# TRIE

# Handwritten Notes of Striver(TUF) Playlist

by: Aashish Kumar Nayak

NIT Srinagar

AASHISH KUMAR NAYAK

aashishkumar.nayak

# TRIE SERIES

L1. Implement TRIE | INSERT | SEARCH | STARTSWITH

L2. Implement TRIE-2 | INSERT | countWordsEqualTo() | countWordsStarting-with() |

L3. Longest Word with All prefixes | complete string | Trie

L4. Number of Distinct substrings in a string | Trie |

L5. Bit prerequisites for TRIE problems

L6. Maximum XOR of Two Numbers in An Array

L7. Maximum XOR with an Element From Array | Queries

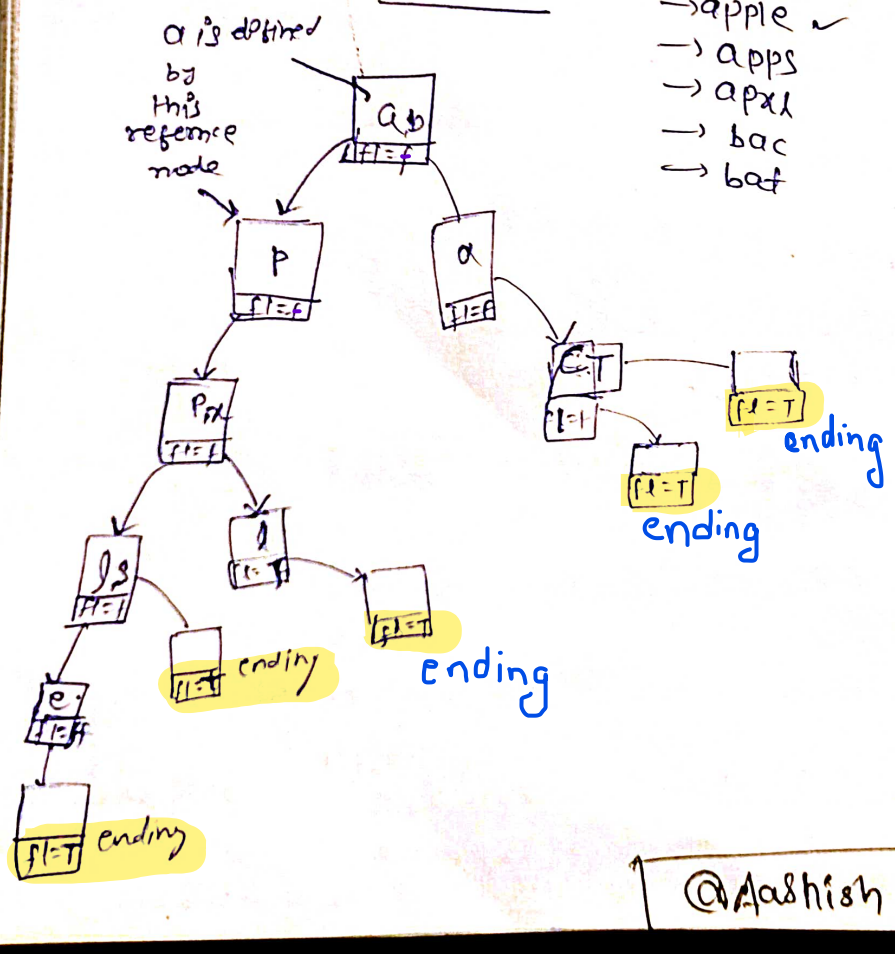
# TRIE

## L-1 [Implement] TRIE | INSERT | SEARCH | STARTWITH

Trie data structure is used whenever we have insert(word) then they will asking you is this particular word exits.
and is there any word where prefix is this word.

```
5
I hello   // means insert hello
1 help    // means insert help
2 help    // means search help // output True
3 hel     // means is there any word starts with hel // output True
2 hel     // means search hel // output false.
```

### Trie INSERT




```
Trie { Node*
       a[26];
       bool fl;
     }
```

@Aashish Kumar Nayak

## Search functionality.

lets say some search of apps

we will traverse in trie and if our last reference's flag is True means we got that word. else return false.

## Starts with functionality

letsay → ba
→ apd

we will traverse in trie and if we are standing at a not null position, means there is word with given prefix.
if we encounter a null during traversal of trie we will mean we will not get any word with that prefix.

code :-

```
Struct Node {
    Node *links[26];
    bool flag = false;
    bool containKey(char ch)
        { return (links[ch-'a'] != NULL);
        }
    void put(char ch, Node *node)
        { links[ch-'a'] = node;
    Node *get(char ch)
        { return links[ch-'a'];
        }
    void setEnd()
        { flag = true;
        }
    bool isEnd()
        { return flag;
        }
};
```

@Aashish kumar Nayak

```
Class Trie {
  private:
      Node* root;
  public:
  Trie(){
          root = new Node();
        }
  void Insert(string word)
  {   Node *node=root;
      for(int i=0 ; i< word.size() ; i++)
        { if(!node->Containkey(word[i]))
            { node->put(word[i], new Node());
            }
        node = node->get(word[i]);
      }
      node->setEnd();
  }

~~void~~ bool search(string word)
{   Node *node = root;
    for(int i=0; i<word.size(); i++)
      { if(!node->containkey(word[i]))
          { return false;
          }
        node = node->get(word[i]);
      }
      return node->isEnd();
}
```

```
bool startswith(string prefix)
{
    Node* node = root;
    for(int i=0; i<prefix.size(); i++)
    {   if(!node->containkey(prefix[i]))
        {   return false;
        }
        node = node->get(prefix[i]);
    }
    return true;
}};
int main()
{   int n=5;
    vector<int> type = {1,1,2,3,2};
    vector<string> value = {"hello", "help", "help", "hel", "hel"};
    Trie trie;
    for(int i=0; i<n; i++)
    {   if(type[i]==1)
        {   trie.insert(value[i]);
        }
        else if(type[i]==2) {if(trie.search(value[i])) {
            cout<<"true"<<"\n";
        else cout<<"false"<<"\n"; }
        else {
            if(trie.startswith(value[i]))
            {   cout<<"true"<<"\n";
            }
            else{cout<<"false"<<"\n"; }
    }}}
```

@Aashish kumar nayak

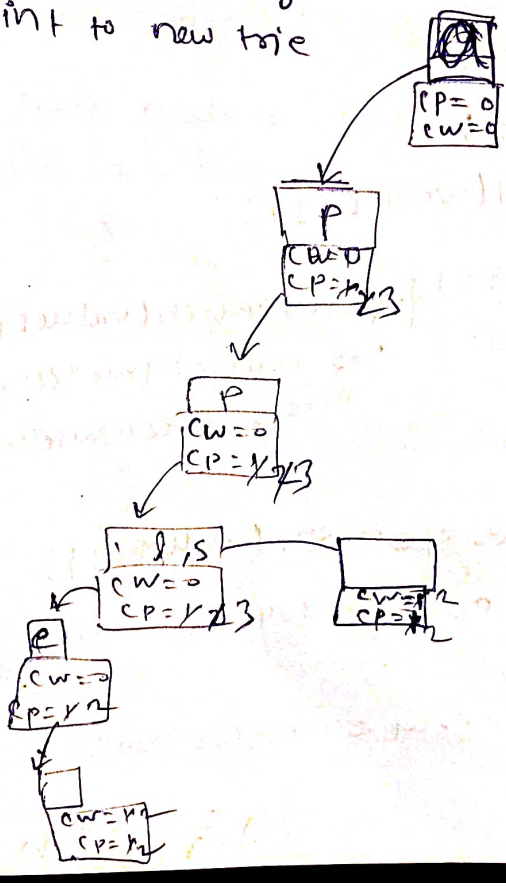
# Implement Trie - II

implement functionalities

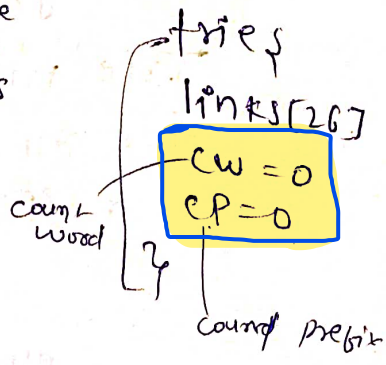
1. countWords equalTo("word")
2. countWordsStartingWith("word")
3. erase("word")

```
1
6
insert  samsung
insert  sumsung
insert  vivo
erase   vivo
countWordsEqualTo  samsung   // output - 2
countWordsStartingWith vi    // output - 0
```

insert "a" and having point to new trie




- apple
- apple
- apps
- apps




@Aashish Kumar Nayak

erase

Reduce the count prefix and also at the end also reduce count word.

Code :-

```
Struct Node {
    Node *links[26];
    int cntEndwidth = 0;
    int cntprefix = 0;

  bool containsKey(char ch)
{ return (links[ch-'a'] != NULL);
 }
Node *get(char ch)
 { return links[ch-'a'];
 }
void put(char ch, Node* node)
 {  links[ch-'a'] = node;
  }
  void increaseEnd()
    { cntEndwidth cntEndwidth++;
    }
  void increaseprefix()
    { cntprefix++;
    }
  void deleteEnd() {
  {  cntEndwith--;
  }
   int getEnd()
     { return cntEndwith;
     }
    int getprifix()
     { return cntprefix;
    }
};
```

```
class Trie {
private:
    Node * root;
public:
Trie() {
        root = new Node();
    }
void insert(string word)
{ Node *node = root;
for(int i=0; i< word.length(); i++)
    {
    if(!node->containsKey(word[i]))
        {
        node->put(word[i], new Node());
        }
    node = node->get(word[i]);
    node-> increasePrefix();
    }
node -> increaseEnd();
}
int countWordsEqualTo(string &word)
{
    Node *node = root;
for(int i=0; i< word.length(); i++)
    {
    If(node->containsKey(word[i]))
        { node = node->get(word[i]);
        }
    else
        {return 0;
        }
    }
return node->getEnd();
}
```

```
int countwordsStartingwith( string &word)
{
    Node *node =root;
    for(int i=0; i<word.length(); i++)
    {
        if(node->containskey(word[i]))
        {
            node = node->get(word[i]);
        }
        else {
            return 0;
        }
    }
    return node->getprefix();
}
void erase(string &word)
{
    Node *node = root;
    for(int i=0; i<word.length(); i++)
    {
        if(node->containskey(word[i]))
        {
            node = node->get(word[i]);
            node->reduceprefix();
        }
        else{
            return;
        }
        node->deleteEnd();
    }
};
int main()
{
    Trie T;
    T.insert("apple");
    T.insert("apple");
    T.insert("apps");
    T.insert("apps");
```

```
String word1 = "apps";
cout<<"counts words equal to"<<word1<<" "<< T.countwordsEqualTo(word1)<<endl;
String word2 = "abc";
cout<<"counts words Equal to"<<word2<<" "<< T.countwordsEqualTo(word2)<<endl;
String word3 = "ap";
cout<<"count words starting with"<<word3<<" "<<T.countwordsStartingwith(words)<<endl;
String word4 ="appl";
cout<<"count words starting with"<<word4<<" "<< T.countwordsstarting with(word4)<<endl;
T.erase(word1);
cout<<"count words equal to"<<word1<<" "<<T.countwordsEqualto(words)<<endl;
return 0;
}
```

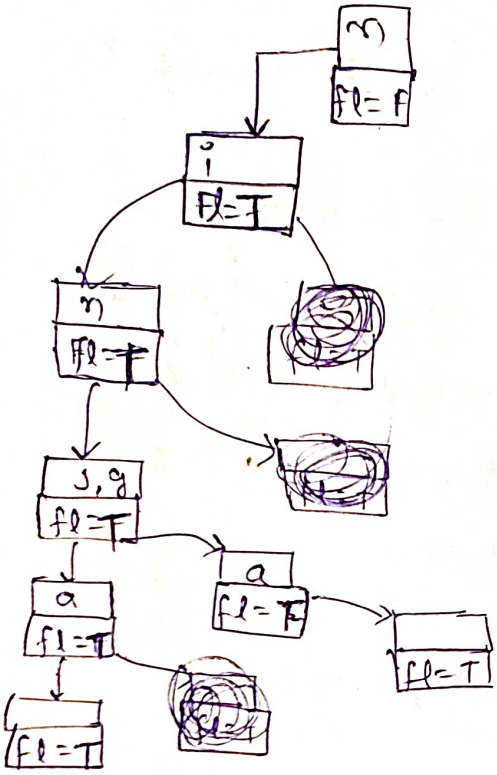
Aashish kumar Nayak

# Longest word with All Prefixes

complete strings

These should be C.S.

## Complete String




n
ninja
ninj
ni
nin
ninga

Insert all these prefixes

Trie {

arr[26]
bool flag
}

Complete string
n

T.C.
O(N) x O(len)  (string length)

S.C.
O(N x len)

**Code :-**

```
Struct Trie {
    Node* links[26];
    bool flag = false;

    bool containsKey(char ch)
    { return links[ch - 'a'];
    }

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

void put(char ch, Node* node)
  { links[ch - 'a'] = node;
  }
```

@Aashish Kumar Nayak

```
Void setEnd()
    {
      return flag = true;
    }
    bool isEnd()
      {
        return flag;
      }
};

Class Trie
{
  private : Trie root;
  public :
      Trie()
        {
          root = new Node();
        }
  public : void insert(String word)
  {  Node * node = root;
    for(int i=0; i<word.length(); i++)
        { if(!node->containsKey(word[i]))
          {
            node->put(word[i], new Node());
          }
          node= node->get(word[i]);
        }
    node->setEnd();
  }
public : bool checkPrefixExists(String word) {

  }
};
```

```
string completeString (int n, vector<string> &a)
{ Trie trie;
  for (auto &it : a)
  {
    trie.insert(it);
  }
  String longest = "";
  for (auto &it : a)
  { if(cheakIfprefixExists(it))
    { if(it.length() > longest.length())
      { longest = it;
      }
      else if (it.length() == longest.length() &&
                                it < longest)
      { longest = it;
      }
    }
  }
  if( longest == "" ){return "None";}
} return longest;
```

```
public : bool cheakIfprefixExists(string word)
{
    bool f1 = true;
    Node * node = root;
    for(int i = 0; i<word.length(); i++)
    { if(node->containsKey(word[i]))
      { node = node->get(word[i]);
        if(node->isEnd() == false) return false;
      }
      return false;
    }
  return true;
}
```

@Ashish Kumar Nayak

# Number of different substrings in a string

(Including empty string)

a b a b

starts with a: a, a b, a b a, a b a b

starts with b: b, b a, b a b

a x, b x, a b x — Already have been covered

all case distinct

4 + 3 + 1 = 7 + 1 = 8

1 for empty substring

```
Set <string> st;

for( i=0 ; -> n-1)
    str = "";
    for( j=i -> n-1)
        str = str + s[j];
        st.add(str); ---- log m

print(st.size());  -> distinct substring
```

T.C. $O(N^2)$

T.C. $O(N^2 \times \log M)$ (M: length of substring; log M: no. of elements in set)

all of the substring generated are distinct.

min -> 1

max -> n

$\approx (n/2)$

$n^2 \times n/2 \approx n^3$

a → O(1)
ab → O(2)
abc → O(3)

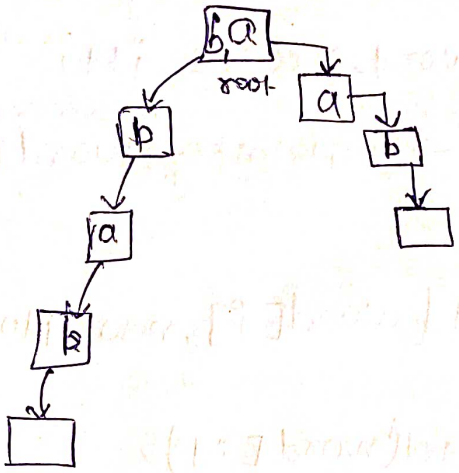
Set

aba
abad
} S.C. O(4) in trie

but in set

S.C.

O(3) + O(4)




abab

tries

arr[26];

}

++++ = 5

++ 3

2 + 1 (empty)

= (8)

distinct substring

T.C. → O(N²)

S.C. →

code:

```
class Node {
        Node * links[26];
        bool containsKey(char ch)
        { return links[ch-'a'];
        }
    void put(char ch, Node* node)
        { links[ch-'a'] = node;
    }
    Node * get(char ch)
        { return links[ch-'a'];
        }
};
int countDistinctSubstrings(string &s)
{   int cnt = 0;
    Node* root = new Node();
    for(int i=0; i<word.size(); i++)
        { Node * node = root;
```

```
for(int j = i; j<word.size(); j++)
    {if(!node->containsKey(word[i]))
     {cnt++;
      node->put(word[i], new Node());
     }
     node = node->get(word[i]);
    }
  }
  return cnt+1;
}
```

## Bit prerequisites for Tries

9 → 1001 → 63 ... 31 ... 3 2 1 0 | 0 0 0 0 ... ... 3 2 1 0 | ... 1001

32 bits / 64 bits

## XOR

1/0

1^0 = 1
0^1 = 1
} XOR of different bits gives you 1.

0^0 = 0
1^1 = 0
} XOR of same bits gives you 0.

if there are even no. of 1s → 0 (it gives you)
if there are odd no. of 1's → 1 (it gives you)

## How to check if a bit is set or not

9 → 0000 ... 1001

3rd bit → 1

0 0 0 ... 0 0 0 1
0 0 0 1

(num >> 3) & 1

if (num >> i) & 1 != 0) ✓

else (num >> i & 1 == ) ✗

## How do we turn on a bit

IMP

9 → 0 0 0 . . . . 1 0 0 1 (i = 2, bits 2,1,0)

0 0 0 . . . . 0 0 0 1

0 0 0 . . . . 0 1 0 0 (1 << 2)

1 1 0 1

num = num | (1 << i)

## MAXIMUM XOR OF TWO NUMBERS in an Array

Given an array of numbers, & a number x,
find the max value of (arr[i]^n).

→ insert all the numbers into the trie.
→ Take x & find the max no. from array where (no ^ x) ↑↑
(binary bits)

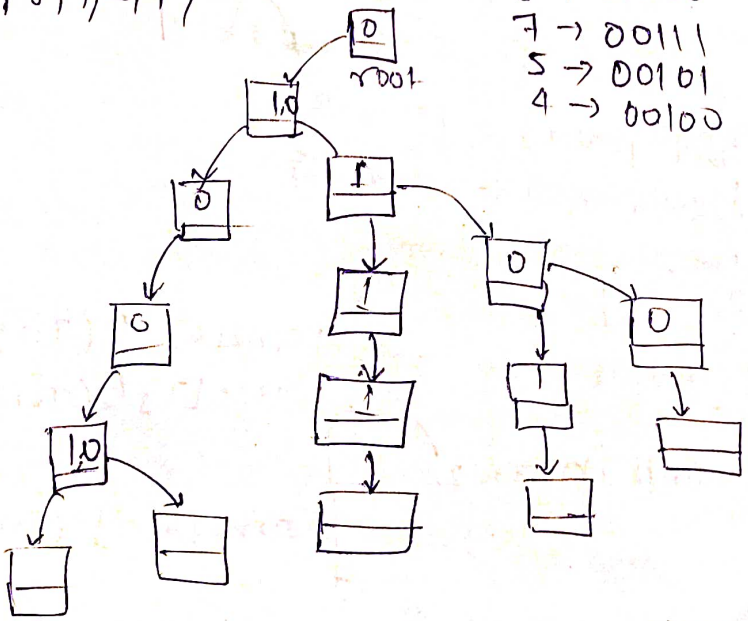
9 → (0 0 0 0 . . . . . 0 1 0 0 1) → we copy store it as a string

let's understand in terms of 5 bit

arr[] = {9, 8, 7, 5, 4}

9 → 01001
8 → 01000
7 → 00111
5 → 00101
4 → 00100

Nodes




Now fig. out which is the largest

# Intution of Maximising xor

$x = 8 \wedge a[i]$ — ↑↑ as high as possible

$2^4\ 2^3\ 2^2\ 2^1\ 2^0$

0 1000

a[i] → 1 0111

1 1111 ↑↑ 31

9 → 0 1001 ✗

8 → 0 1000 ✗

7 → 00111 ✓

5 → 00101 ✓ ✗

4 → 00100 ✓ ✗

x → ✓

0 1000 0

0 0 1 1 1

0 1111 1 → 15 max

need 0

need 1

need 1 but not available so two 0.

$x = 8$

9th 3rd 2nd 1st 0th

0 1000

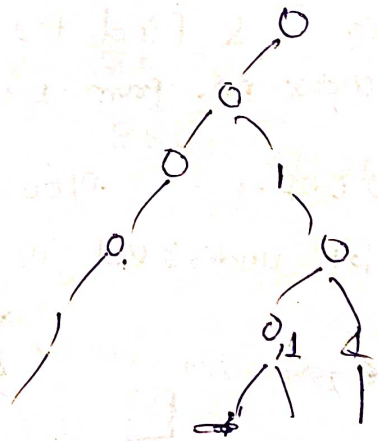
0 0111

0 0000

✗ 1111

root

create 1st 1 or 0



arr[] → x

arr2[] → y

$(x \wedge y) = ??$ ↑↑

arr1 → insert in Trie

x → all the elements in arr2

arr2[0] ^ (Trie)

arr2[1] ^ (Trie)

arr2[n-1] ^ (Trie)

maximum of these.

@Aashish Kumar Nayak

```
struct Node {
    Node * links[2];
    bool containsKey(int ind)
    { return (links[ind] != NULL); }
    Node * get(int ind)
    { return links[ind]; }
    void put(int ind, Node* node)
    { links[ind] = node; }
};

class Trie {
    private: Node *root;
    public:
        Trie() { root = new Node(); }
    public:
        void insert(int num)
        { Node* node = root;
          for(int i = 31; i >= 0; i--)
          { int bit = (num >> i) & 1;
            if(!node -> containsKey(bit))
            { node -> put(bit, new Node()); }
            node = node -> get(bit);
          }
        }
    public:
        int findMax(int num)
        { Node *node = root;
```

```
class Trie {
    root
    insert()
    getMax(x)
}
```

T.C. → O(N×32) + O(M×32)
arr1 / go amid arr2

@Ashish Kumar Nayak

```
int maxNum = 0;
for(int i = 31; i>=0; i--)
{ int bit = (num >> i) & 1;
  if(node -> containsKey(!bit))
  {
    maxNum = maxNum | (1<<i);
    node = node -> get(!bit);
  }
  else
  {
    node = node -> get(bit);
  }
}
return maxNum;
}};
int maxXOR(int n, int m, vector<int>&arr1, vector<int>&arr2)
{ Trie trie;
  for(int i=0; i<n; i++)
  { trie.insert(arr1[i]);
  }
  int maxi = 0;
  for(int i=0; i<m; i++)
  { maxi = max(maxi, trie.findMax(arr2[i]));
  }
  return maxi;
}
int main(){
  vector<int> arr1 = {6, 8};
  vector<int> arr2 = {7, 8, 2};
  int n = 2, m = 3;
  cout<<maxXOR(n, m, arr1, arr2)<<endl;
  return 0;
}
```

@Aashish Kumar Nayak

# Maximum XOR with an Element From Array/ queries

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

Queries: `1 3` ——— XOR 1 ^ ( no. in array which is lesser than 3) and return max.

```
5 6
1 1
1
1 0
```

brute force

```
for ( i=0 → q-1)
{ xi = queries[i][0];
  ai = queries[i][1];
  Max_XOR = -1;
for(j=0; j<n; j++)
  { if (arr[j] <= a[i])
      maxXor = max (max_XOR, arr[j] ^ xi)
  }
ans -. pb.max_XOR).
```

Trie Method — initially for the array.

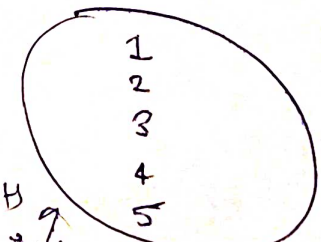
arr[] → {1, 3, 2, 5, 4}

{3, 1}, 0
{5, 2}, 1
{2, 5}, 2
{3, 1}, 3

0 0 0 0 →

less than 1 put in trie

Trie →




Now it gets converted into previous problem.

get max(2)

@Aashish Kumar Nayak

Code :-

```
struct Node{
    Node *links[2];
    bool containsKey(int ind)
    { return (links[ind] != NULL);
    }
    Node* get(int ind)
    { return links[ind];
    }
    void put(int, ind, Node* node)
    { links[ind] = node;
    }
};
class Trie{
    private : Node* root;
    public :
        Trie()
        { root = new Node();
        }
public:
    void insert(int num){
        { Node *node = root;
        for(int i=31; i>=0; i--)
            { int bit = (num >> i) & 1;
            if(!node->containsKey(bit))
            { node -> put(bit, new Node());
            }
        node = node -> get(bit);
        }
    }
}
```

```
public:
    int findmax(int num)
    { Node * node = root;
      int maxNum = 0;
      for(int i = 31; i >= 0; i--)
        { int bit = (num >> i) & 1;
          if(node->containskey(!bit))
            { maxNum = maxNum | (1<<i);
              node = node->get(!bit);
            }
          else {
                 node = node->get(bit);
               }
        }
      return maxNum;
    }
};
vector<int> maxXorQueries(vector<int>& arr, vector<vector<int>>& queries)
{ vector<int> ans(queries.size(), 0);
  ~~vector<pair<int,pair>>~~
  vector<pair<int, pair<int, int>>> offlineQueries;
  Sort(arr.begin(), arr.end());
  int i = 0;
  int n = arr.size();
  Trie trie;
  for(auto &it : offlineQueries)
  { while(i<n && arr[i] <= it.first)
    { trie.insert(arr[i]);
      i++;
    }
    if(i!=0) ans[it.second.second] = trie.findmax(it.second.first);
    else ans[it.second.second] = -1;
  }
  return ans;
}
```

@Aashish Kumar Nayak