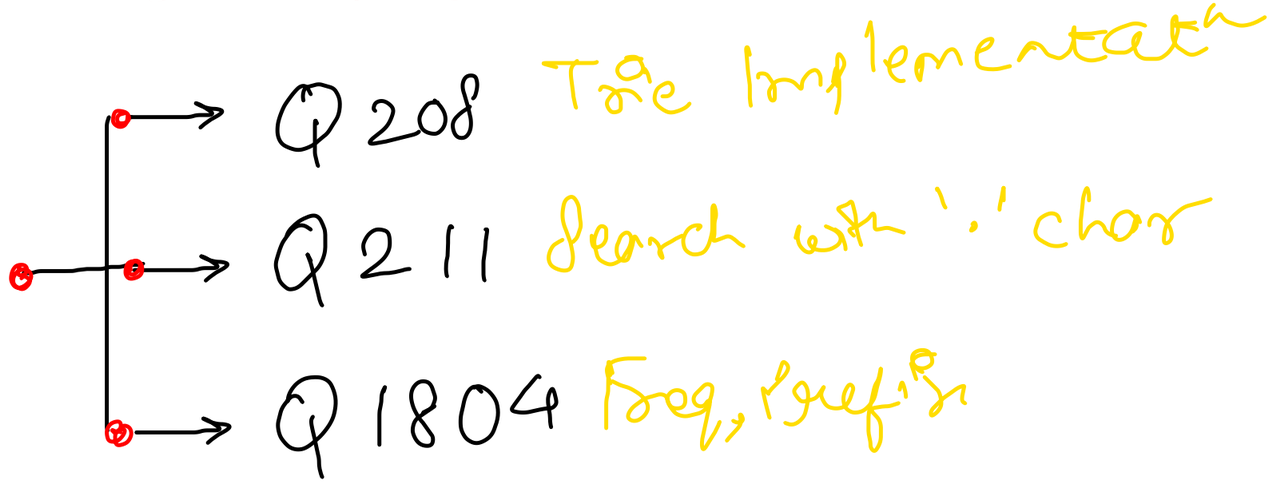
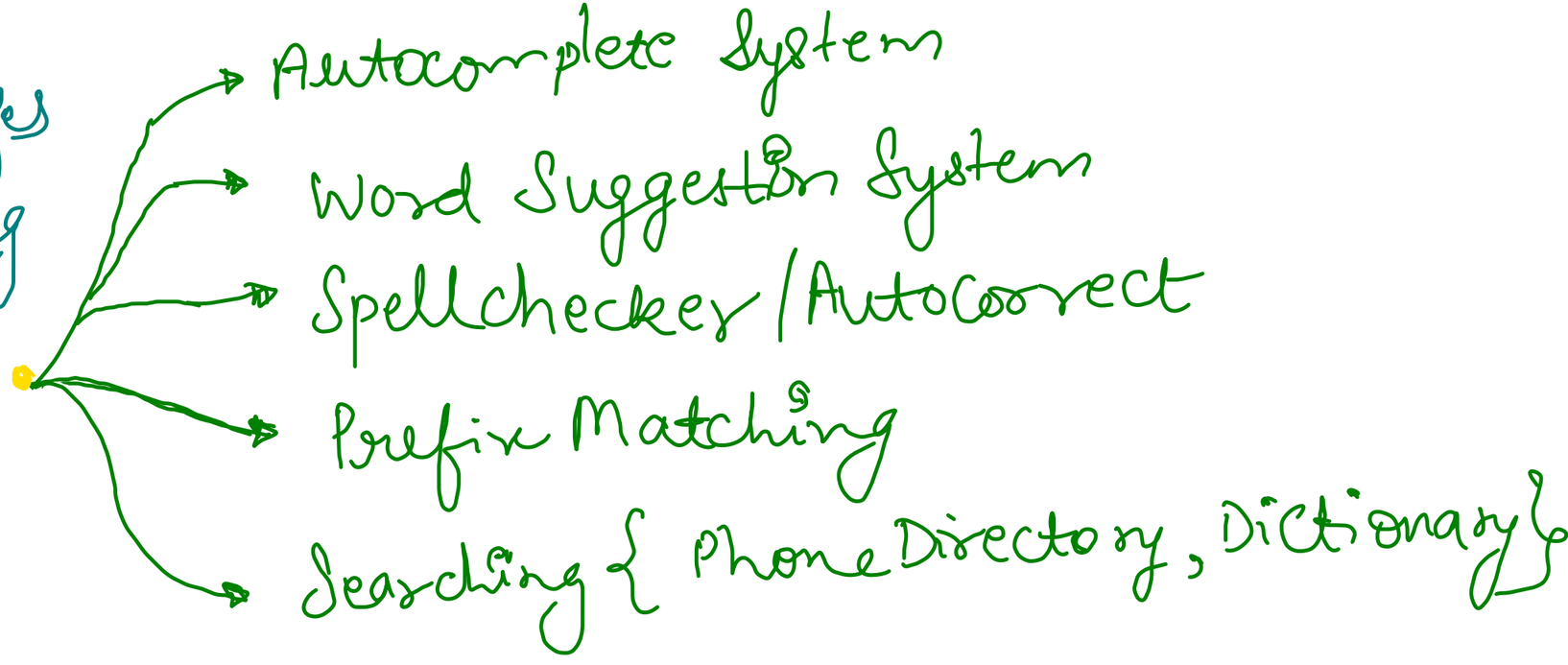
TRIE IMPLEMENTATION



Advantages & Disadvantages of Trie over Hashing

Applications of Tries



Dictionary Problems

- Q720 longest Word in Dictionary
- Q677 weighted Prefix Search
- Q648 Replace Word with Prefix
- Q14 Longest Common Prefix
- Q676 Magic Dictionary
- IB Shortest Unique Prefix
- GFG (Q1698) Count Distinct Substrings
- HW GFG Most Frequent Word

- Q1268 Search Suggestion Systems (SD)
- 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

GFG Subarrays with XOR < K

GFG Unique Rows in Boolean Matrix

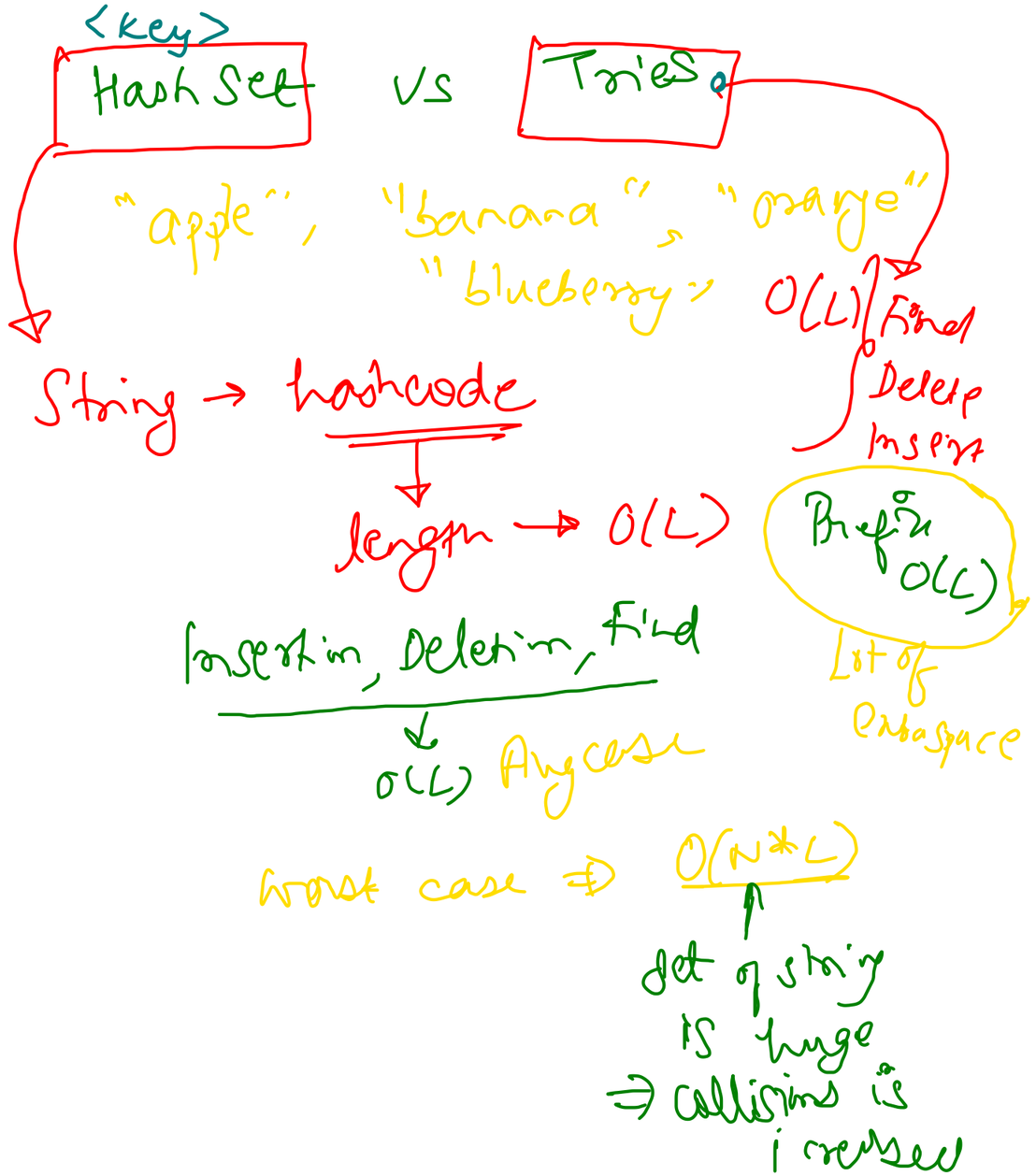
Bit Manipulation Basics

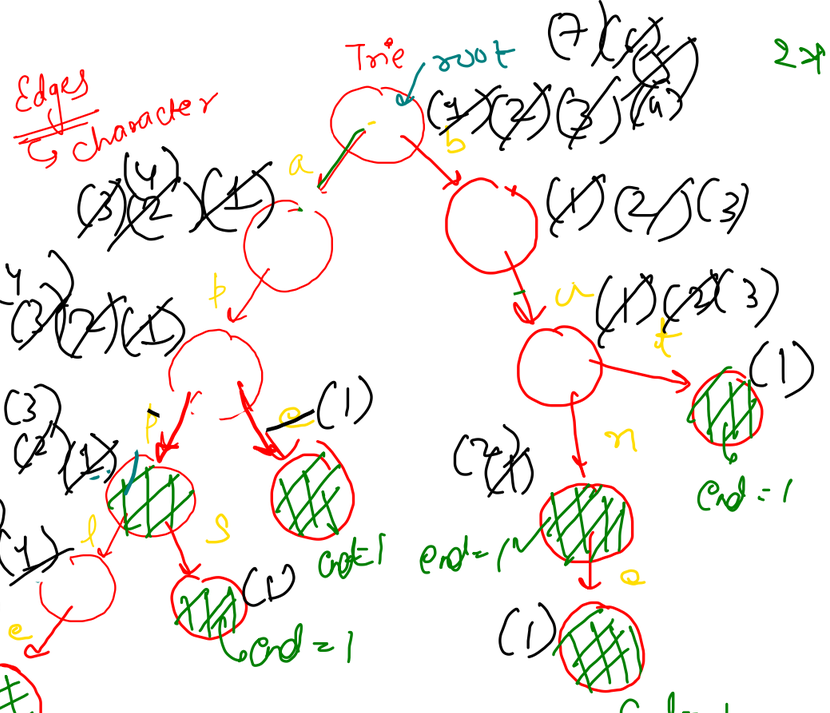
- Set, get, Unset
- left shift, rightshift
- XOR Properties

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.



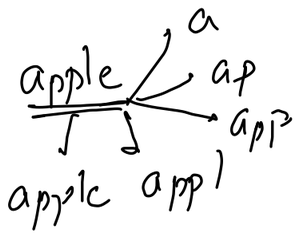


2 → insert("apple")
insert("bane")
insert("bad")
insert("ban")
insert("app")
insert("apps")

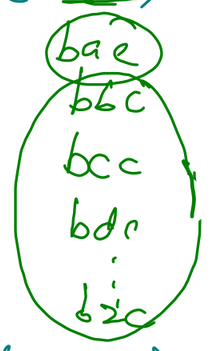
search("apple") True
search("bank") False
search("ap") False

prefix("apple") True
prefix("bank") False
prefix("ap") True

iSeek("ape")



search("b.c") F



search("b.t")

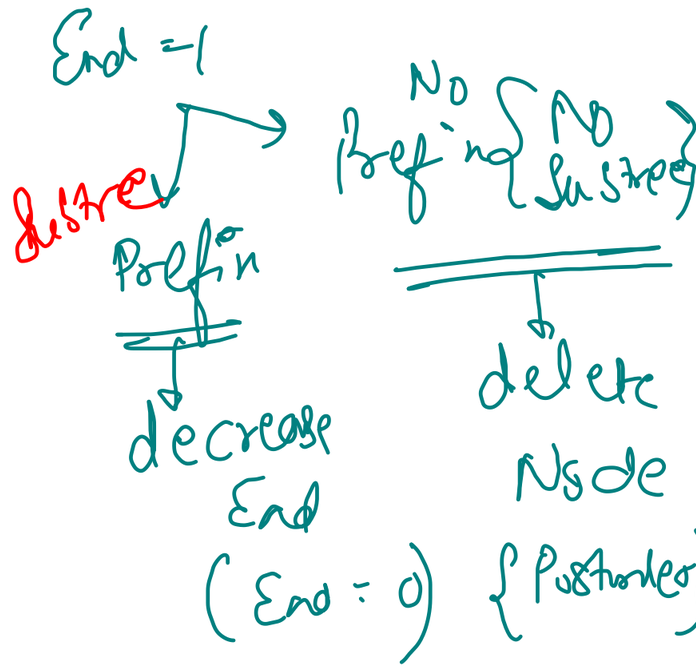
bat, bbt, bct, ... bzt

printprefix("ap") DFS: Broader

{ ape, app, apple, apples, apps }

delete("ban")

End > 2 : decrease End



```java
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;
    }
}
```

```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();
}

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

Take →

For 1 String { Insert, Delete, Find, Prefix } → O(L)

Space for entire Trie → O(NAL)

Q211 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

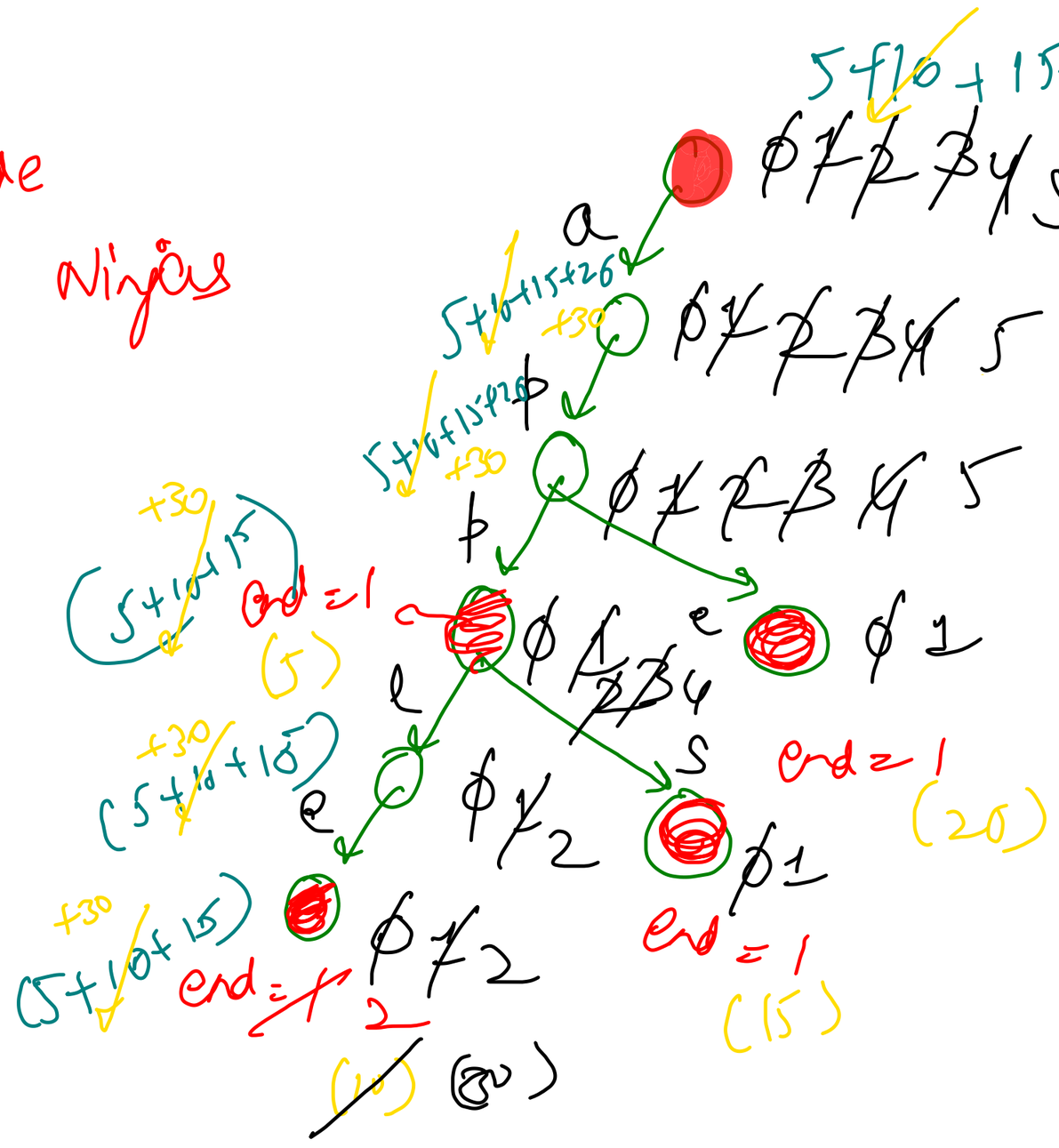
$(26)^h$

↳ All are dots (.)

Q1804

Leetcode

→ Coding Ninjas



freq("apple") → 2

freq("apples") → 0

freq("app") → 1

pref("apple") → 2

pref("app") → 4

```java
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--;
    }
}
```

```java
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();
}
```

O(L)

```java
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();
}

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();
}
```

O(L)

O(L)

```java
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();
}
```

(Q677) Map Sum Pairs

```java
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);
    }
}
```

```java
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();
}
```

O(L)

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

    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);
}
```

O(L)

O(L)

Weighted Prefix Search

```java
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;
}
```

O(L)

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`.

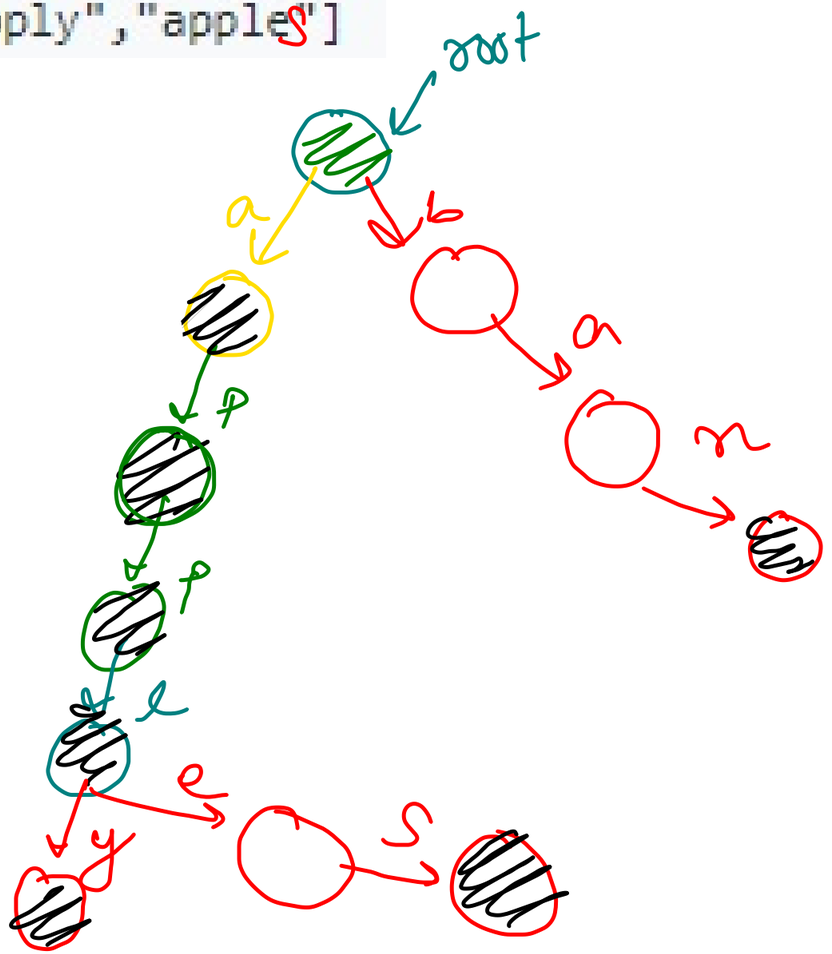
If there is more than one possible answer, return the longest word with the smallest lexicographical order. If there is no answer, return the empty string.

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

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

⇓

Deepest node



longest word = "" ⓪

= "a" ①

= "ap" ②

= "app" ③

= "appl" ④

= "apply" ⑤

720 longest word in dictionary

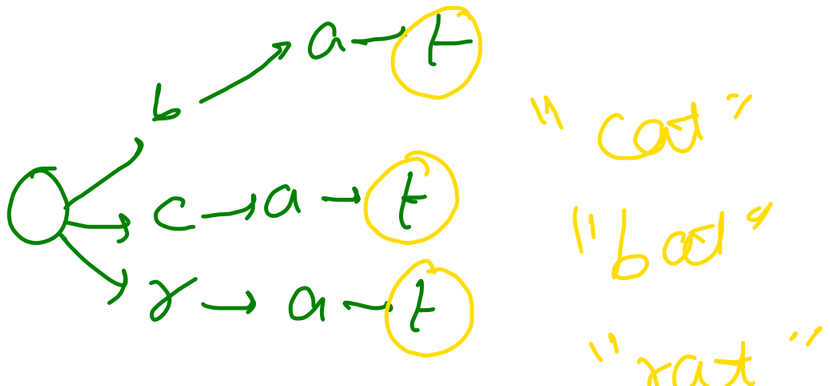
720 longest word in Dictionary

```
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.

Assumption

```
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;
}
```

longest Common Prefix = {lowest common Ancestor of shaded nodes}

"flower"

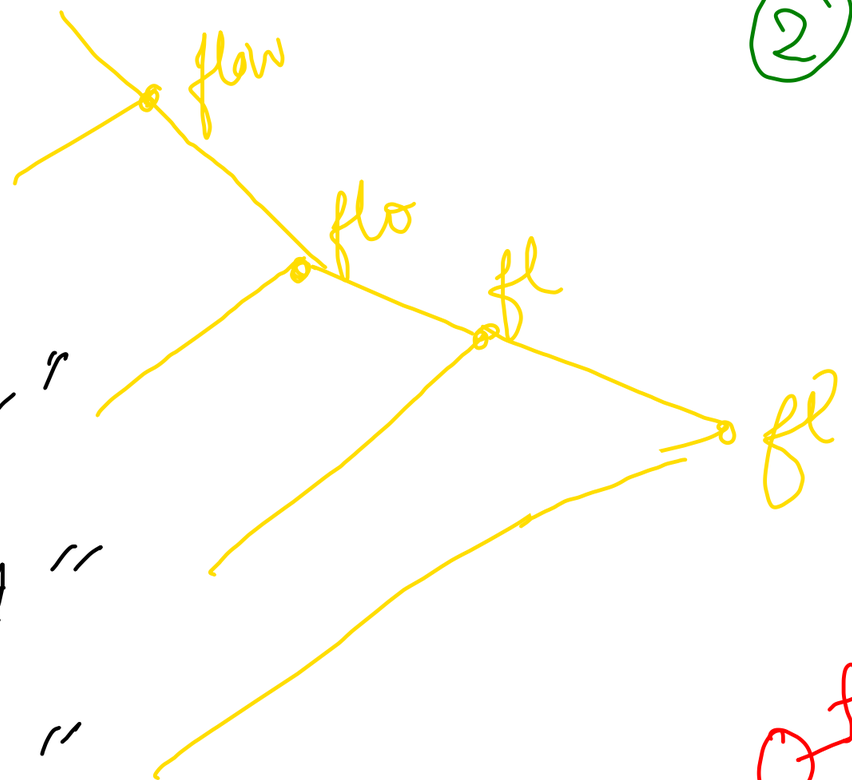
"flow"

"flow"

"flow"

"flight"

"flow"

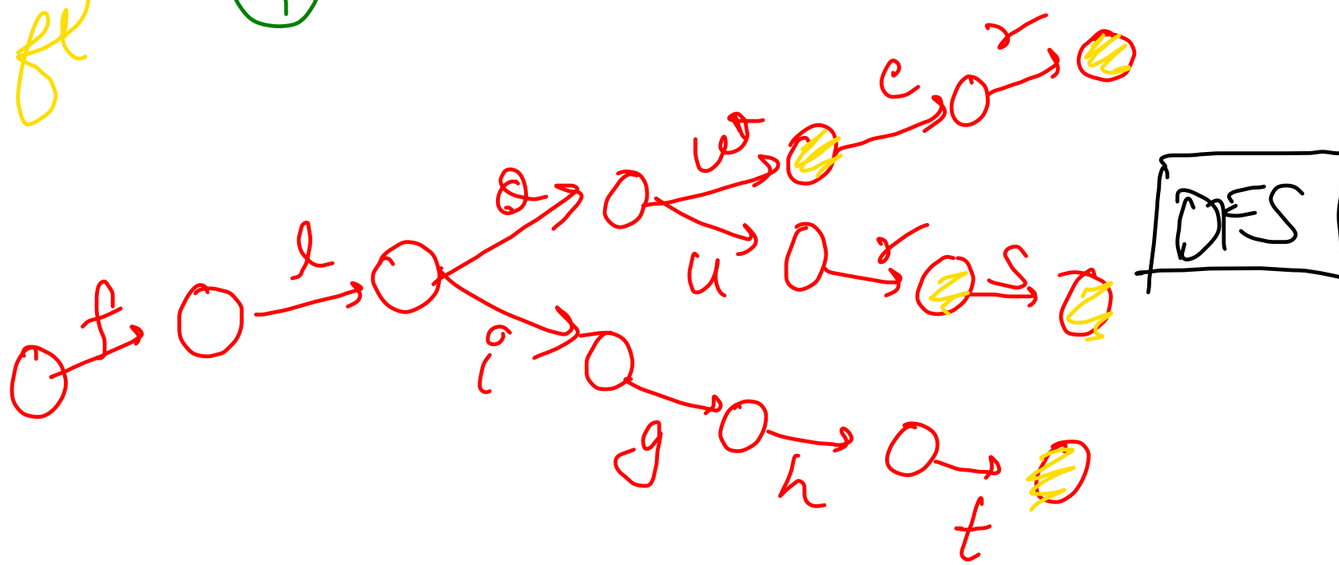


① Vertical scanning

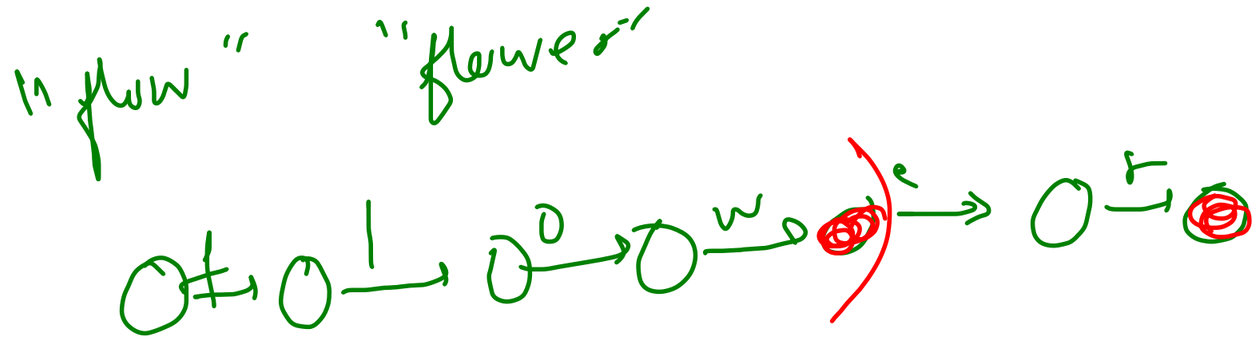
② Horizontal scanning

③ Divide & Conquer

④ Trie :→ Queries



```java
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);
}
```



```java
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]

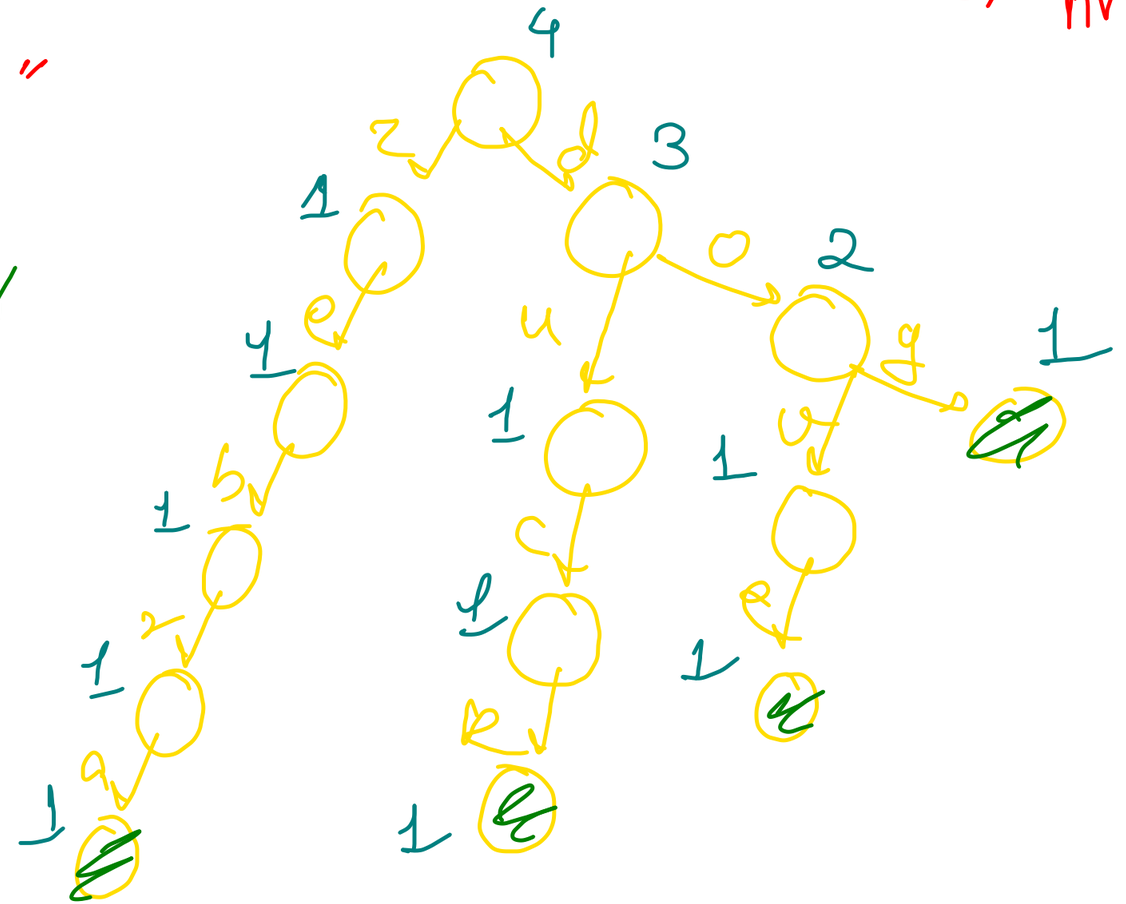
"z" "dog" "du" "dov"

Approach: Find first character which have prefix count = (1)

"dog" vs "dogs"

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

Answer will always exist



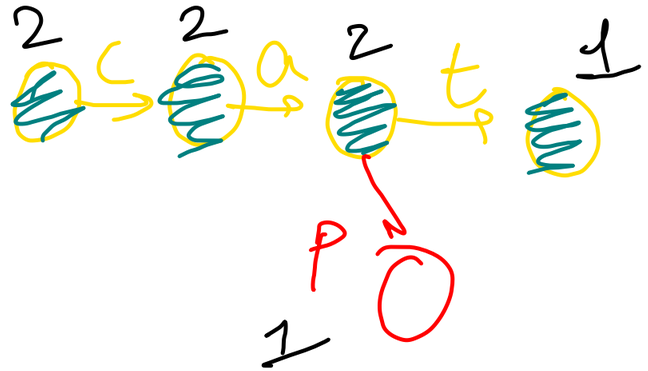
```java
public static class Node{
    Node[] children = new Node[26];
    int prefCount = 0;
    int end = 0;

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

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

```java
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++;
}
```

```java
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;
}
```

```java
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 Prefix



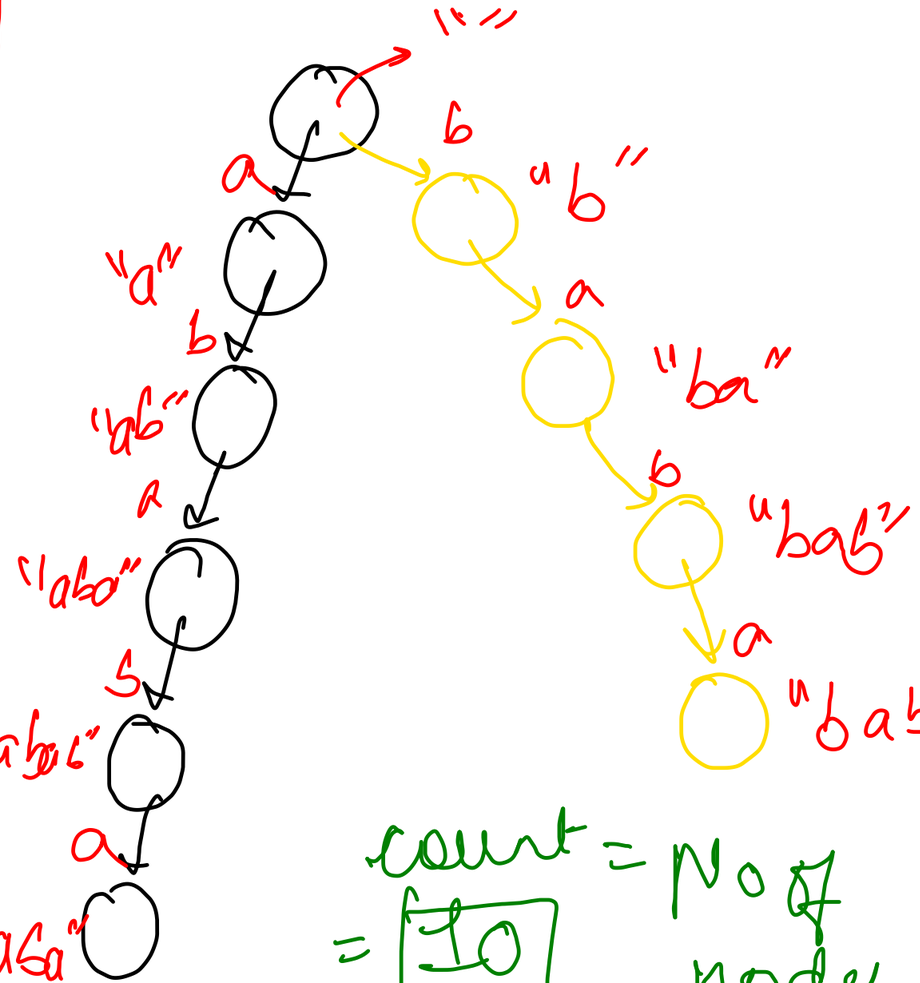
Count Distinct substrings

"ababa"

a
ab
aba
abab
ababa

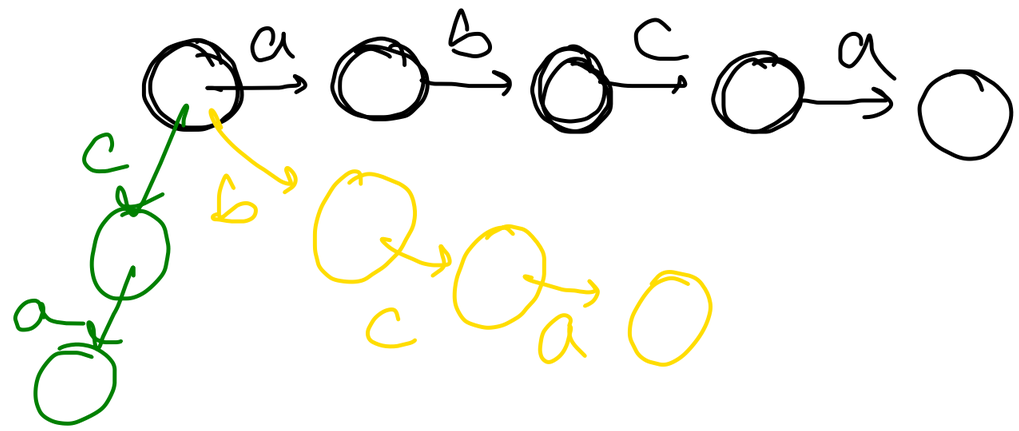
b
ba
bab
baba
""

$O(N^2)$



count = No of node in the tree

= 10



$O(N^2)$

```java
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();
    }
}
```

```java
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 ⟹ getEnd()

② Word Prefix ⟹ root != null

```
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 void exist(int r, int c,  char[][] board, String ssof, Node root){
    if(root == null) return;  // if word is prefix of atleast 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 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

- Set, get, unset
- left shift, right shift
- XOR Properties

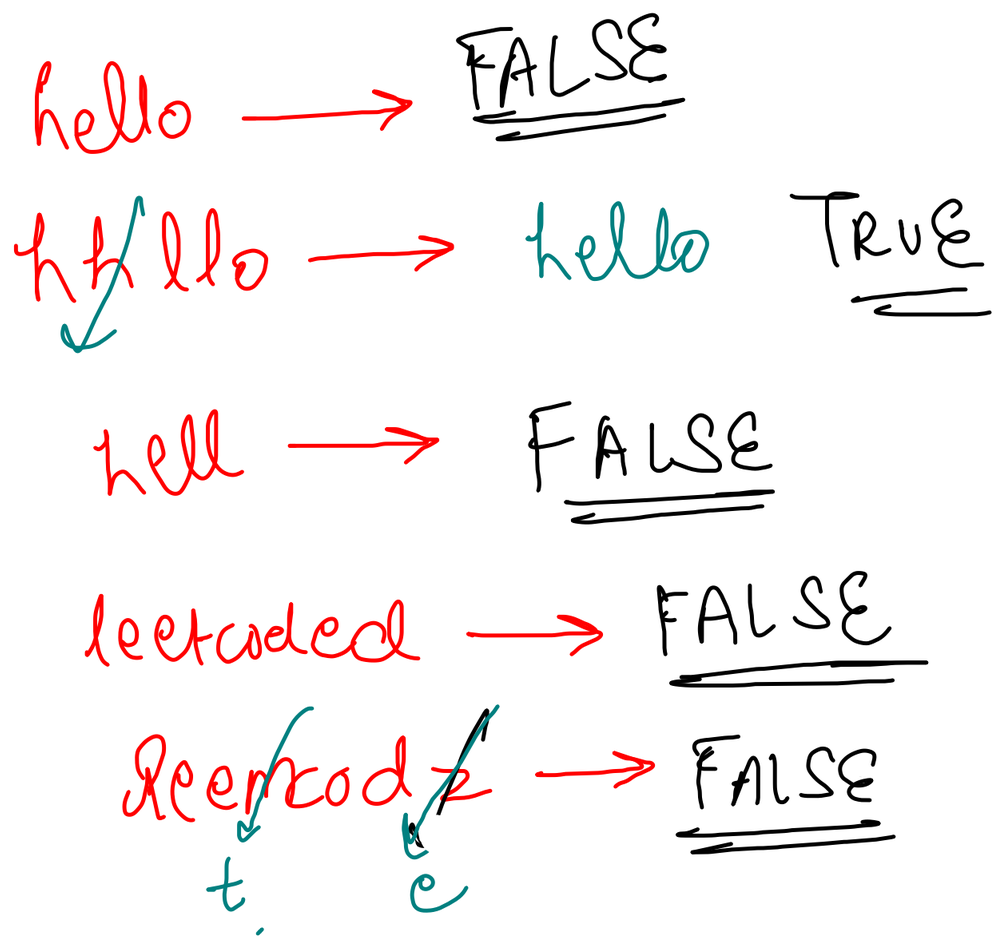
Magic Dictionary

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.

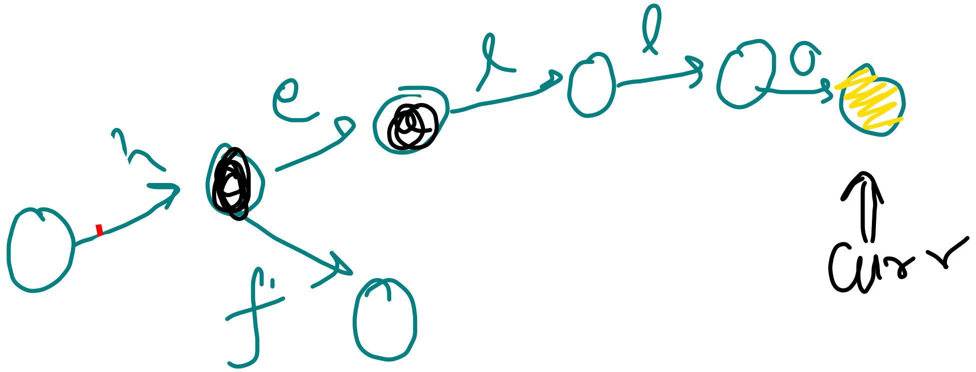
Implement the `MagicDictionary` class:

- `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]
```



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



hhllo
01234
0, false

→

hhllo (e)
01234
1, false

→

hello
01234
2, true

→

hello
01234
3, true

→

hello
01234
4, true

↓

hello
01234
5, true

base case: idx == str.length

↙ change == true ↘ curr.eow == true

for all char a to z
if char → replace { curr → contains(ch)

if change == true
return false

else { idx+1, curr.get(ch), true }

"hello" 0,f → 1,f → 2,f → 3,f → 4,f → 5,f ⇒ return false

"hello" (l l) 0,f → 1,f → 2,f → 3,t → return false

hell — false
end of word == false

```java
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);
}
```

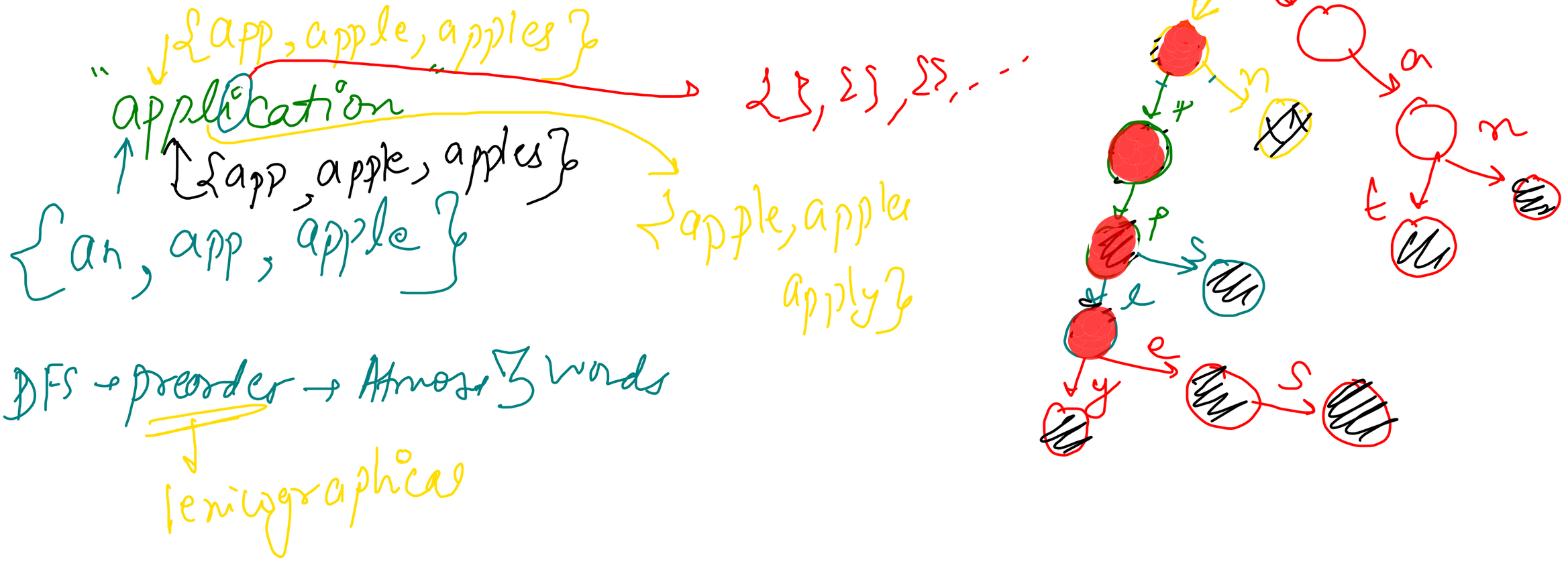
Search Suggestion System { System Design }

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

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.*



```java
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);
        }
    }
}
```

```java
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

"elppa" "elam"

- insert (R to L)
- search (R to L)

Online ⇒ stream of characters

a, d, l, c, m, a, l, e

false false false false false false false true

"elamclda"

suffix of online characters in dictionary → Y → true; N → false

Palindrome pairs {O(N²)}

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.

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

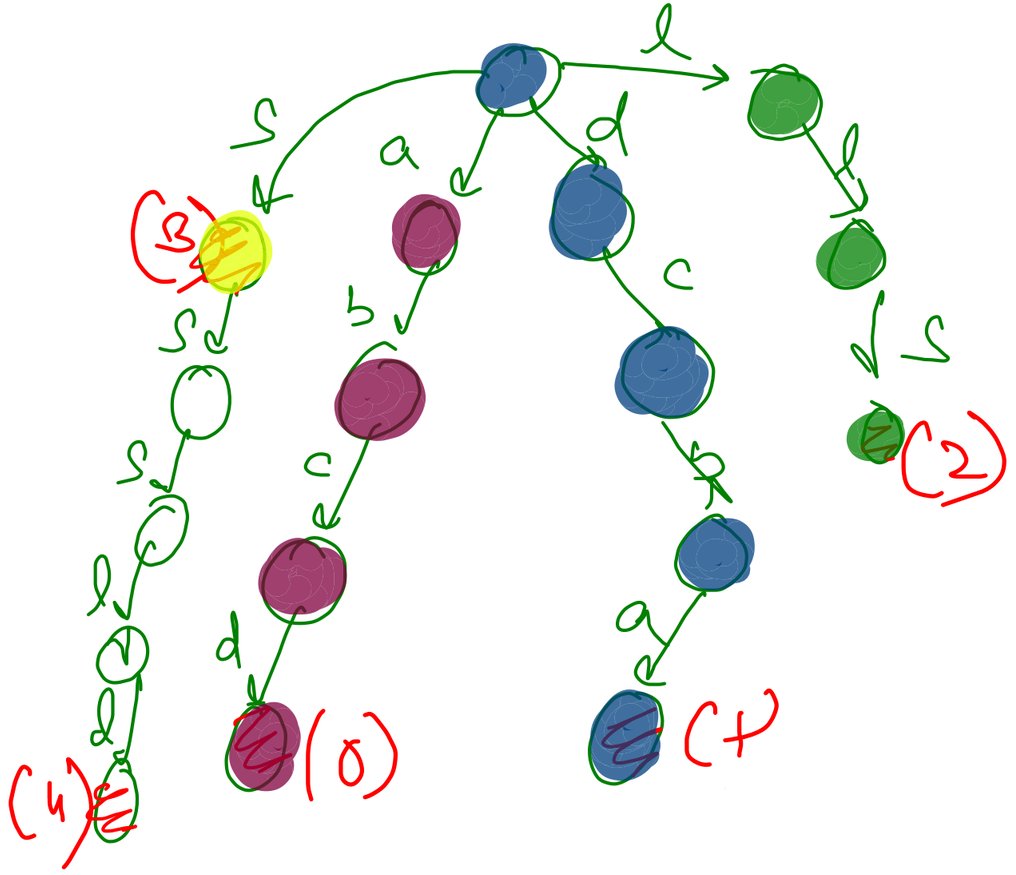
0 + 1
abcd + dcba
abcddcba

1 + 0
dcba + abcd
dcbaabcd

2 + 4
lls + sssll
dlsssslls

3 + 2
s + lls
slls

x + y ⇒ palindrome
- x is reverse of y
- Suffix of y = reverse of x && remaining y should also be a palindrome
- prefix of x = reverse of y && remaining x should be a palindrome



XOR Problems

Bit Manipulation Basics

- Set, get, Unset
- left shift, rightshift
- XOR Properties

Q421 Maximum XOR Pair - I

Q1707 Maximum XOR Pair - II

Q1803 XOR Pairs in Range

HW GFG Subarrays with XOR < K

GFG Unique Rows in Boolean Matrix

Bit Manipulation Basic

Integer { 4 byte → 32 bits }

9 → 7 → 5 → 3 → 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 ⇒ 0001 ⇒ x << 2 = $x * 2^2$

0001 ⇒ 0001 << 2 = 0100

0001 ⇒ 0001 << 3 = 1000

2^0 2^3

$x << 3 = x * 2^3$

right shift ⇒ 5 >> 1 = $5/2^1 = 2$

0101 >> 1 = 0010

(5) (2)

0101 >> 2 = 0001

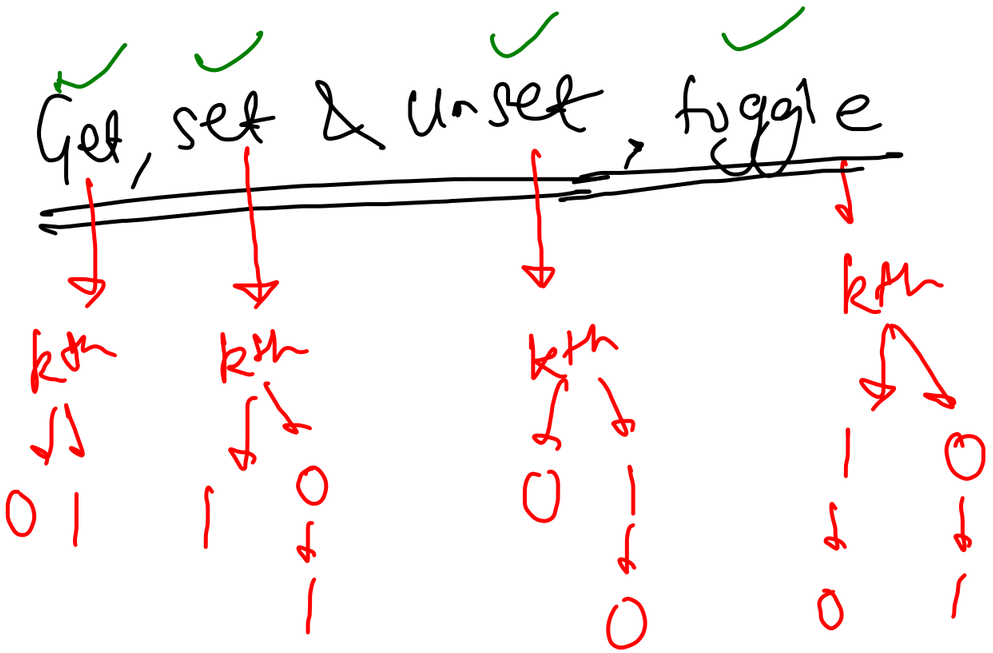
(1)

5 >> 2 $= 5/2^2 = 5/4 = 1$

left shift ⟶ $x << k = x * 2^k$

right shift ⟶ $x >> k = x / 2^k$

Get, set & unset, toggle



1 & 1 = 1

1 & 0 = 0

0 & 1 = 0

0 & 0 = 0

Get $n0(x) = 5$

(Right to Left)
(0-based)

0 1 0 1
3rd 2nd 1st 0th

$k = 2^{nd}$ bit

0 1 0 1

$0001 << 2 = 0100$

AND 0100

$x \& (1 << k)$

$!= 0$

$\Rightarrow$ Set (1)

$k = 3^{rd}$ bit

0 1 0 1

$0001 << 3 = 1000$

AND 0000

$x \& (1 << k)$

$== 0$

$\Rightarrow$ unset (0)

Set

$x = 5$

0 1 0 1

$k = 2^{nd}$ bit

$0001 << 2 = 0100$

OR 0 1 0 1

0 1 0 1

$k = 3^{rd}$ bit

$0001 << 3 = 1000$

OR 1 1 0 1

$1 | 1 = 1$

$1 | 0 = 1$

$0 | 1 = 1$

$0 | 0 = 0$

Set $\Rightarrow$ $x = x | (1 << k)$

bitwise OR

Unset

mask

$= \sim(1 << k)$

$= \sim(0100)$

$= 1011$

0101

2nd bit

1011

0001 (AND)

0101

3rd bit

0111

0101 (AND)

$= 0111$

mask $= \sim(1 << 3)$

tilde

negation

$= \sim(1000)$

unset $\Rightarrow$ $x \;\&\; (\sim(1 << k))$

toggle

0 1 0 1

$k = 2^2$

0 1 0 0

0 0 0 1

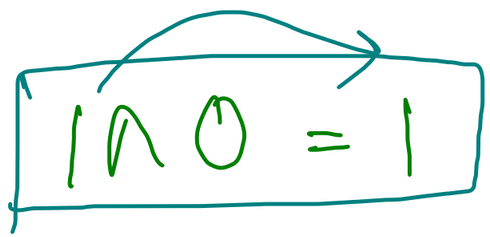
0 1 0 1

$k = 3^4$

1 0 0 0

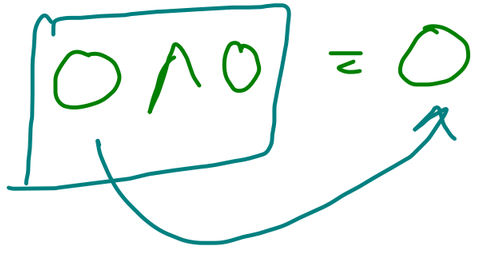
1 1 0 1

XOR



0 ∧ 1 = 1

1 ∧ 1 = 0



toggle → x ∧ (1 << mask)

XOR Properties

① $x \wedge 0 = x$

Anything $\wedge$ 0 = Same thing

② $x \wedge x = 0$

Something $\wedge$ Something = 0

③ $a \wedge b = b \wedge a$

$a \wedge b = c \iff a \wedge c = b$

$\iff b \wedge c = a$

5, 9 → 1, 3 → 7

0101 1001 0001 0011 0111

$2^3 2^2 2^1 2^0$ $2^3 2^2 2^1 2^0$

Man XOR Pair

$5 \wedge 5 = 0$

$5 \wedge 9 = 1100 = 12$ (= $9 \wedge 5$)

$5 \wedge 1 = 0100 = 4$ (= $1 \wedge 5$)

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

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

$9 \wedge 9 = 0$

$9 \wedge 1 = 1000 = 8$ (= $1 \wedge 9$)

$9 \wedge 3 = 1010 = 10$ (= $3 \wedge 9$)

$9 \wedge 7 = 1110 = 14$ (= $7 \wedge 9$)

$1 \wedge 1 = 0$

$1 \wedge 3 = 0010 = 2$

$1 \wedge 7 = 0110 = 6$

$3 \wedge 3 = 0$

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

$7 \wedge 7 = 0$

5 → 0101 (0 1 0 1)

9 → 1001 (1 0 0 1)

1 → 0001 (0 0 0 1)

3 → 0011 (0 0 1 1)

7 → 0111 (0 1 1 1)

5 & 9 = 12

9 & 7 = 14

1 & 9 = 8

3 & 9 = 10

7 & 9 = 14

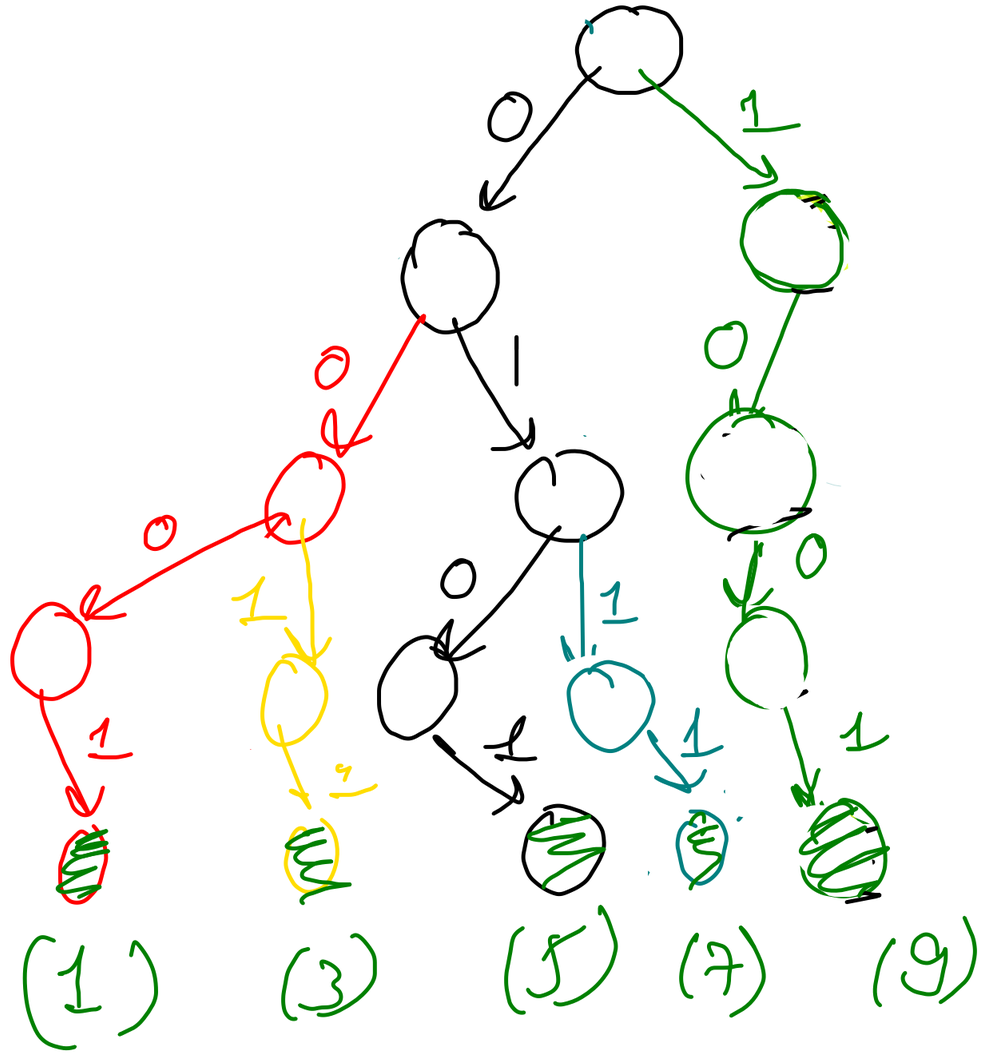
0 → left

1 → right

```
Node {
    Node left
    Node right
}
```

O(32 * N)

Trie + Search Construction



left most set bit ⇒ MSB ⇒ Trie

prefix Tree

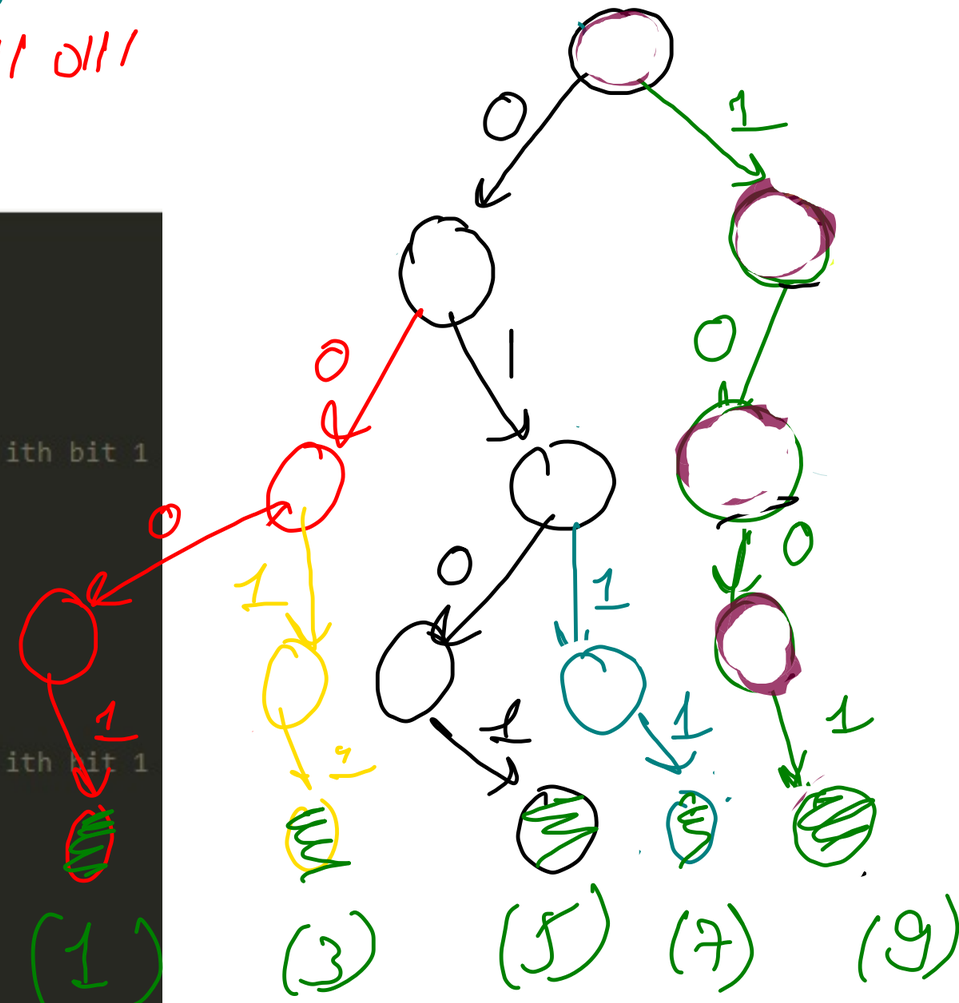
```java
public static class Node{
    Node left;
    Node right;
}

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;
        }
    }
}
```

```java
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;
}
```

5, 9, 1, 3, 7

0101 1001 0001 0011 0111

```java
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

1 0 0 1

1 1 0 1

1 0 0 0

0 1 1 1

1 0 0 0

0 1 1 1

1101

1001

0111

1000

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

Not Max XOR Pair -(

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

0101 1001 0001 0011 0111

search (trie, 7)

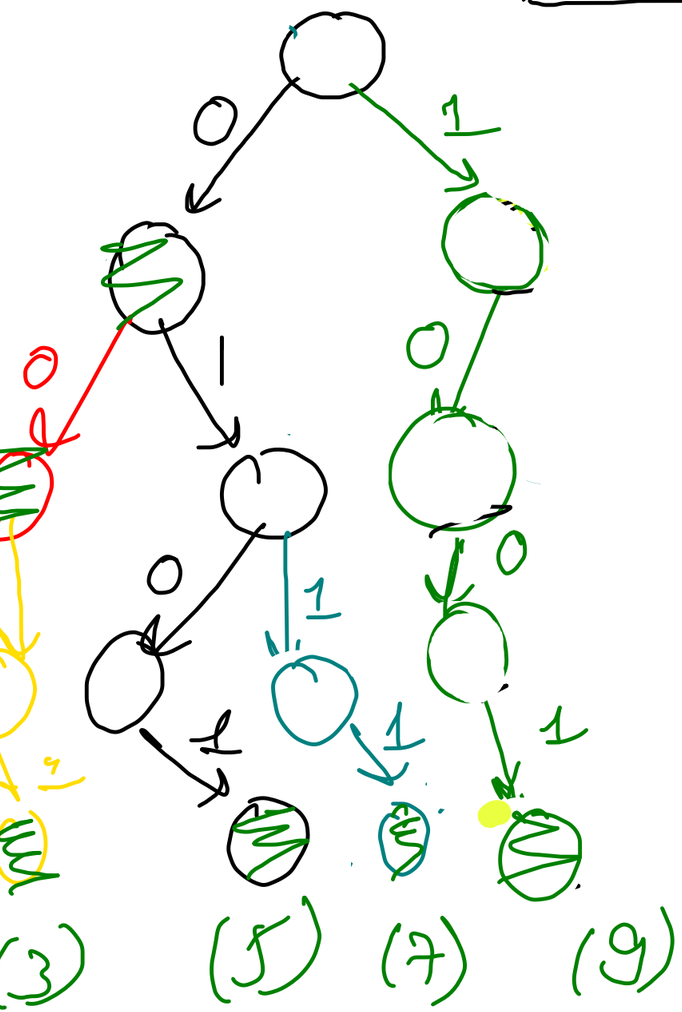
search (trie, 8)

search (trie, 4)

search (trie, 6)

= 1 ∧ 6

we already all queries → offline queries



queries

{6, 7} (3)

{4, 5} (2)

{8, 9} (4)

{7, 3} (1)

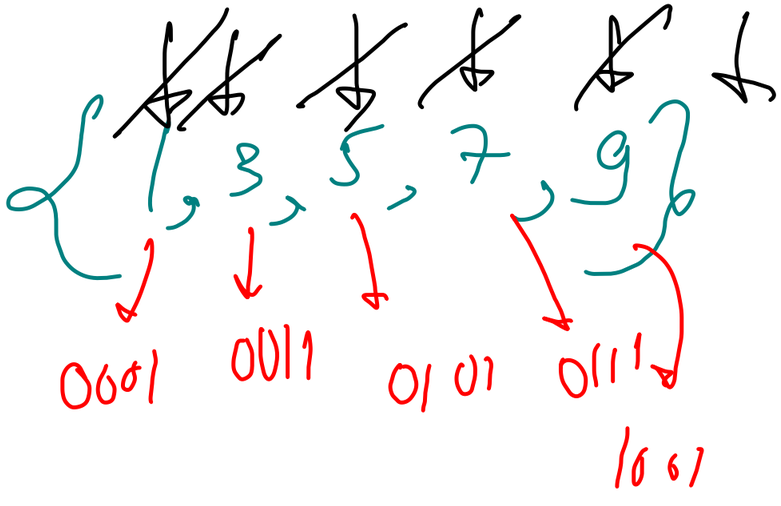
① {7,3} 3rd 7∧1

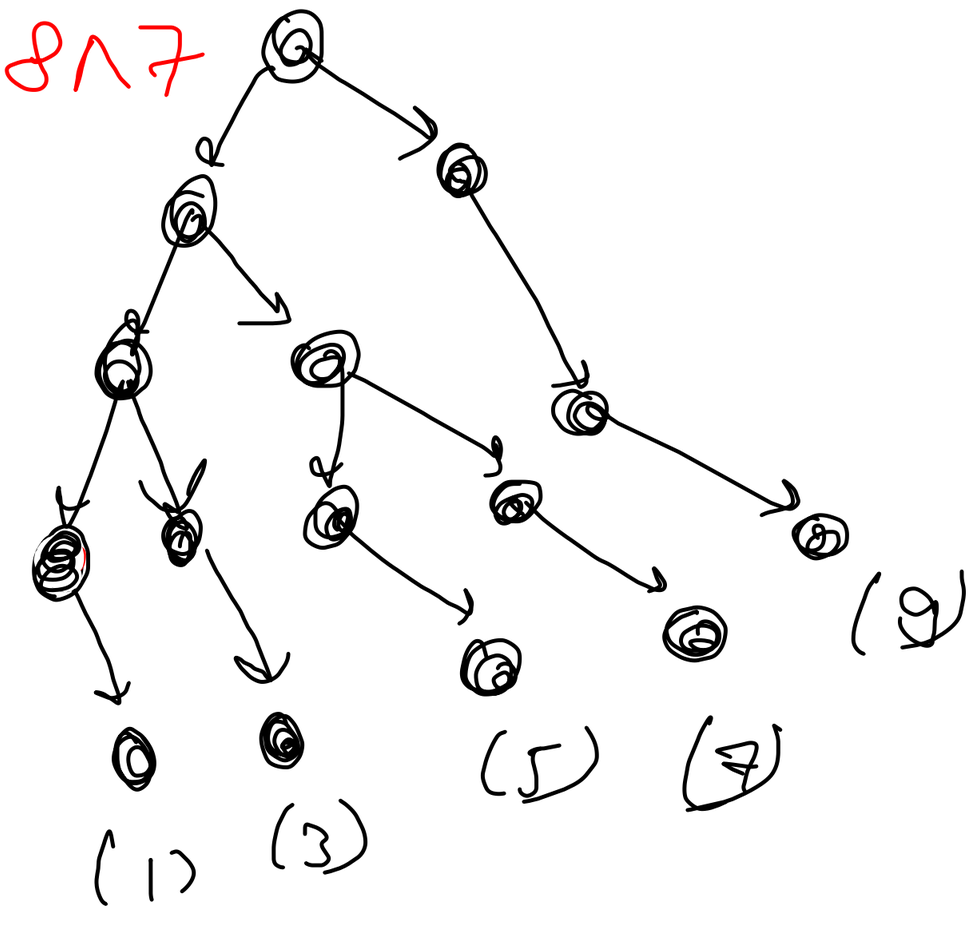
② {4,5} 1st 4∧3

③ {6,7} 0th 6∧1

④ {8,9} 2nd 8∧7

{ 6∧1 | 4∧3 | 8∧7 | 7∧1 }





```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;
}
```

offline queries

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

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

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

Prefix & Suffix Search

Dict. → "apple", "banana"

query("ap", "na") → False

query("ban", "ple") → false

query("ap", "el") → false

query("ap", "le") → true

query("appl", "pple")

Inspiration → Manacher's Algorithm

apple — Trie Insert suffix # prefix

"", apple → apple#appl e

a, pple → pple#appl e

ap, ple → ple#appl e

app, le → le#appl e

appl, e → e#appl e

apple, "" → #apple

Instead of prefix, we have to insert entire word

```
public static class Node{
    private Node[] children = new Node[27];
    int idx = -1;

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

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

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

```
Node root;
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++){
            // System.out.println(word.substring(i) + "#" + word
            insert(root, word.substring(i) + "#" + word, idx);
        }

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

overlapping queries

eg { "apple", "pple"

```
public void insert(Node root, String word, int idx) {
    Node curr = root;
    root.idx = 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.idx = idx;
    }
}
```

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

1803 Count Pairs with XOR in Range [5,14] Insert + Search

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

1 4 2 7
0001 0100 0010 0111

low ⇒ {2, 5}

(≤5) − (≤1)
0101 0001