Trie: (Prefix Tree).

Array
Linked List
Stack & Queue
Heap
Graph
} > Hashmap

⊙ 989994 → A
989432 → B
984321 → C
972432 → D

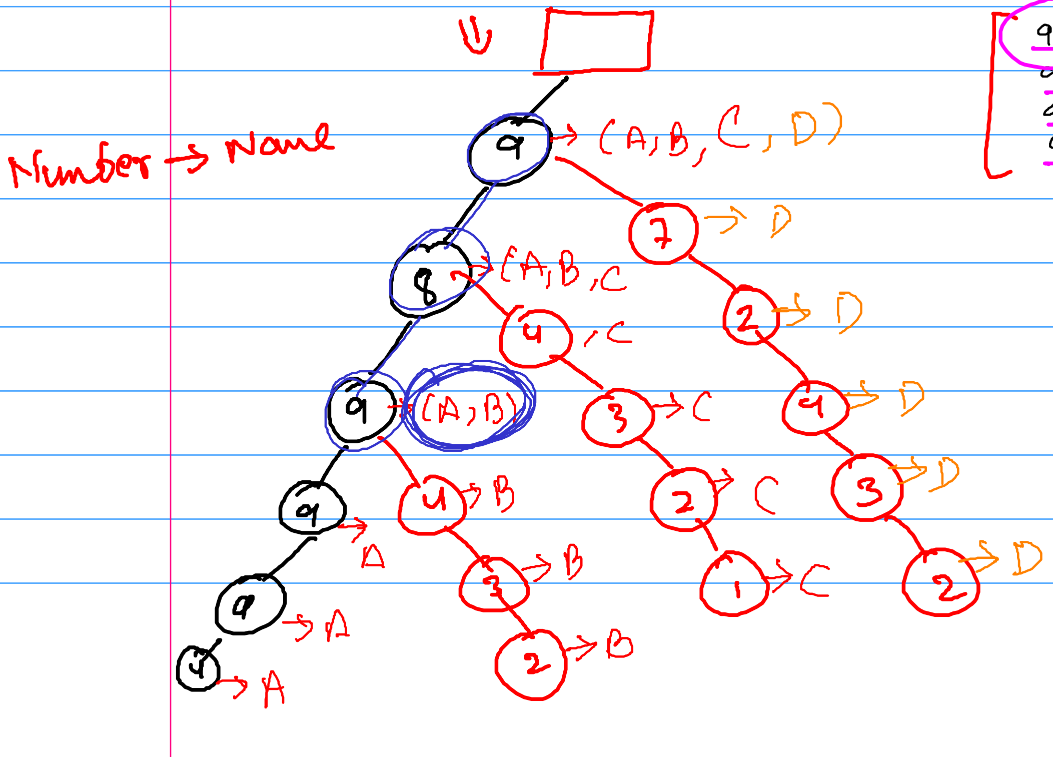
"984432"

< int, string >

⇒ Hash

[98994] → A
[989432] → B
[984321] → C
[972432] → D

(N. l)



[989994 → A
989432 → B
984321 → C
972432 → D]

N

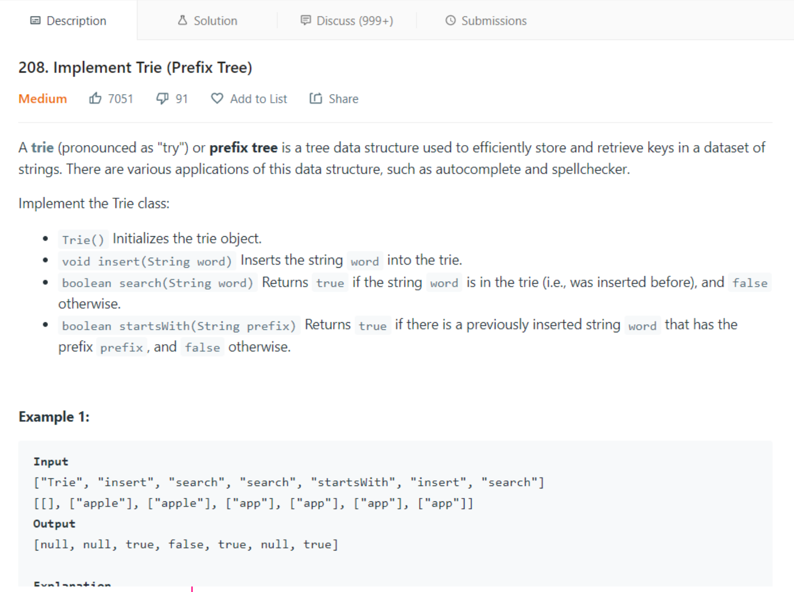
989 (l)

989

O (N.l)

O(l)

$l_1 + l_2 + l_3 \ldots + l_n = \Sigma$



Description | Solution | Discuss (999+) | Submissions

208. Implement Trie (Prefix Tree)

Medium 7051 91 Add to List Share

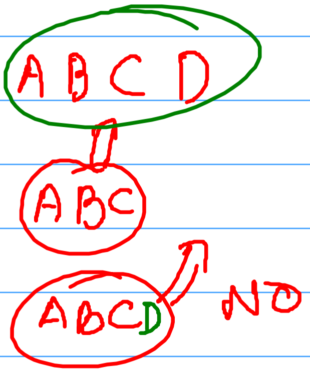
A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

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.

Example 1:

```
Input
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output
[null, null, true, false, true, null, true]
```



ABCD
PQR
XYZ

"yes"
PQ- prefix

XY as word
"No"

XYZ as prefix
"yes"

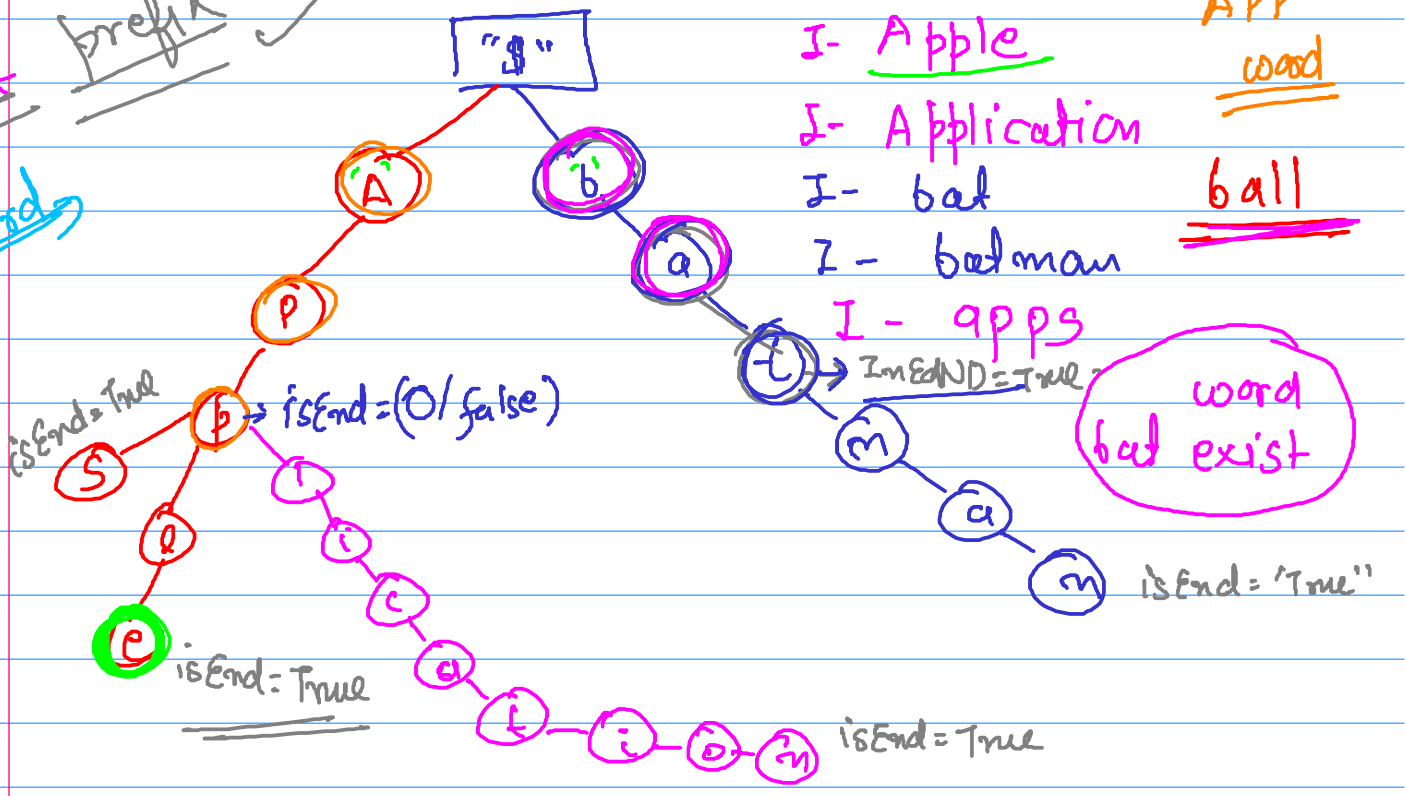
apple search(Apple)

app(search) (app

prefix

bat

word



I- Apple

I- Application

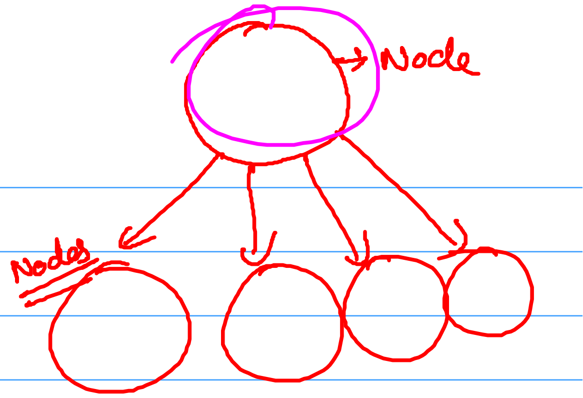
I- bat

I- batman

I- apps

APP word

ball

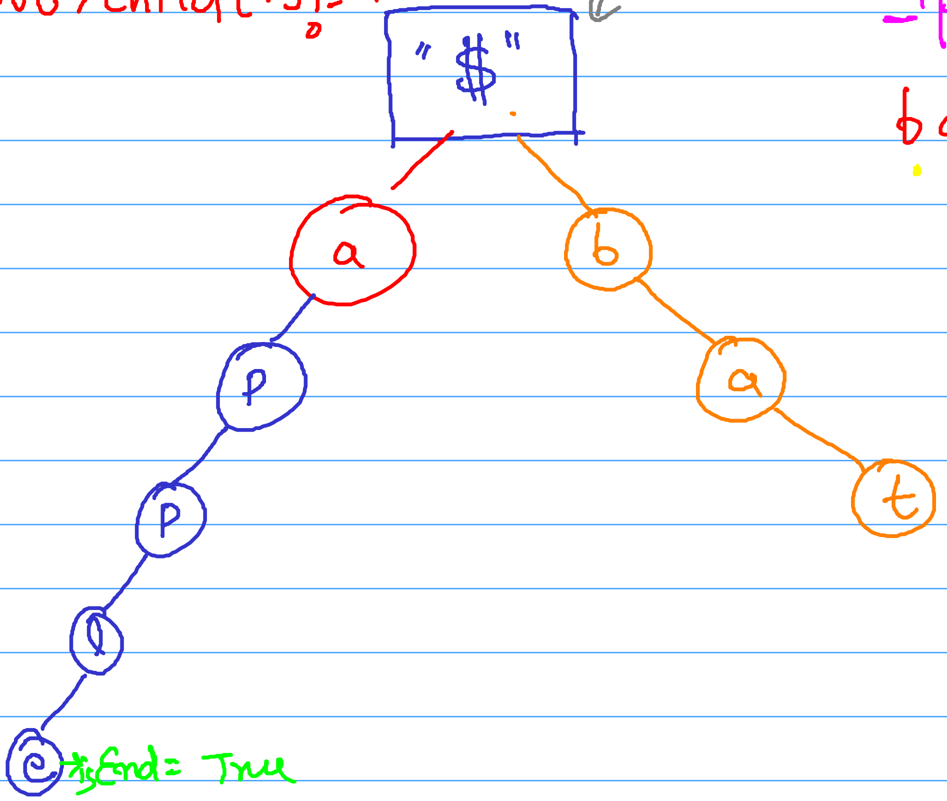


```
char ch;
Node* child[26];
bool isEnd;
```

⊙ (root→child[1]!=NULL)

a = 0

b ⇓ 1



apple

bat

```
help( Node* root)
{
    ~~root=NULL;~~
    t→val=100;
}
```

```
main()
{
    Node* root ;
    root = new Node(10);
    help( root)
    cout<<root→val<<endl;
}
```

RTE

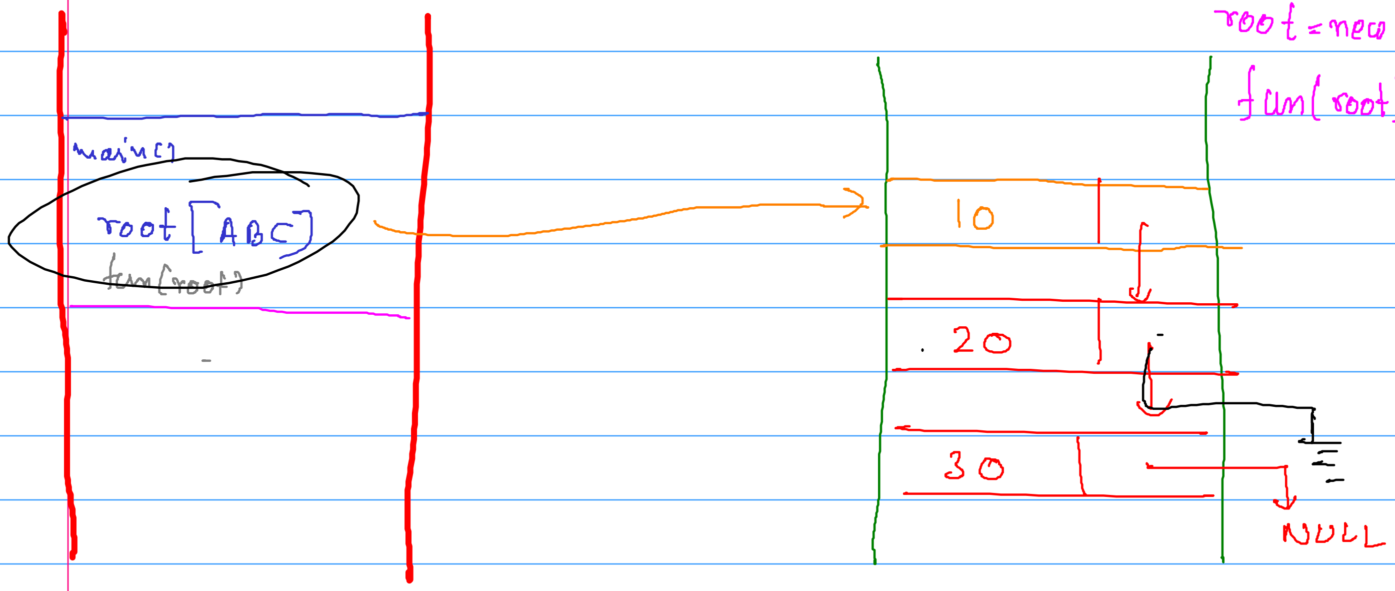
10

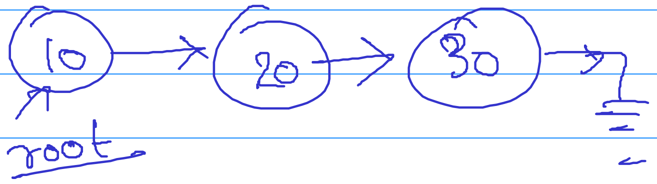
{ val
next

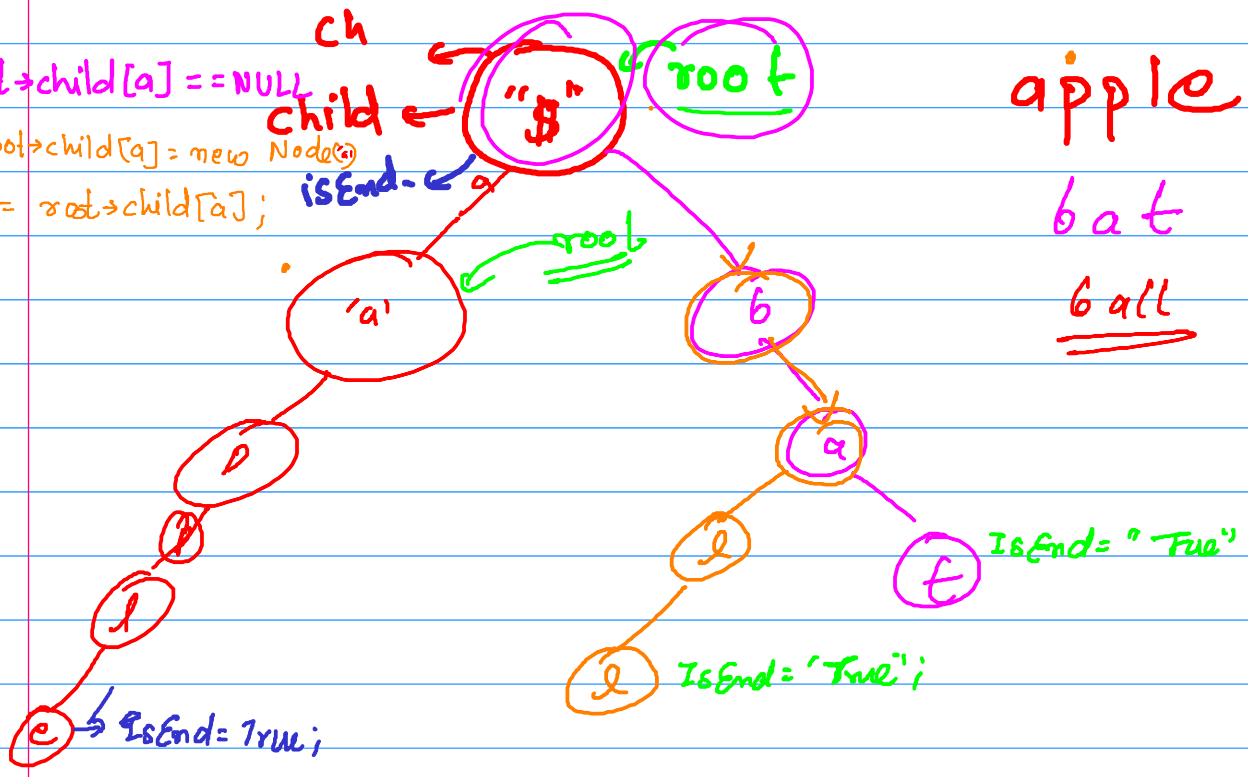
Stack (Limited)

Heap

```
main()
{
  Node* root;
  root = new Node();
  fun(root)
```





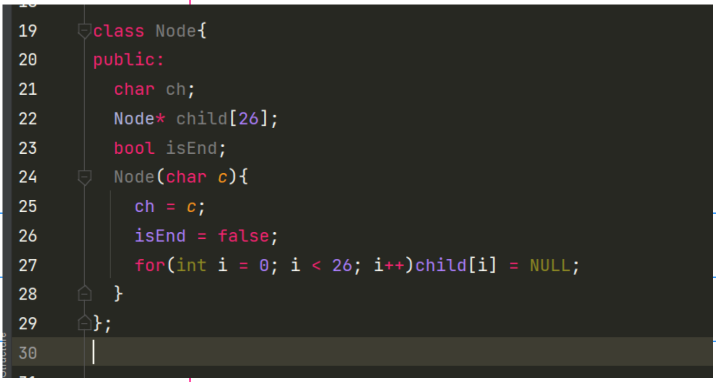


```
root→child[a] == NULL
root→child[a] = new Node()
root = root→child[a];
```

apple

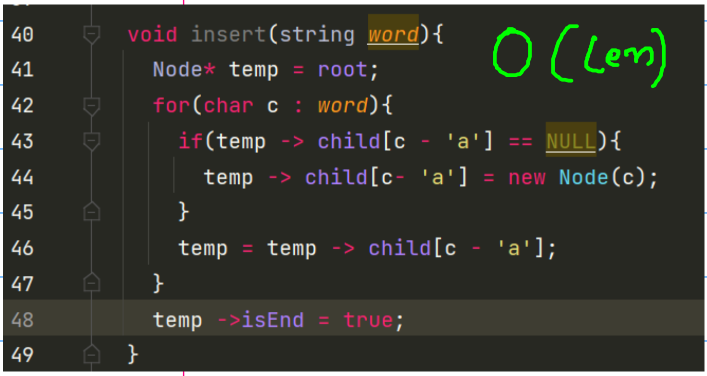
bat

ball



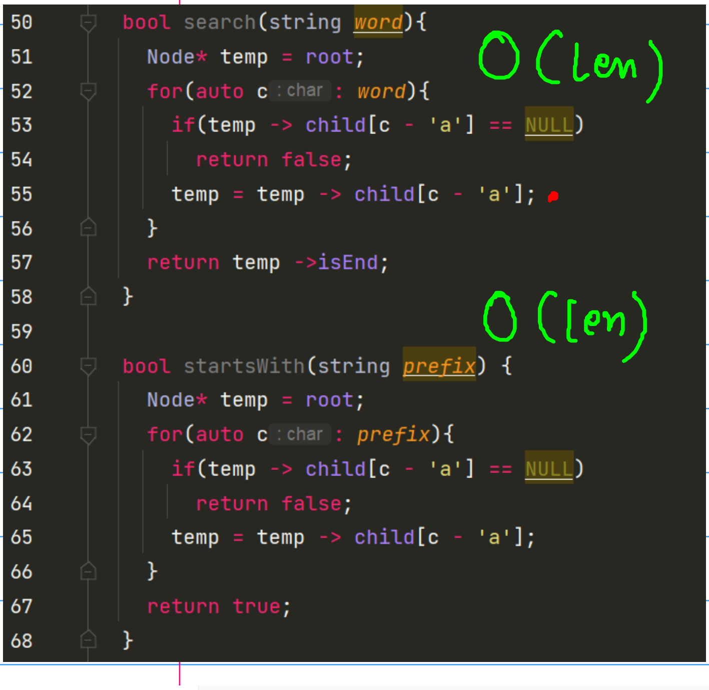
```
class Node{
public:
  char ch;
  Node* child[26];
  bool isEnd;
  Node(char c){
    ch = c;
    isEnd = false;
    for(int i = 0; i < 26; i++)child[i] = NULL;
  }
};

```

APPLE



```
  void insert(string word){
    Node* temp = root;
    for(char c : word){
      if(temp -> child[c - 'a'] == NULL){
        temp -> child[c- 'a'] = new Node(c);
      }
      temp = temp -> child[c - 'a'];
    }
    temp ->isEnd = true;
  }
```

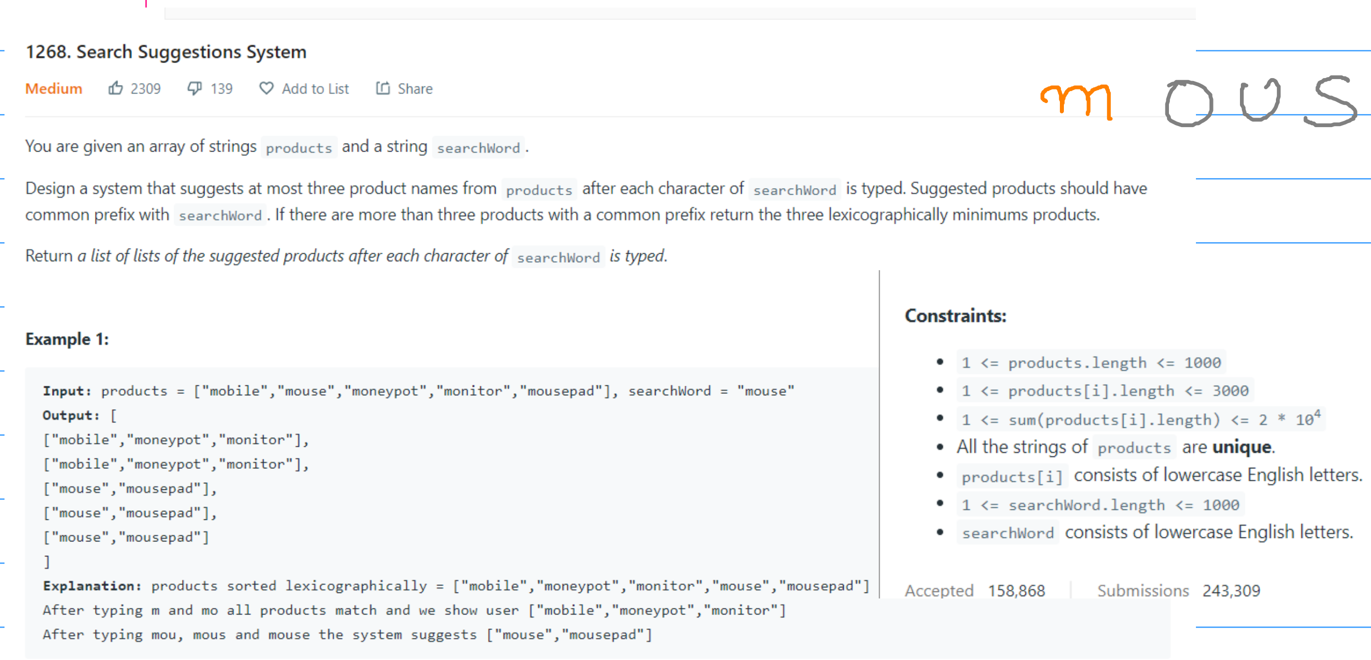
O(Len)



```
  bool search(string word){
    Node* temp = root;
    for(auto c :char : word){
      if(temp -> child[c - 'a'] == NULL)
        return false;
      temp = temp -> child[c - 'a'];
    }
    return temp ->isEnd;
  }

  bool startsWith(string prefix) {
    Node* temp = root;
    for(auto c :char : prefix){
      if(temp -> child[c - 'a'] == NULL)
        return false;
      temp = temp -> child[c - 'a'];
    }
    return true;
  }
```

O(Len)

O(len)



1268. Search Suggestions System

mouse

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.*

Example 1:

```
Input: products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord = "mouse"
Output: [
["mobile","moneypot","monitor"],
["mobile","moneypot","monitor"],
["mouse","mousepad"],
["mouse","mousepad"],
["mouse","mousepad"]
]
Explanation: products sorted lexicographically = ["mobile","moneypot","monitor","mouse","mousepad"]
After typing m and mo all products match and we show user ["mobile","moneypot","monitor"]
After typing mou, mous and mouse the system suggests ["mouse","mousepad"]
```

Constraints:

- `1 <= products.length <= 1000`
- `1 <= products[i].length <= 3000`
- `1 <= sum(products[i].length) <=` $2 * 10^4$
- All the strings of `products` are **unique**.
- `products[i]` consists of lowercase English letters.
- `1 <= searchWord.length <= 1000`
- `searchWord` consists of lowercase English letters.

Accepted 158,868 | Submissions 243,309

(M   M)

vector

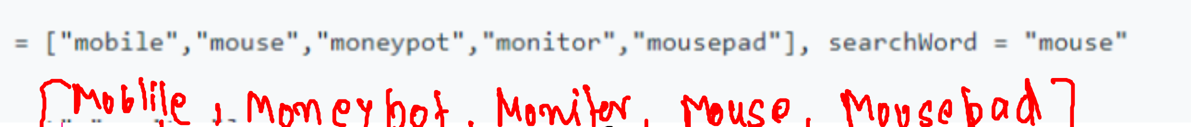
M → (, ,)

MAP

3 → word

{

}


= ["mobile","mouse","moneypot","monitor","mousepad"], searchWord = "mouse"

[Mobile, Moneypot, Monitor, Mouse, Mousepad]

(Mobie, Mon, monitor)

(mobie, Mon, munter)

Mony, moupad

(Mobile, Moneypot)

(Mobile

(Mobile

(Mobie

(Mobile

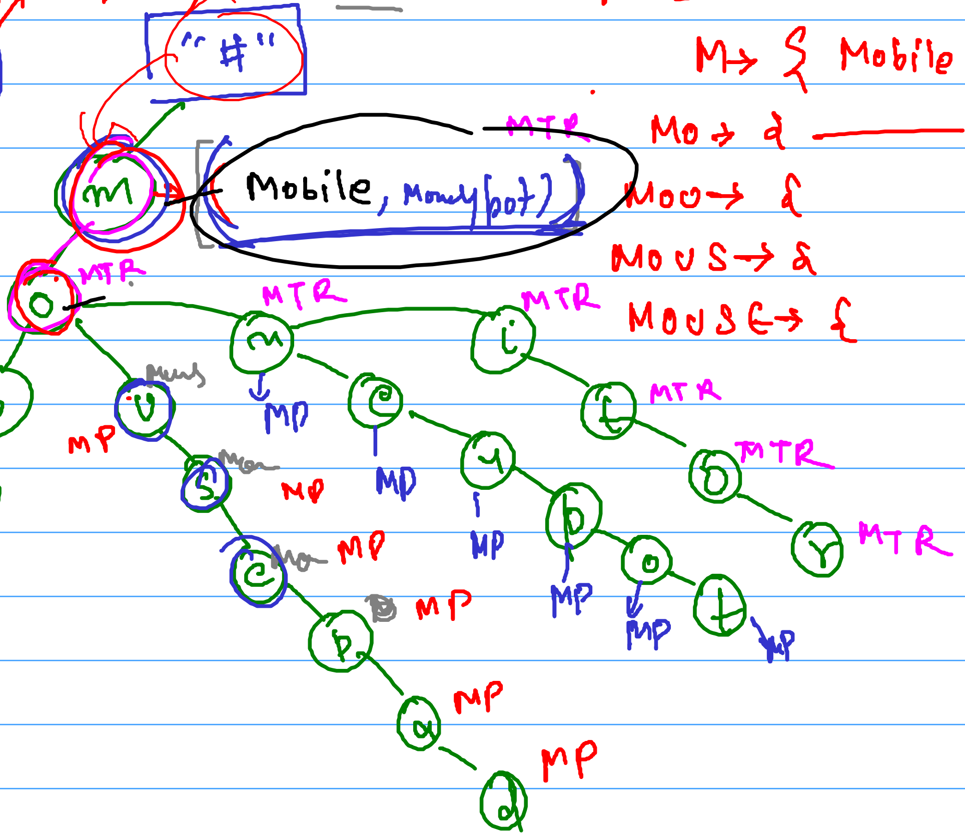
M → { Mobile, m, m

Mo → { ———— }

Moo → { }

MOUS → { }

MOUSE → { }



```
char ch;
child [];
isEnd;
vector<string> ANS.
```

[ ]-
[ ]-
[ ]-
[ ]-

```
void insert (string word){
    Node* temp: root;
    for(auto c: word){
        if(temp->child[c-'a'] == NULL)
            temp->child[c-'a'] = new Node(c);
        temp-> child [c-'a'];
        if(temp->Ans.size()<3)
        temp->Ans.pb(word);
    }
    temp->isEnd = True;
}-
```

```
for (aut c: word){
    if (temp->child[c-'a'] == NULL)
        return;
    temp= temp->child(c-'a');
    result.pb(temp->Ans);
}
```

⊙ Trie & XOR Problems

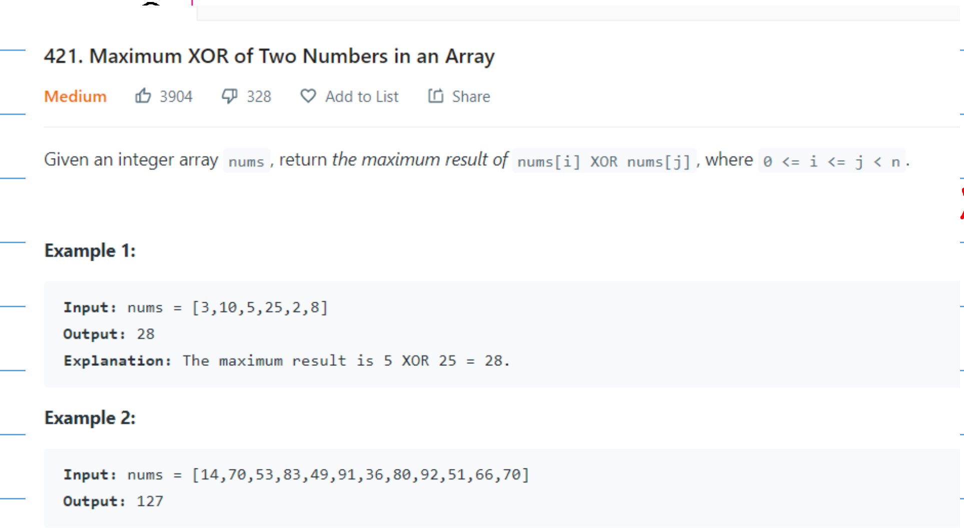
⊙ Given an array of N integer & an int k.
count No. of pairs i < j s.t.
$a(i) \wedge a(j) = k$

$a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \ldots a_j$

$$\boxed{a_i} \wedge a_j = k$$

$$a_i \wedge \cancel{a_j} \wedge \cancel{a_j} = a_j \wedge k$$

$$\boxed{a_i = a_j \wedge k}$$



421. Maximum XOR of Two Numbers in an Array

Medium 3904 328 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.
```

Example 2:

```
Input: nums = [14,70,53,83,49,91,36,80,92,51,66,70]
Output: 127
```

3rd

X :

(0011) (1010) (0101) 8 (1000)

3, 10, 5, 2, 8

0 9 15 (0010) 13

0011

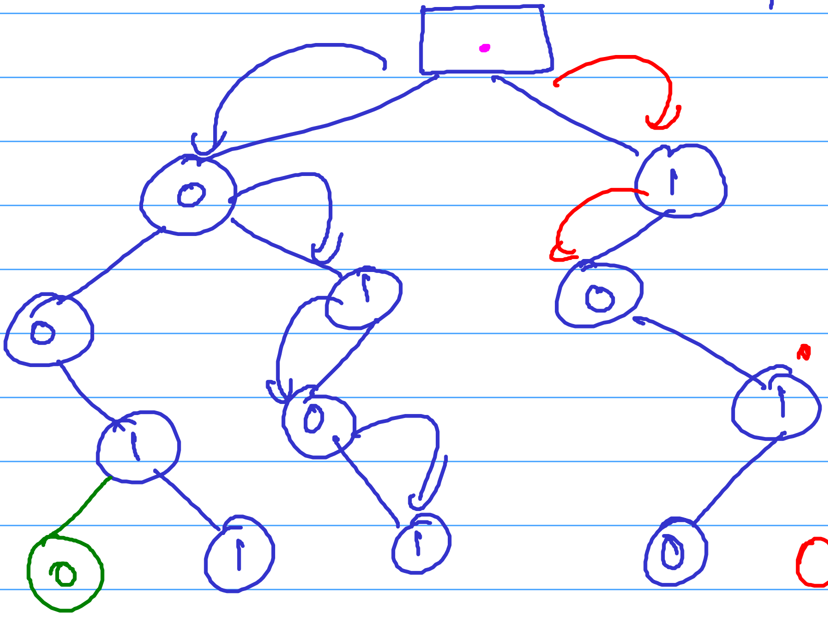
0

1000

1101

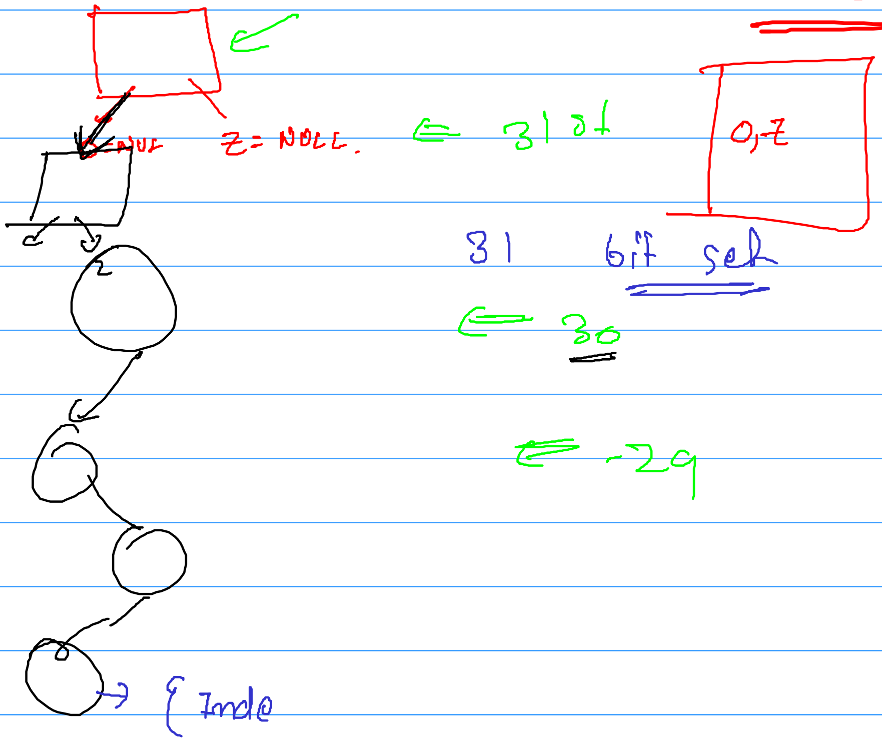
0101

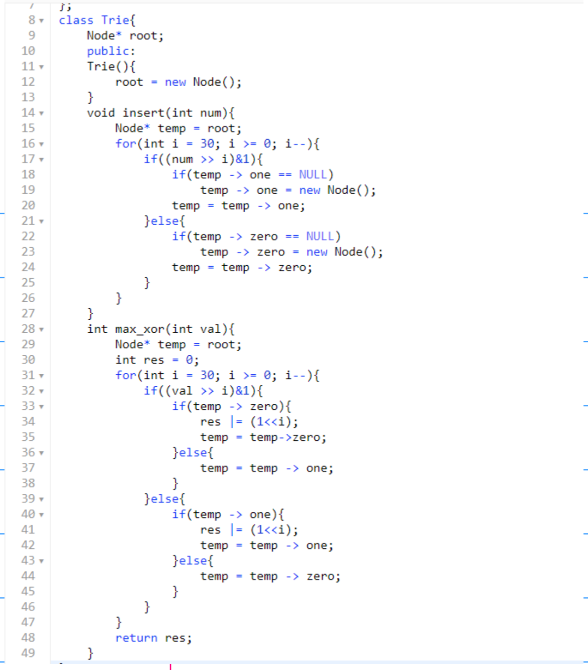
1111 15



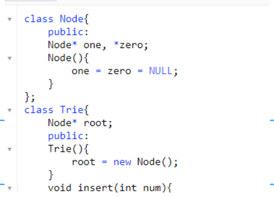
00100

1000

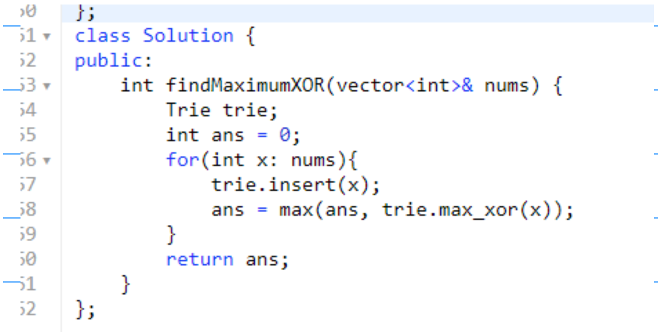




```
};
class Trie{
    Node* root;
    public:
    Trie(){
        root = new Node();
    }
    void insert(int num){
        Node* temp = root;
        for(int i = 30; i >= 0; i--){
            if((num >> i)&1){
                if(temp -> one == NULL)
                    temp -> one = new Node();
                temp = temp -> one;
            }else{
                if(temp -> zero == NULL)
                    temp -> zero = new Node();
                temp = temp -> zero;
            }
        }
    }
    int max_xor(int val){
        Node* temp = root;
        int res = 0;
        for(int i = 30; i >= 0; i--){
            if((val >> i)&1){
                if(temp -> zero){
                    res |= (1<<i);
                    temp = temp->zero;
                }else{
                    temp = temp -> one;
                }
            }else{
                if(temp -> one){
                    res |= (1<<i);
                    temp = temp -> one;
                }else{
                    temp = temp -> zero;
                }
            }
        }
        return res;
    }
```



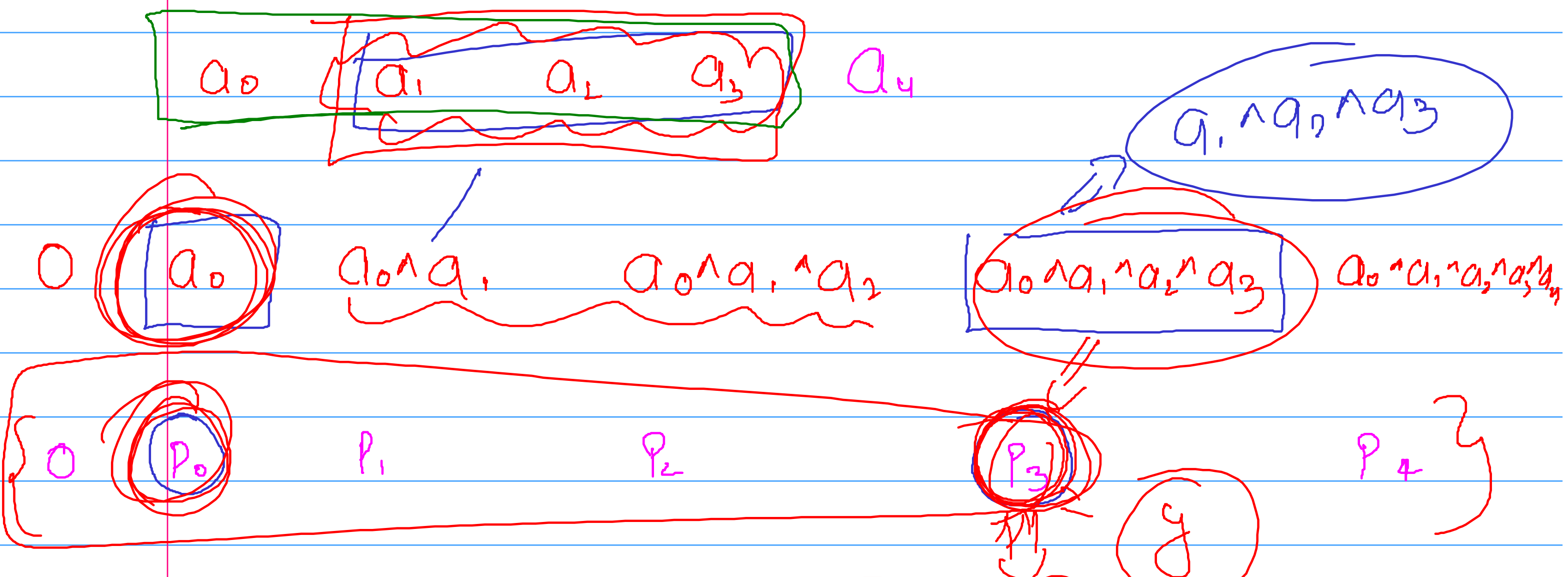
```
class Node{
    public:
    Node* one, *zero;
    Node(){
        one = zero = NULL;
    }
};
class Trie{
    Node* root;
    public:
    Trie(){
        root = new Node();
    }
    void insert(int num){
```

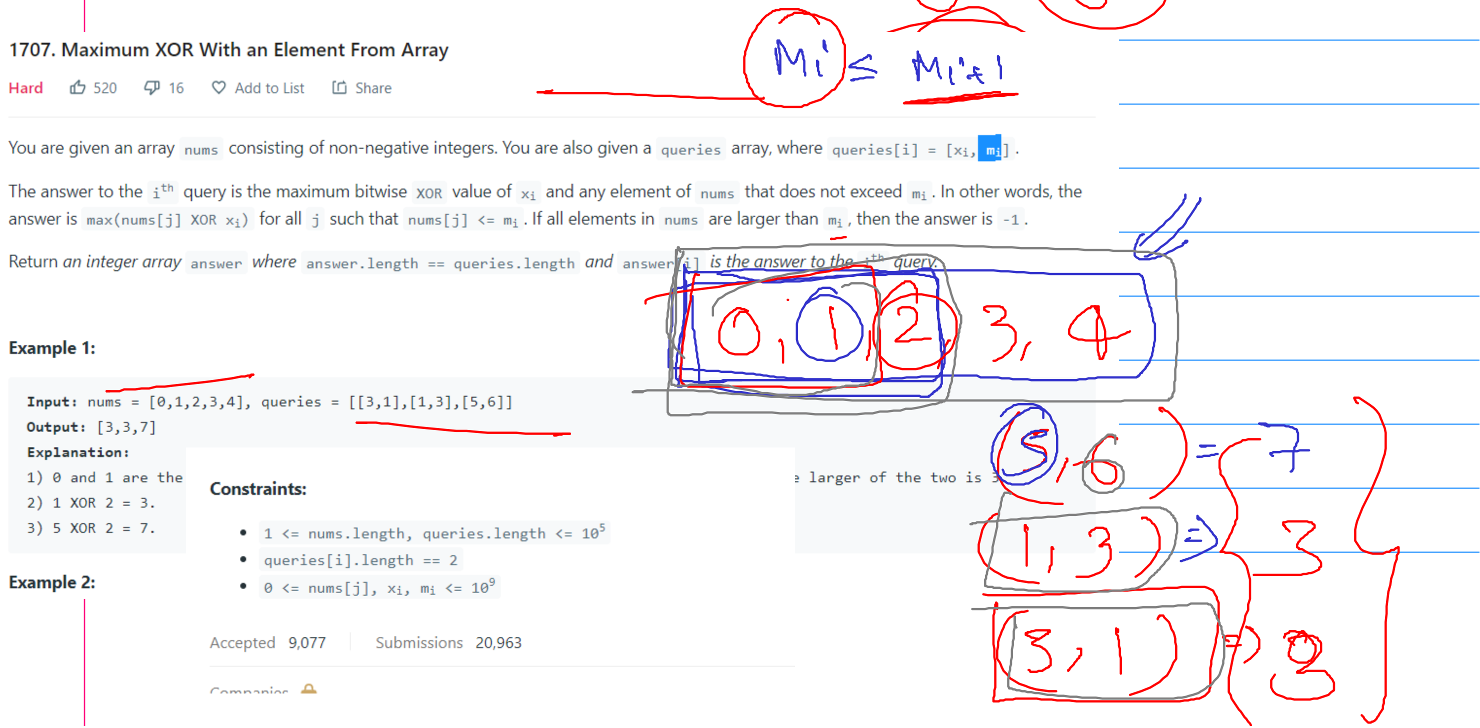


```
};
class Solution {
public:
    int findMaximumXOR(vector<int>& nums) {
        Trie trie;
        int ans = 0;
        for(int x: nums){
            trie.insert(x);
            ans = max(ans, trie.max_xor(x));
        }
        return ans;
    }
};
```

O(N. (Max No. bits)).

MAXIMUM SOBARRAY XOR





1707. Maximum XOR With an Element From Array

Hard 520 16 Add to List Share

You are given an array `nums` consisting of non-negative integers. You are also given a `queries` array, where `queries[i] = [xi, mi]`.

The answer to the `ith` query is the maximum bitwise `XOR` value of `xi` and any element of `nums` that does not exceed `mi`. In other words, the answer is `max(nums[j] XOR xi)` for all `j` such that `nums[j] <= mi`. If all elements in `nums` are larger than `mi`, then the answer is `-1`.

Return *an integer array* `answer` *where* `answer.length == queries.length` *and* `answer[i]` *is the answer to the* `ith` *query.*

Example 1:

```
Input: nums = [0,1,2,3,4], queries = [[3,1],[1,3],[5,6]]
Output: [3,3,7]
Explanation:
1) 0 and 1 are the                                    e larger of the two is 3
2) 1 XOR 2 = 3.
3) 5 XOR 2 = 7.
```

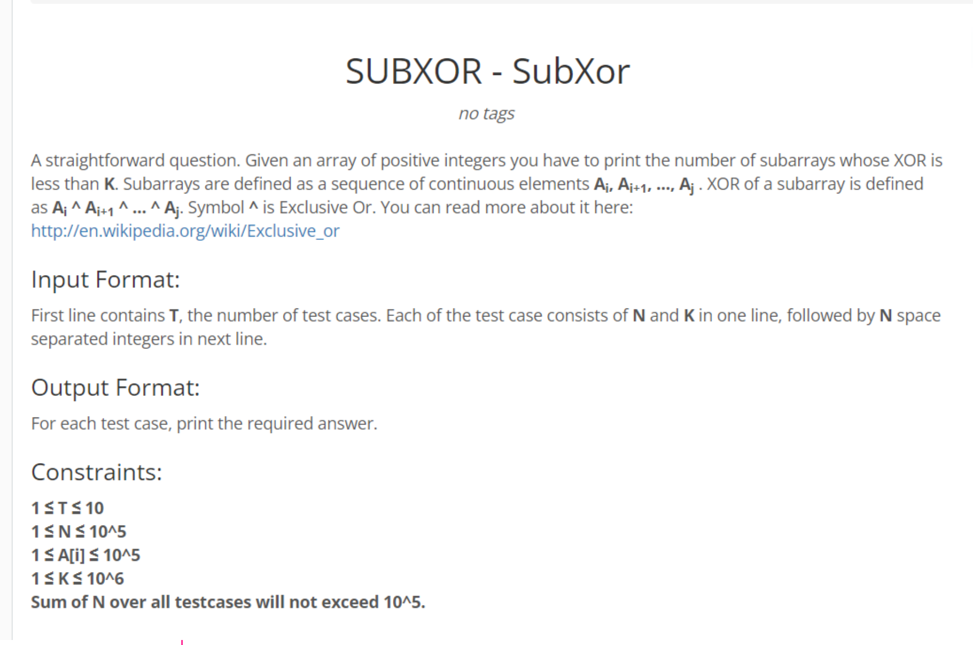
Example 2:

Constraints:

- `1 <= nums.length, queries.length <= 10^5`
- `queries[i].length == 2`
- `0 <= nums[j], xi, mi <= 10^9`

Accepted 9,077 Submissions 20,963

Companies



SUBXOR - SubXor

no tags

A straightforward question. Given an array of positive integers you have to print the number of subarrays whose XOR is less than **K**. Subarrays are defined as a sequence of continuous elements **A_i, A_{i+1}, ..., A_j** . XOR of a subarray is defined as **A_i ^ A_{i+1} ^ ... ^ A_j**. Symbol ^ is Exclusive Or. You can read more about it here: http://en.wikipedia.org/wiki/Exclusive_or

Input Format:

First line contains **T**, the number of test cases. Each of the test case consists of **N** and **K** in one line, followed by **N** space separated integers in next line.

Output Format:

For each test case, print the required answer.

Constraints:

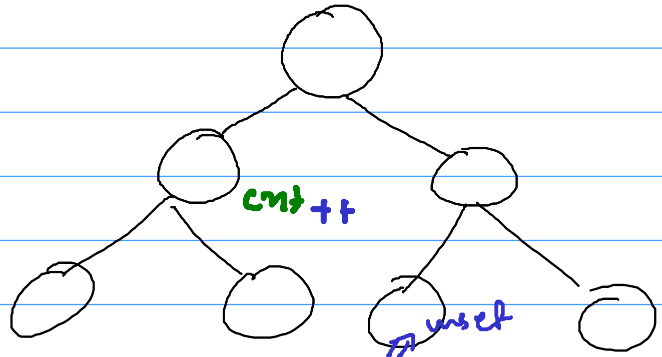
1 ≤ T ≤ 10
1 ≤ N ≤ 10^5
1 ≤ A[i] ≤ 10^5
1 ≤ K ≤ 10^6
Sum of N over all testcases will not exceed 10^5.

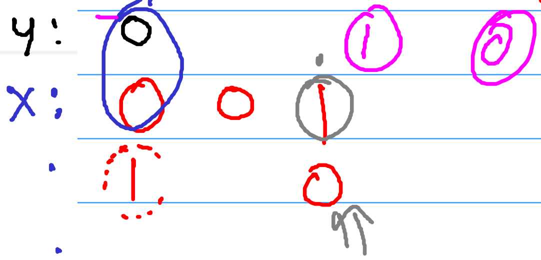
a_0 a_1 a_2 a_3 . . . a_{n-1}

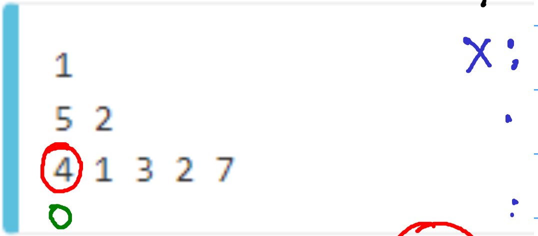
X: 1
Chng: X
k: 1

$a_i \wedge a_j < K$.

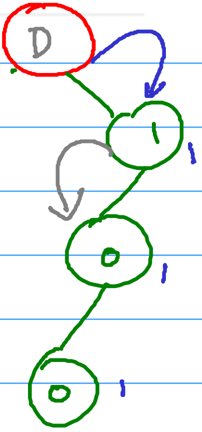
a_j







```
1
5 2
4 1 3 2 7
0
```



```
if ((x>>i)&1) {
    if ((k>>i)&1)
    {
        ans += (temp->one->cnt)
        temp = temp->zero;
    }
    else {

    }.
```

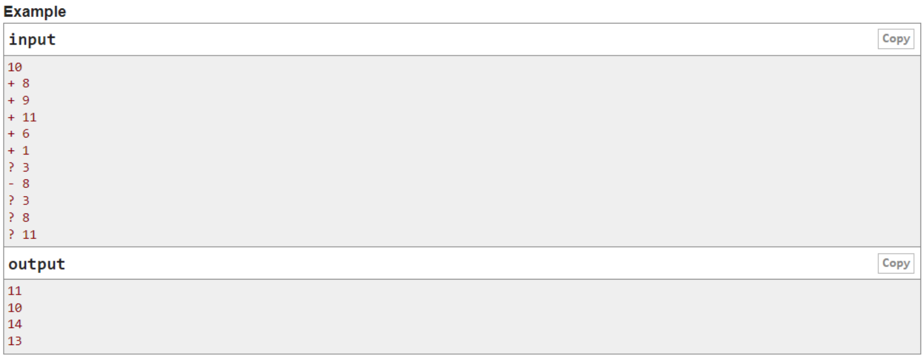
```
if ((val>>i)&1) {
    if((k>>i)&1) {
        res += (temp->one->cnt);
        temp-> temp->zero;
    }else{
        temp: temp->one;
    }
}
```

```
else {
    if((k>>i)&1){
        res += (temp->zero->cnt);
        temp= temp->one;
    }
    else{
        temp= temp->zero;
    }
}
```

val : 1 → ⓞ

k : 0

val : $\overset{\wedge(0)}{0} \to 0$



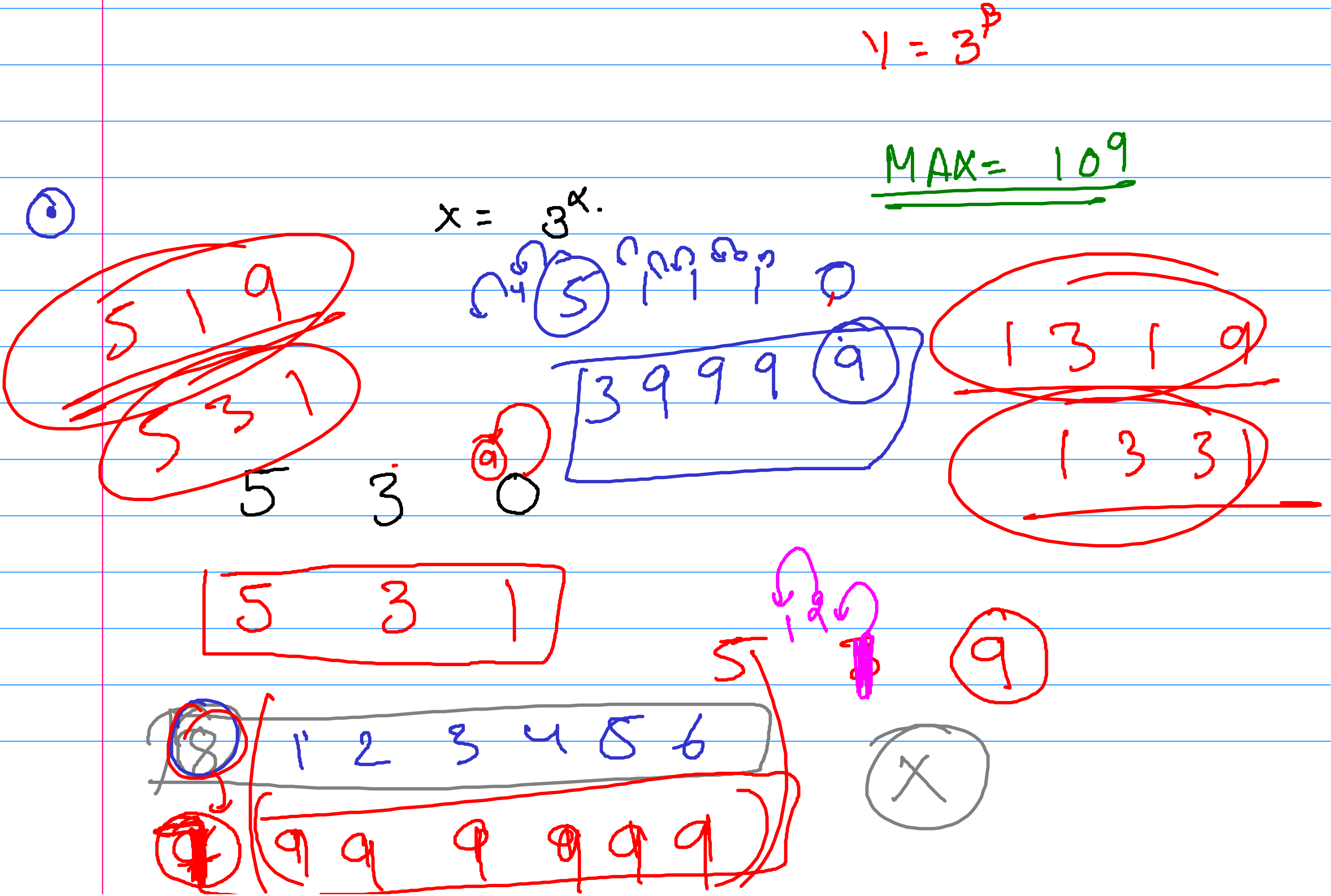
Example

input

```
10
+ 8
+ 9
+ 11
+ 6
+ 1
? 3
- 8
? 3
? 8
? 11
```

output

```
11
10
14
13
```



0

S[i] = '0'

1 9 9 9 9

'9' '9' '9 9 9 9 9'

S[i] = '9';

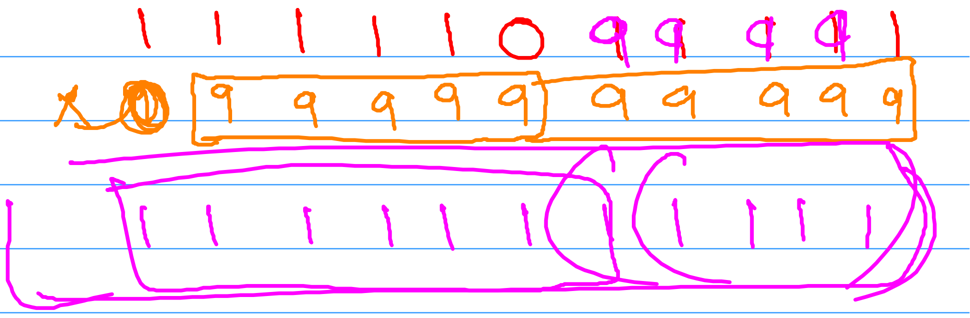
```
j = i-1;
while(j>=0 && S[j]=='0'){
        S[j]= '9';
        j--;
}
if(j==-1){
        S[0]= '0'
else{
        S[j] -= 2;
}
```

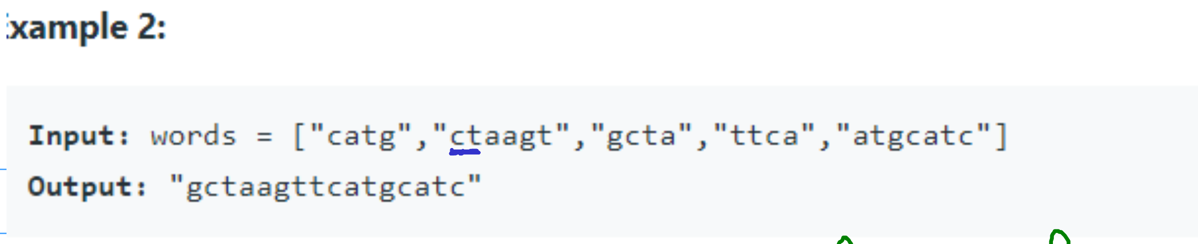
9 9 9 9 9 9

3 1 1 1 0

0 9 9 9 9

19999





xample 2:

```
Input: words = ["catg","ctaagt","gcta","ttca","atgcatc"]
Output: "gctaagttcatgcatc"
```

c a t g

{ catg

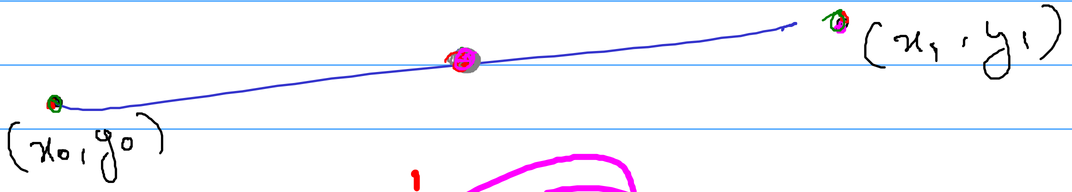
& { atg

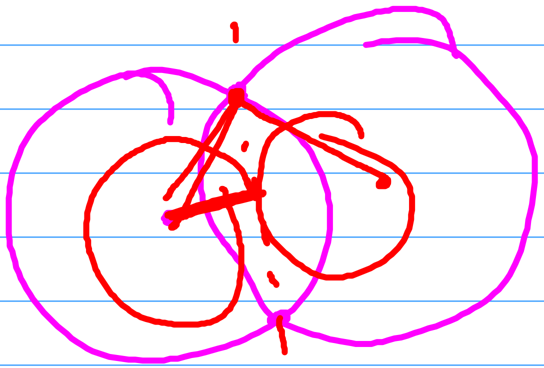
ca& tg

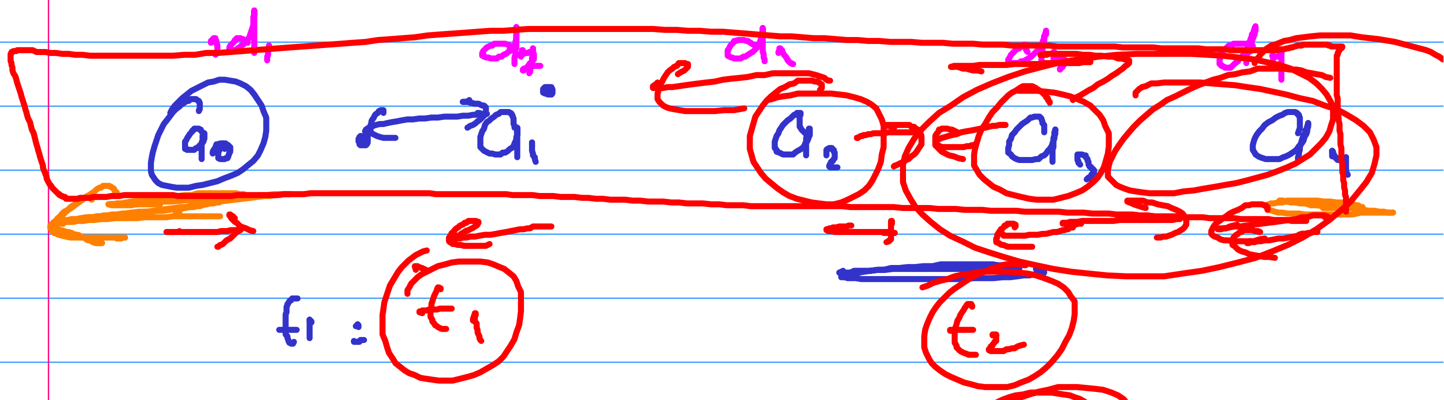
cat& g

catg& " " }

idx, vis







$a_0 - t_2$

$a_0 + t_2$

$\underline{\underline{O}}$

N^2.