] words) {
    res = new HashSet<>();

    Node root = new Node();
    for(String word: words){
        // insert in trie
        insert(root, word);
    }

    for(int i=0; i<board.length; i++){
        for(int j=0; j<board[0].length; j++){
            exist(i, j, board, "", root);
        }
    }

    List<String> ans = new ArrayList<>();
    for(String str: res){
        ans.add(str);
    }
    return ans;
}

```

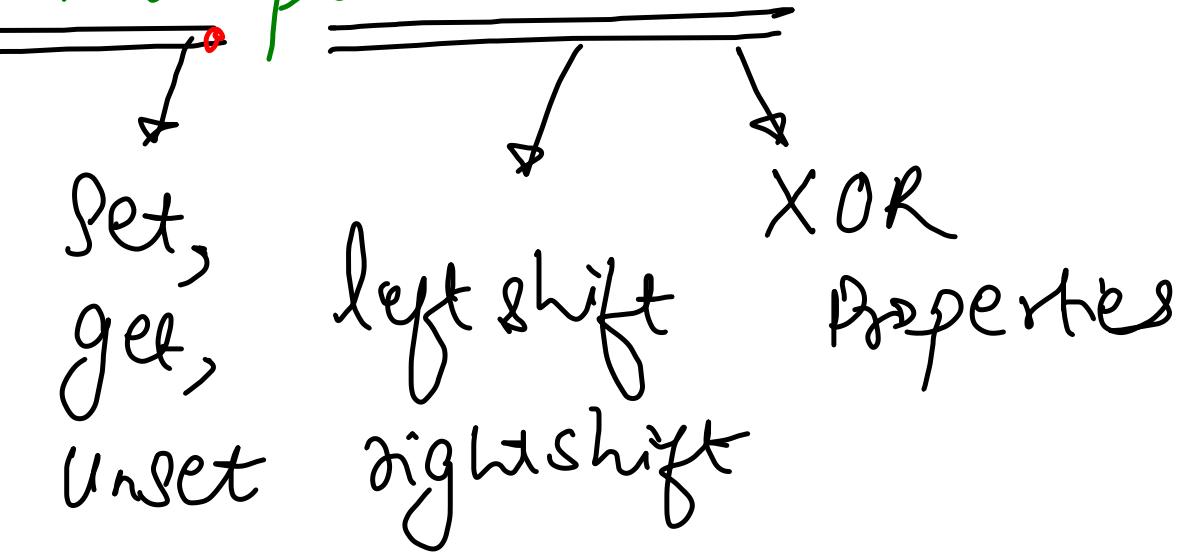
Magic Dictionary

Search suggestion System (SD)

Stream of Characters

Prefix & Suffix Search → Manacher's Algo

Bit Manipulation Basics



Magic Dictionary

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]
```



hello → FALSE

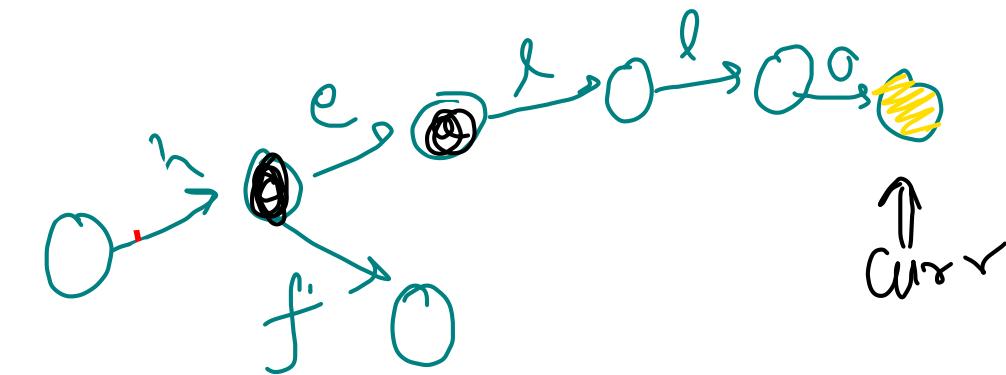
hhllo → hello TRUE

hell → FALSE

leetcoded → FALSE

Reencodz → FALSE

Search parameters { String word, int id, Node curr, boolean change }



hello
01234
0, false

hello
01234
1, false

for all char at \geq
if char \rightarrow replace { curr \rightarrow contains(ch)
if change == true
return false
else { id++ , curr.get(ch) , true } }

"Hello"
0,f \rightarrow 1,f \rightarrow 2,f \rightarrow 3,f \rightarrow 4,f \rightarrow 5, f \Rightarrow return false

"hello"
0,f \rightarrow 1,f \rightarrow 2,f \rightarrow 3,t \Rightarrow
return false

hell - false
end of word
== false

```
public boolean search(String word, int idx, Node curr, boolean change){
    if(idx == word.length()){
        if(change == true && curr.isTerminal() == true) return true;
        return false;
    }

    char ch = word.charAt(idx);
    if(curr.contains(ch) == true && search(word, idx + 1, curr.get(ch), change))
        return true;

    if(change == true) return false;

    for(char chn = 'a'; chn <= 'z'; chn++){
        if(chn == ch) continue;

        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);
}
```

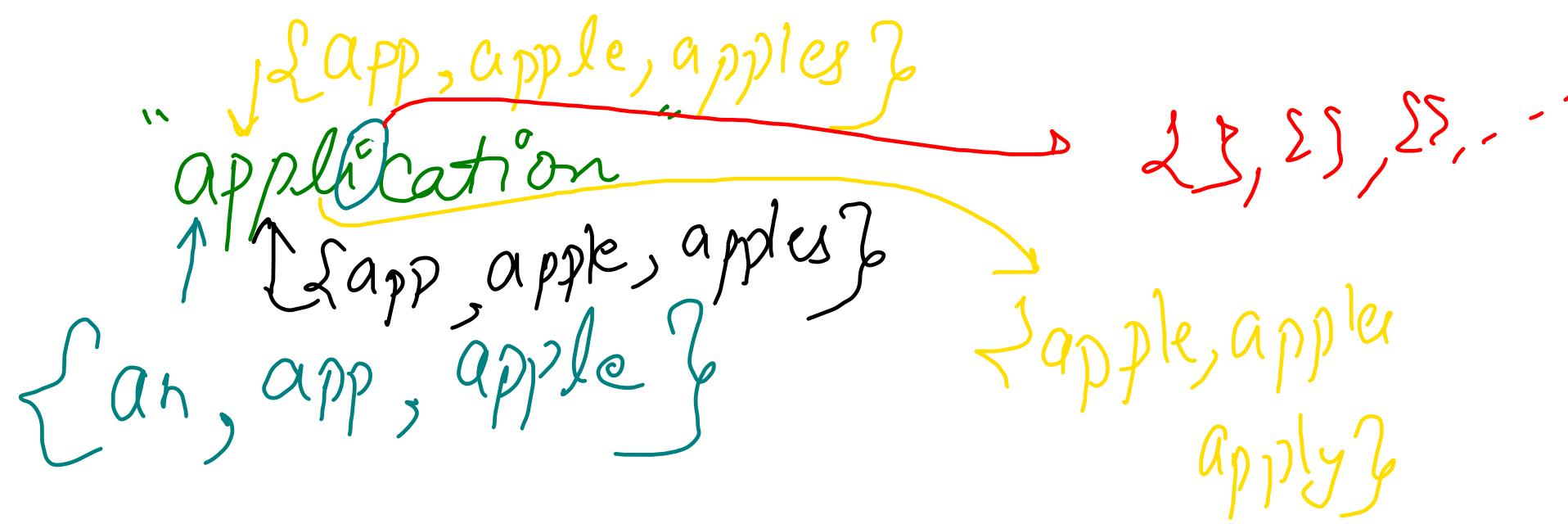
Search Suggestion System {System Design}

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.

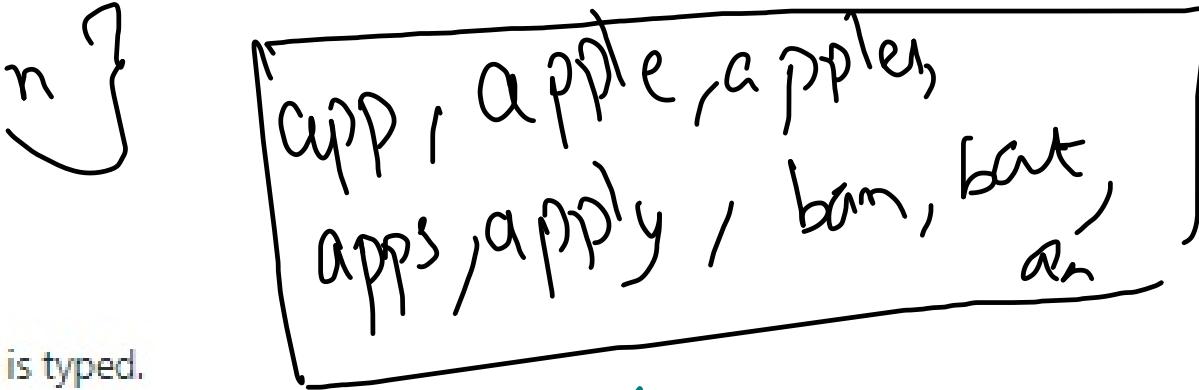
Suggested products should have common prefix with `searchWord`. If there are more than three products with a common prefix return the three lexicographically minimums products.

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



DFS → Preorder → Across 3 words

lexicographical



```
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;  
}
```

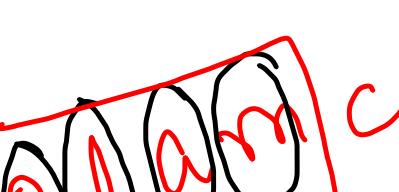
3 Almost lexicographically minimum Products

Q1032 Stream of characters

{ "apple", "male" } → Dictionary

insert (R+L)
search (R+L)

The diagram illustrates the search for the string "elam" in a stream of characters. The stream is represented as a sequence of characters with arrows indicating the search path. The search starts at the first character 'e'. The path leads through 'l', 'a', and 'm' to a final state labeled "true". The search fails for characters 'p', 't', 'd', 'c', and 'n'.

"clear"  "clear" 
suffix of online characters
in dictionary

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 $\text{words}[i] + \text{words}[j]$ is a palindrome.

Input: $\text{words} = ["abcd", "dcba", "lls", "s", "sssll"]$

Output: $[[0,1], [1,0], [3,2], [2,4]]$

Explanation: The palindromes are $["dcbaabcd", "abcddcba", "slls", "llssssll"]$

$$\checkmark \\ 0+1 \\ abcd + dcba \\ abcd dcba$$

$$\checkmark \\ 1+0 \\ dcba + abcd \\ dcba abcd$$

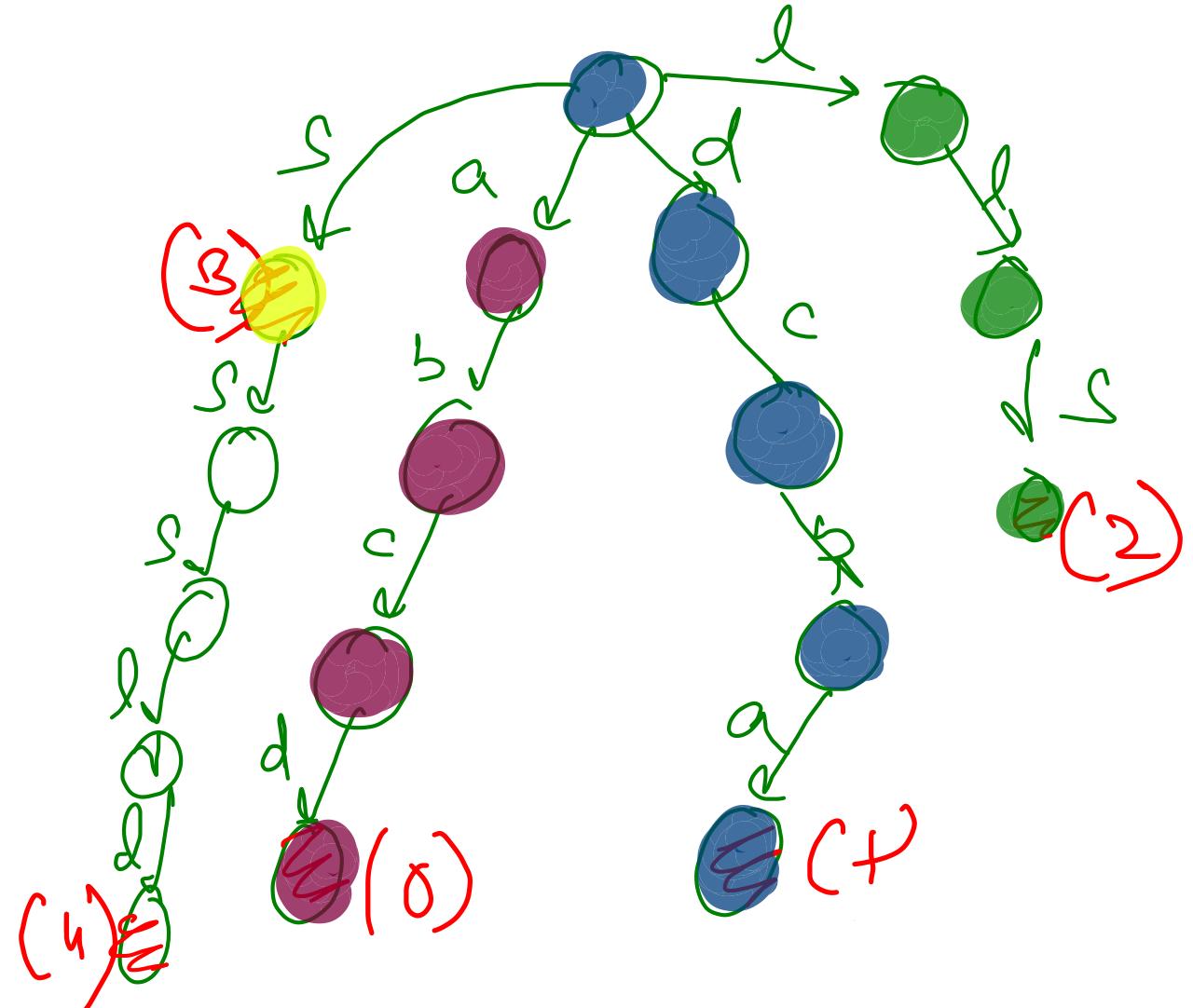
$$2 \rightarrow 4 \\ llS + sSSll \\ llSSSSll$$

$$\checkmark \\ 3+2 \\ S + llS \\ llS$$

$x+y \Rightarrow \text{palindrome}$

x is reverse of y
 Suffix of y = reverse of x & remaining
 pref. of x = reverse of y y should
 & remaining x should be also
 a palindrome

Palindrome Pairs $\{O(N^2)\}$



XOR Problems

Bit Manipulation Basics

↑

Set,
get,
unset

XOR

left shift
right shift

Properties

Q421 Maximum XOR Pair - I

Q1707 Maximum XOR pair - II

Q1803 XOR Pairs In Range

HW QFG Subarrays with XOR $\leq K$

QFG Unique Rows in Boolean Matrix

Bit Manipulation Basics

Integer { 4 bytes \rightarrow 32 bits }

9 \rightarrow 7 \rightarrow 5, 3 \rightarrow 1

1001 0111 0101 0011 0001
 $2^3 2^2 2^1 2^0$ $2^3 2^2 2^1 2^0$ $2^3 2^2 2^1 2^0$

left shift \Rightarrow 0001 \Rightarrow $x \ll 2 = x * 2^2$
 $x \ll 2 = 0100$

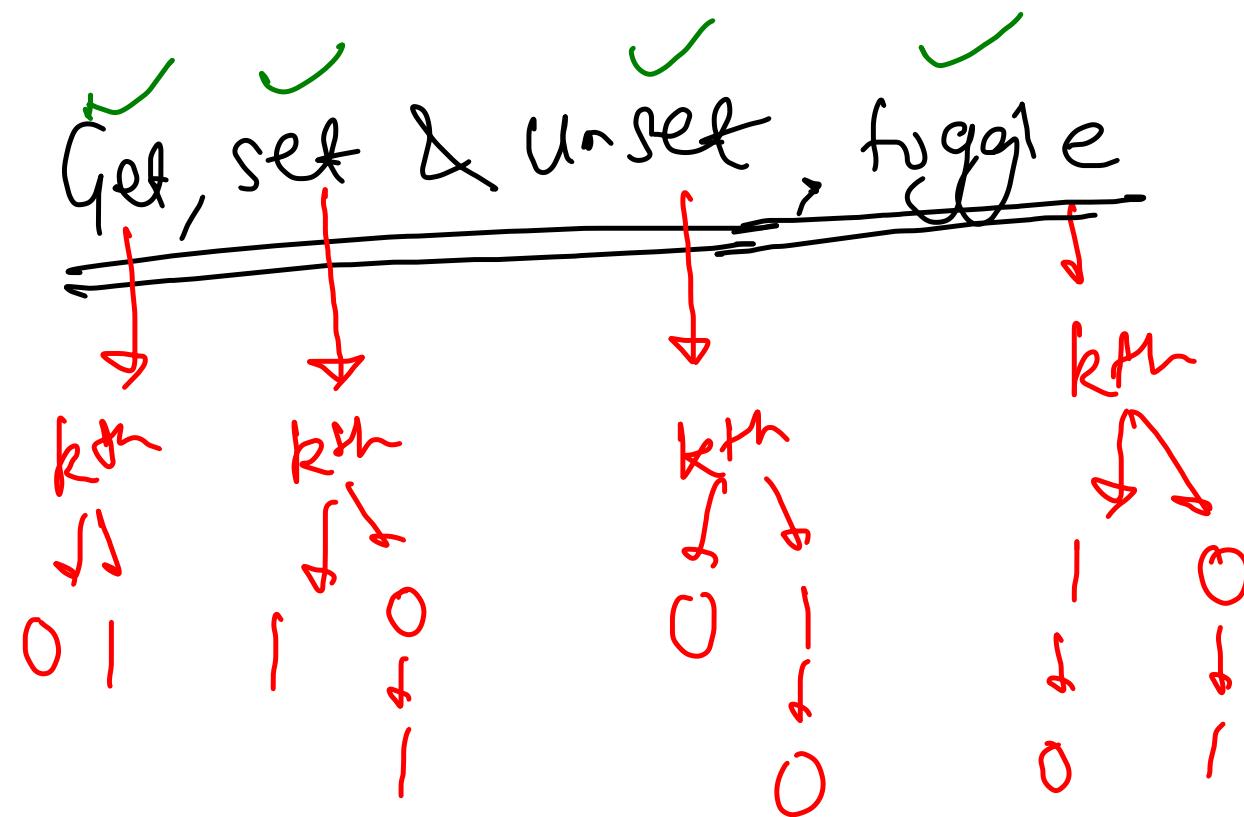
0001 \Rightarrow $\overset{0001}{\underset{2^0}{\curvearrowright}} \ll 3 = \frac{1000}{2^3}$
 $x \ll 3 = x * 2^3$

right shift \Rightarrow 0101 \Rightarrow 0010
(5) (2)
 $5 \gg 1 = 5/2 = 2$

0101 \Rightarrow 0000
 $5 \gg 2 = 5/2^2 = 5/4 = 1$

left shift $\rightarrow x \ll k = x + 2^k$

right shift $\rightarrow x \gg k = x / 2^k$



$$\begin{aligned} (\&) &= 1 \\ (\&0) &= 0 \\ 0\&1 &= 0 \\ 0\&0 &= 0 \end{aligned}$$

Get

$$no(x) = 5$$

0101
and 2nd 1st 0th

$$k = 2^{\text{nd}} \text{ bit}$$

0101

$$\begin{array}{r} 0001 \ll 2 = 0100 \\ \text{AND} \quad \underline{0100} \\ 0100 \end{array}$$

(Right to Left)
(0-based)

$$k = 3^{\text{rd}} \text{ bit}$$

0101

$$\begin{array}{r} 000k\ll 3 \quad 1000 \\ \text{AND} \quad \underline{0000} \\ 0000 \end{array}$$

$x \& (1 \ll k)$
 $\downarrow = 0$
 $\Rightarrow \text{Set}(1)$

$x \& (1 \ll k)$
 $= 0$
 $\Rightarrow \text{unset}(0)$

Set

$$x = 5$$

$$0101$$

$k = 2^{\text{nd}} \text{ bit}$

$$0001 \ll 2 = 0100$$

(OR)

$$\overbrace{0101}^{0101}$$

$$\begin{array}{rcl} 0001 & = & 1000 \\ \text{LC3} & & \hline \text{OR} & & 1101 \end{array}$$

$x \lceil (1 \ll k)$

bitwise OR

$$\begin{array}{rcl} 110 & = & 1 \end{array}$$

$$\begin{array}{rcl} 110 & = & 1 \end{array}$$

$$\begin{array}{rcl} 011 & = & 1 \end{array}$$

$$\begin{array}{rcl} 010 & = & 0 \end{array}$$

unset

0101

2nd bit

mask

$\approx_2 (1_{2<1})$

$\approx_2 (010b)$

= 1011

$$\begin{array}{r} 1011 \\ \hline 0001 \end{array}$$

AND

0101

3rd bit

0111

$$\begin{array}{r} 0101 \\ \hline 0101 \end{array}$$

AND

mask = $\approx_2 (1_{2<3})$

$\approx_2 (1000)$

= 0111

unset \Rightarrow

$x \& (\approx_2 (1_{2<1}))$

negative
tilde

~~toggle~~

0101
 $k = 2^{\text{nd}}$

0100

$\begin{array}{r} 0001 \\ \hline 0001 \end{array}$

0101
 $k = 3^{\text{rd}}$

1000

$\begin{array}{r} 1101 \\ \hline 1101 \end{array}$

toggle $\rightarrow x \wedge (1 \ll \text{mask})$

XOR

$1 \wedge 0 = 1$

$0 \wedge 1 = 1$

$1 \wedge 1 = 0$

$0 \wedge 0 = 0$

XOR Properties

~~①~~ Anything \neq Anything

~~②~~ Something \neq Something

$$\textcircled{3} \quad a \wedge b = b \wedge a$$

$$a \wedge 0 = 0$$

$a \wedge 0 = \text{Same thing}$

$$a \wedge a = 0$$

$a \wedge \text{Something} = 0$

$$\boxed{a \wedge b = c \Leftrightarrow a \wedge c = b}$$
$$\Leftrightarrow b \wedge c = a$$

$5 \rightarrow g \rightarrow 1 \rightarrow 3 \rightarrow 7$
 0101 1001
 $\begin{matrix} 2^3 & 2^2 & 2^1 & 2^0 \\ 2 & 2 & 2 & 2 \end{matrix}$

Man XOR Pad
 0001 0011 0111
 $\begin{matrix} 2^3 & 2^2 & 2^1 & 2^0 \\ 2 & 2 & 2 & 2 \end{matrix}$

$$5 \wedge 5 = 0$$

$$5 \wedge g = 1100 = 12$$

$= g \wedge 5$

$$5 \wedge f = 0100 = 4$$

$= f \wedge 5$

$$5 \wedge 3 = 0110 = 6$$

$= 3 \wedge 5$

$$5 \wedge 7 = 0010 = 2$$

$= 7 \wedge 5$

$$g \wedge g = 0$$

$$g \wedge 1 = 1000 = 8$$

$= 1 \wedge g$

$$g \wedge 3 = 1010 = 10$$

$= 3 \wedge g$

$$g \wedge 7 = 1110 = 14$$

$= 7 \wedge g$

$$1 \wedge 1 = 0$$

$$1 \wedge 3 = 0010 = 2$$

$$1 \wedge 7 = 0110 = 6$$

$$3 \wedge 3 = 0$$

$$3 \wedge 7 = 0100 = 4$$

$= 7 \wedge 3$

$$7 \wedge 7 = 0$$

5, 9 → 1, 3 → 7
 0101
 $\begin{smallmatrix} 2^3 & 2^2 & 2^1 & 2^0 \end{smallmatrix}$

$$\text{SAG} = 12 \\ g \wedge 7 = 14$$

0 → left

1 → right

Node {

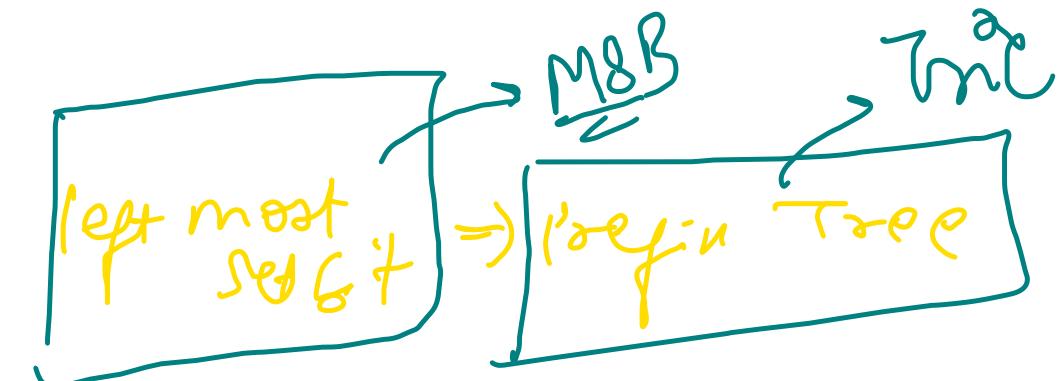
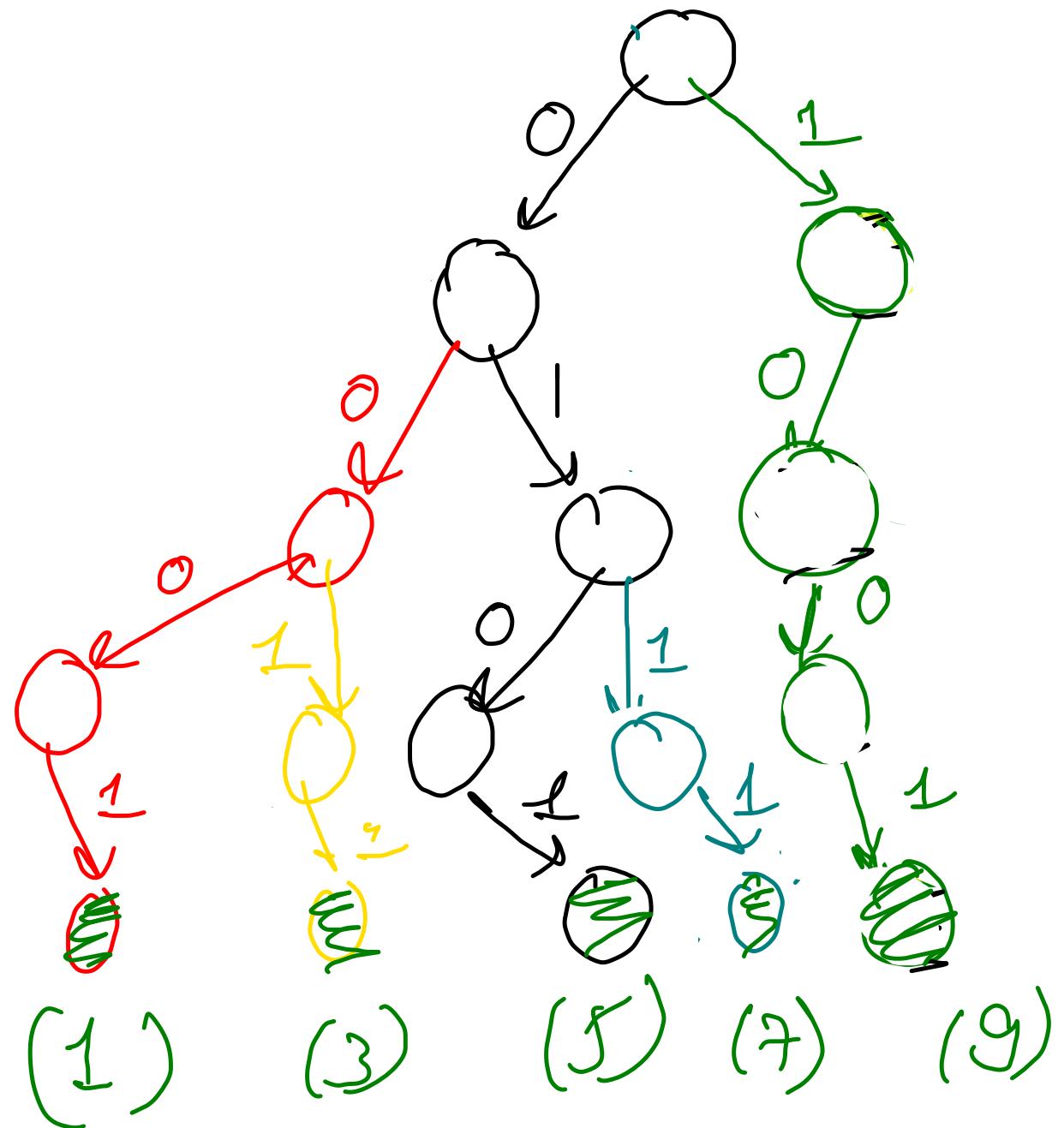
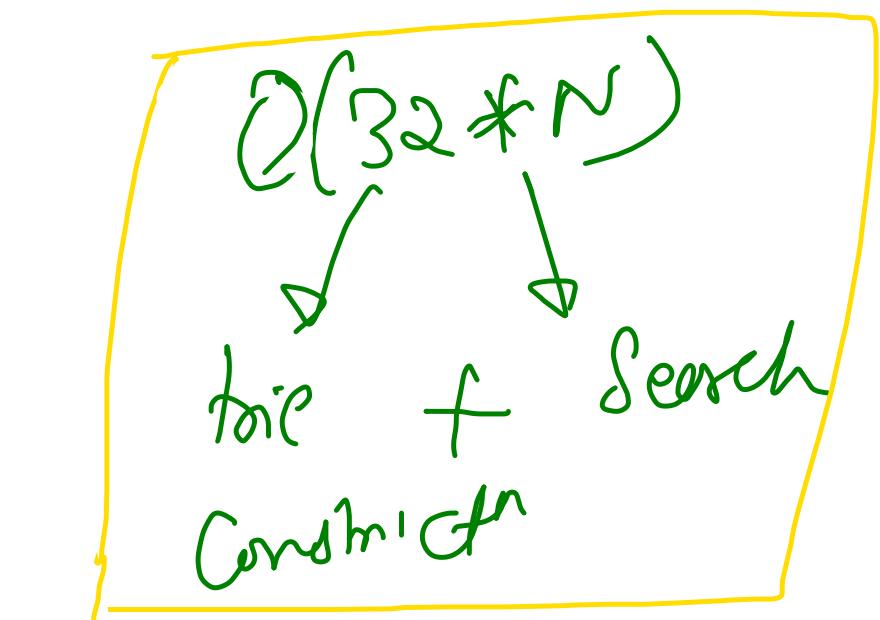
Node left

Node right

rg

g → 1 → 3 → 7
 1001
 0001
 0011
 0111
 $\begin{smallmatrix} 2^3 & 2^2 & 2^1 & 2^0 \end{smallmatrix}$

$$1 \wedge 9 = 8 \\ 3 \wedge 9 = 10 \\ 7 \wedge 9 = 14$$



```

public static 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 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;
}

```

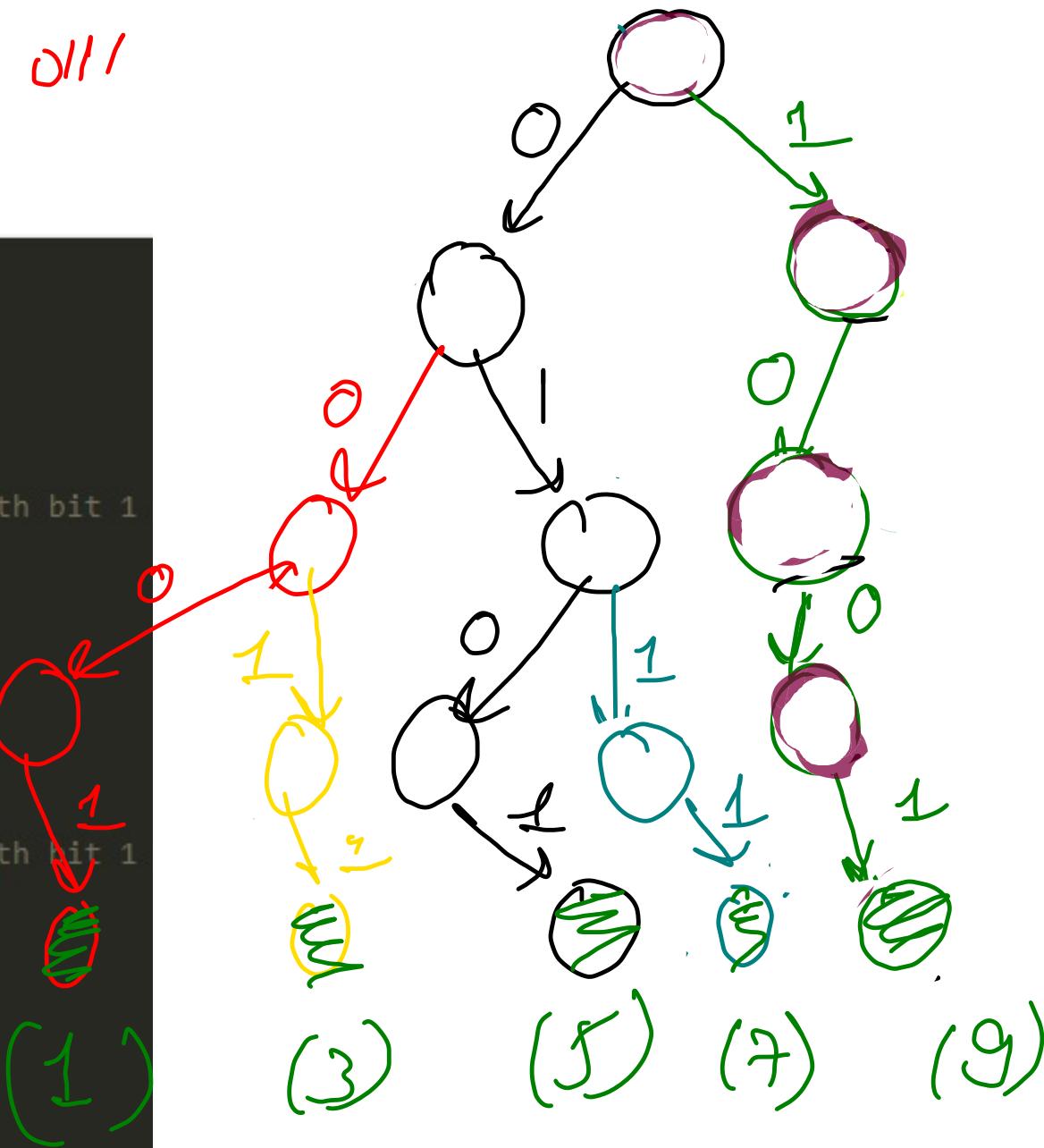
$\begin{matrix} 5 & 9 & 1 & 3 & 7 \\ \rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\ 0/01 & 1/001 & 0001 & 0011 & 0111 \end{matrix}$

```

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 -> XOR ith bit 1
            if(root.right != null){
                root = root.right;
                maxXOR = maxXOR | (1 << i);
            } else {
                root = root.left;
            }
        } else {
            // pair exists with ith bit 0 -> XOR ith bit 1
            if(root.left != null){
                root = root.left;
                maxXOR = maxXOR | (1 << i);
            } else {
                root = root.right;
            }
        }
    }
    return maxXOR;
}

```



~~HW~~ Unique Rows in Boolean Matrix

✓ 1 1 0 1

1101

1 0 0 1

✓ 1 0 0 1

0 1 1 1
1 0 0 0

✓ 1 1 0 1

✓ 1 0 0 1

No of unique rows =
no of nodes in the
deepest level of
tree

✓ 1 0 0 0

✓ 0 1 1 1

1707

Max XOR Pair - 11

nums [5, 9, 1, 3, 7]

0101 1001 0001 0011 0111

search (trie, 7)

search (trie, 8)

search (trie, 4)

search (trie, 6)

= 1A6

we already
all queries

offline
queries

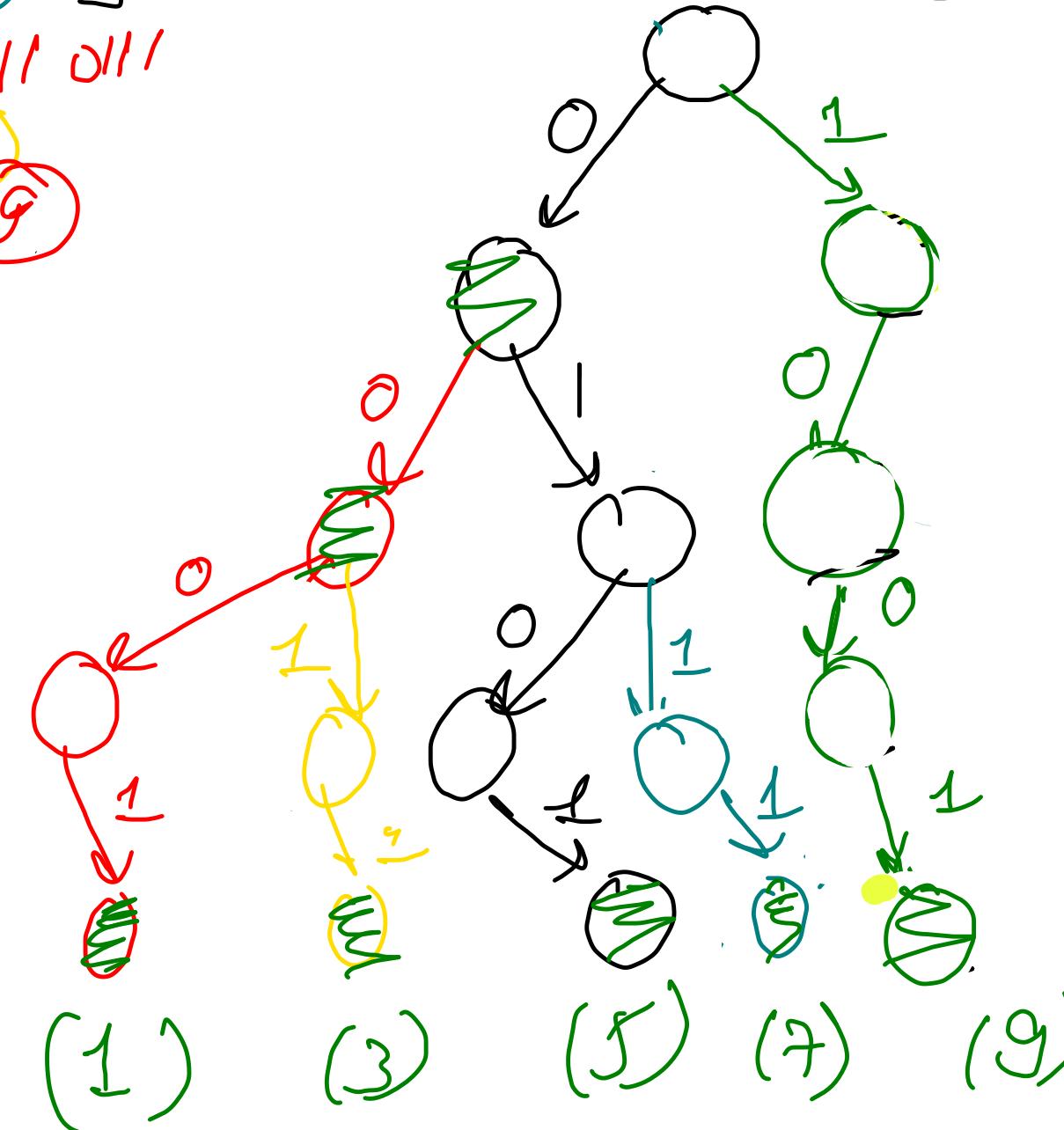
queries

{ 6, 7 } ③

{ 4, 5 } ②

{ 8, 9 } ④

{ 7, 3 } ①



①

$\{7, 3\}$
3rd

②

$\{4, 5\}$
8th

③

$\{6, 2\}$
6th

④

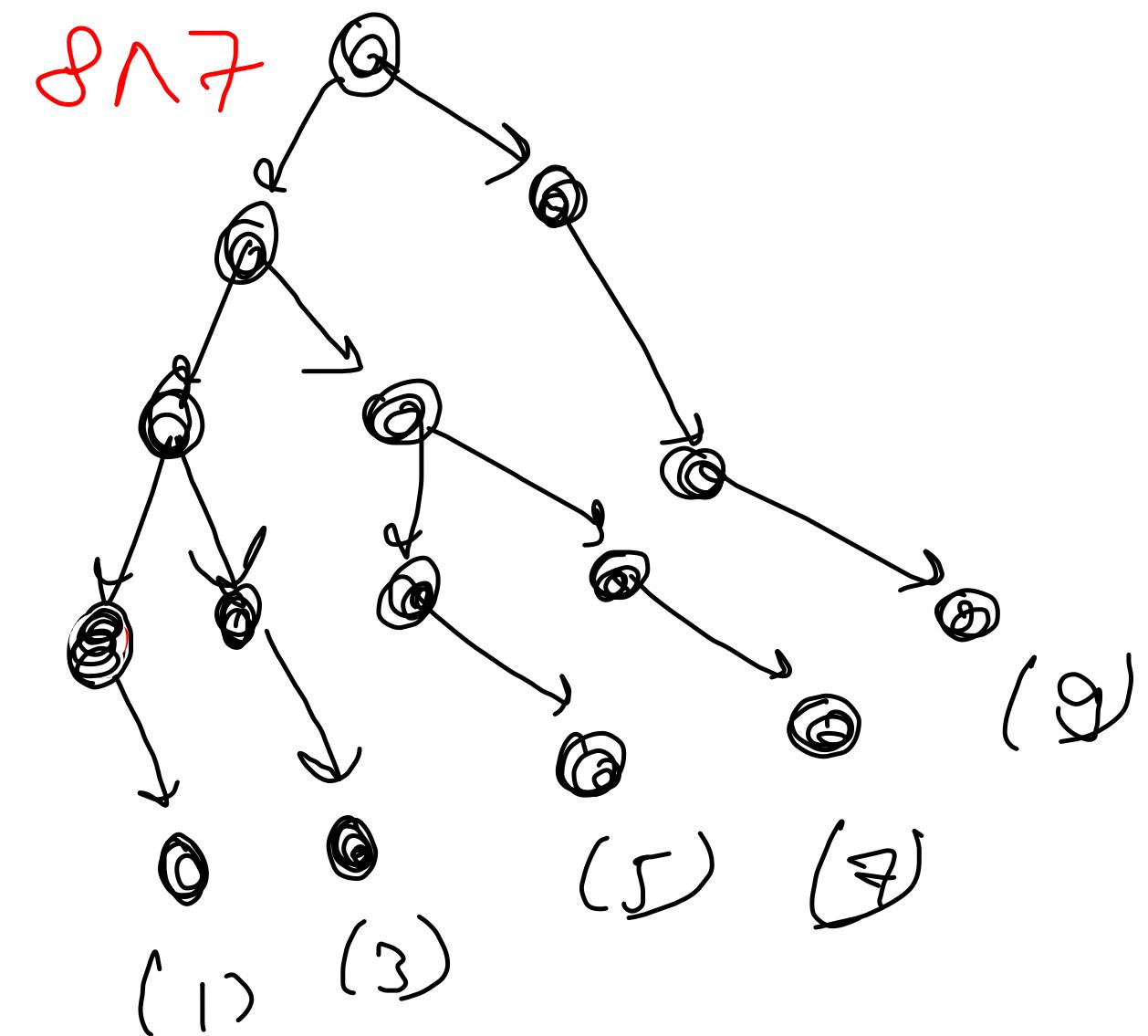
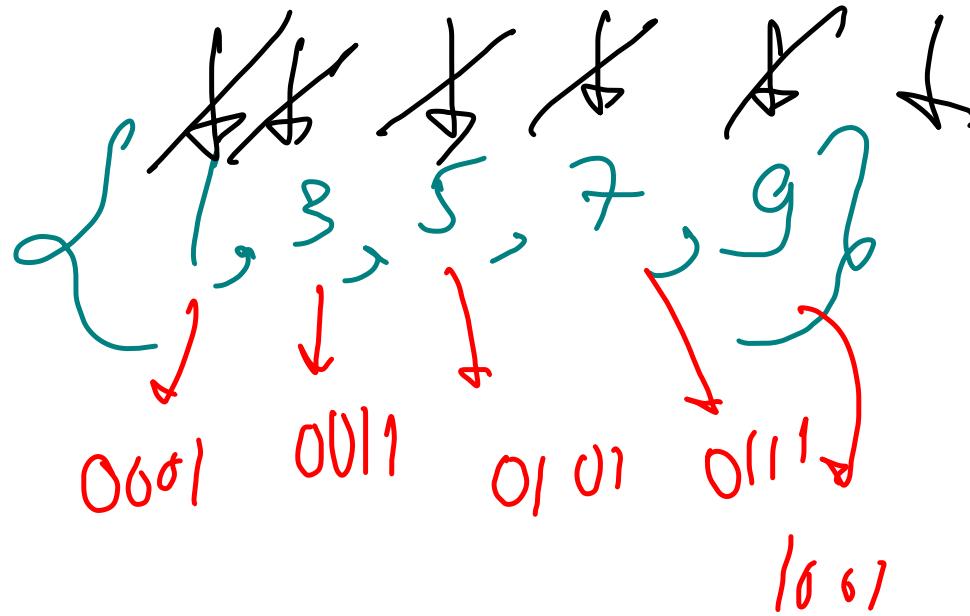
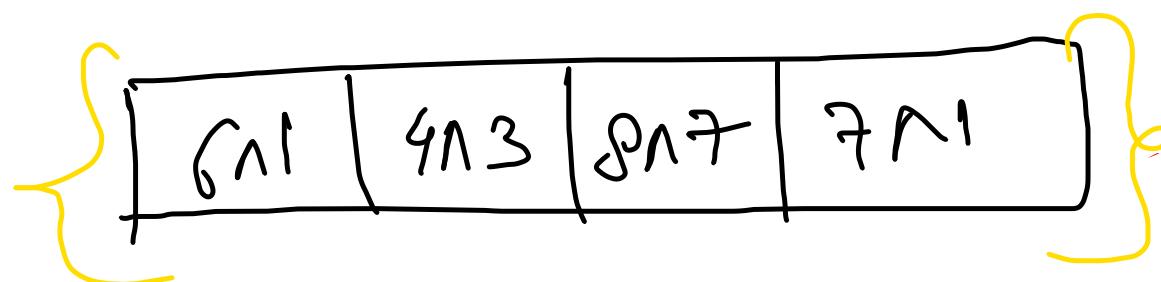
$\{8, 9\}$
2nd

7Λ1

4Λ3

6Λ1

8Λ7



```

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;
}

```

offline quer'g

```

public static class Node{
    Node left = null;
    Node right = null;
}

public void insert(Node curr, int val){ }

public int search(Node curr, int val){ }

```