XOR problems

1. Max XOR Pair - I
2. Max XOR Pair - II
3. XOR Pairs in Range
4. Subarrays with XOR < K
5. Unique rows in Boolean matrix

Bit Manipulation Basics

- set, get, unset, toggle
- leftshift, rightshift
- XOR properties

★ Bit manipulation basics

Integer: 4 bytes ⇒ 32 bits

Eg: 9 ⇒ 1001 → 00000000000000000000000000001001

7 ⇒ 111 → 0111

5 ⇒ 101 → 0101

3 ⇒ 11 → 0011

1 ⇒ 1 → 0001

for 9 → 1001 → $1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0$

⇒ 8 + 0 + 0 + 1

⇒ 9

get bit → get 3rd bit

Approach ① → left shift → 0001 → 1<<2 → 0100 → 4

$1\times2^2 = 4$

$x << y \equiv x \times 2^y$

Approach ② → right shift → 0100 → 4>>2 → 0001 → 1

$4\times2^{-2} = 1$

$x >> y \equiv x \div 2^y$

Get: to find the value of any kth bit in binary representation of a number

(Assume 0-base R to L)

① Using Left shift

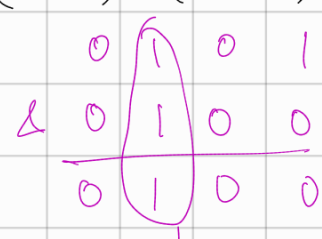
no(x) = 5 → 0 1 0 1

k = 2

(x) & (1 << k)

(0101) & (0001 << 2)

(0101) & (0100)



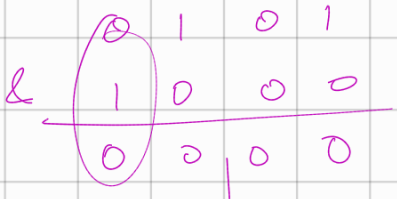
1 → !=0 ✓

k = 3

(x) & (1 << k)

(0101) & (0001 << 3)

(0101) & (1000)



0 → !=0 ✗

② Using Right Shift

no(x) = 5 → 0 1 0 1

k = 2

(x >> k) & 1

(0101 >> 2) & (0001)

(0001) & (0001)

1 → !=0 → ✓

k = 3

(x >> k) & 1

(0101 >> 3) & (0001)

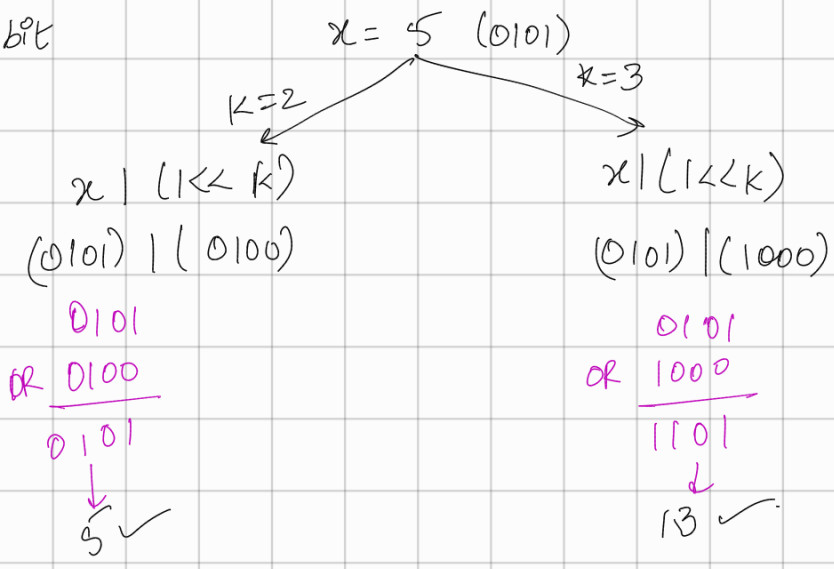
(0000) & (0001)

0 → !=0 → ✗

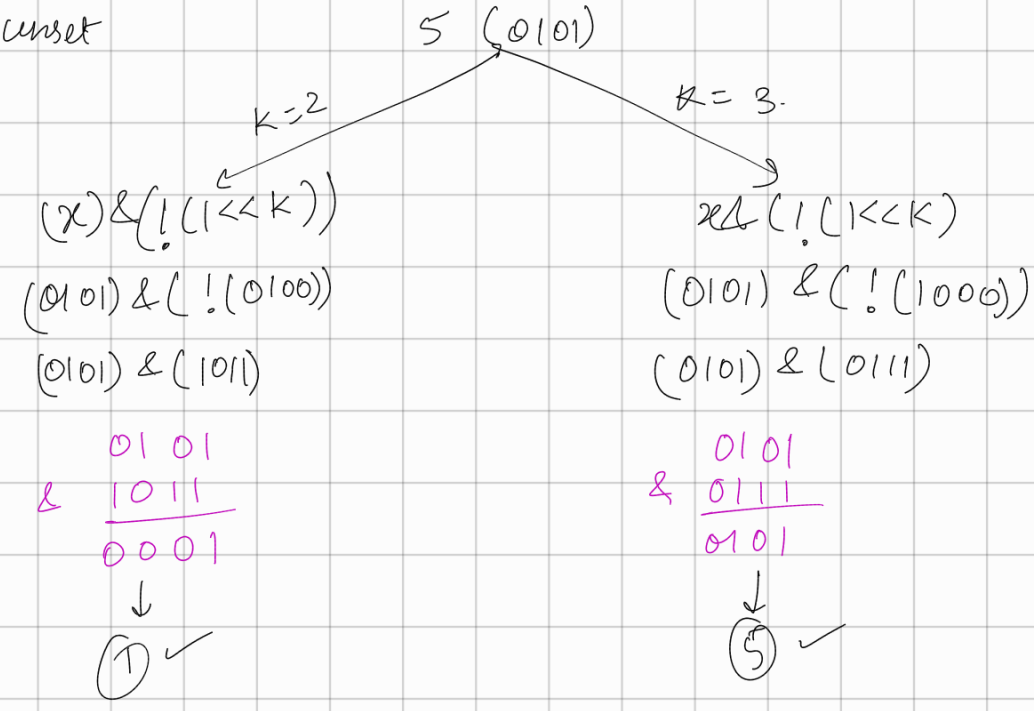
So, both left shift and right shift can be used for get Bit

* Check for !=0 & not for ==1 *

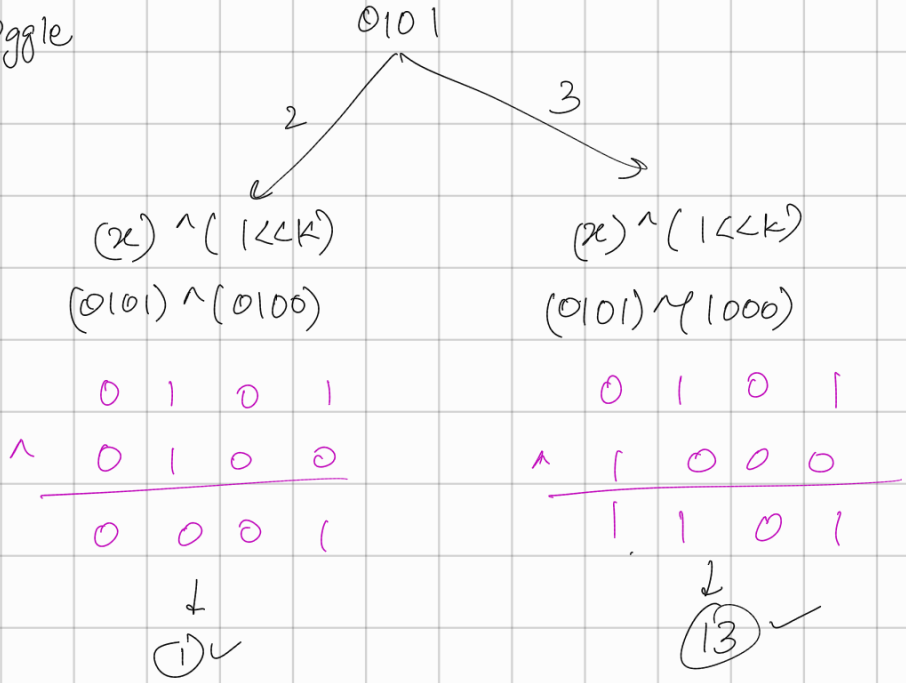
Set bit



unset



Toggle



XOR properties.

① Anything XOR 0 (zero) = The Same Thing

② Something XOR Something = 0

③ $A \wedge B = C \iff A \wedge C = B$

④ $A \wedge B = B \wedge A$

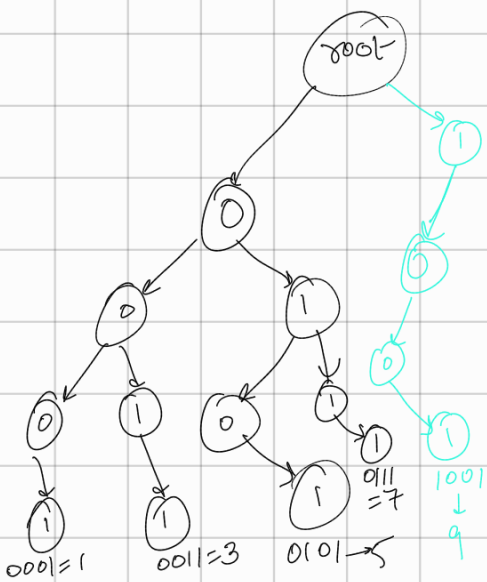
① Max XOR Pair I :

Eg: 5 (0101), 9 (1001), 1 (0001), 3 (0011), 7 (0111)

5^5=0	9^9=0	1^1=0	3^3=0	7^7=0
5^9=1100=12	9^1=8	1^3=2	3^7=4	
5^1=0100=4	9^3=11	1^7=6		
5^3=0110=6	9^7=14			
5^7=0010=2				

for all elements → int → O(32)
↳ long → O(64)

Approach, make tree of 32 bits → 0 = left child
1 = right child

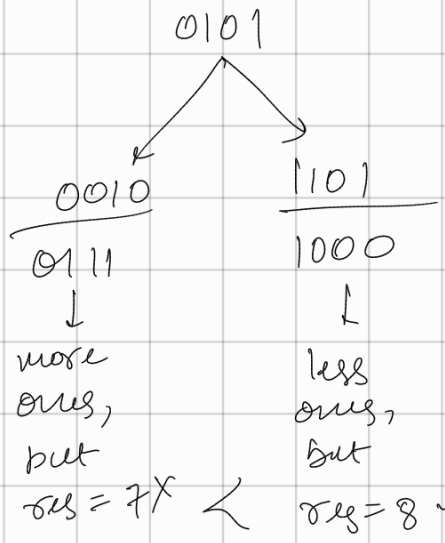


for 0101
→ 1010 XOR
1111

Ideally

But if ideal answer does not exist, we want the closest to ideal from left side / MSB side

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



for 5 = 0101

0 → ideal → 1, ✓ → 1
1 → ideal → 0, ✓ → 0
0 → ideal → 1, ✗ → 0
1 → ideal → 0, ✗ → 0

5 XOR g

ans = 1101 = 13

for g = 1001

1 → ideally = 0, ✓ → 0
0 → ideally = 1, ✓ → 1
0 → ideally = 1, ✓ → 1
1 → ideally = 0, ✗ → 1

g XOR 7 = 1110 = 14 ✓

TC: for each number 32 bits operation is done twice (insert, check)

∴ TC = O(32 × N) = O(N)

Optimization: Insert & Check at the same time, maybe 9^7 will be missed but still 7^9 will give right answer

CODE:

```
public static class Node {
    Node left, right;
}
static Node root;
public void insert(Node root, int val){
    for(int i=31; i>=0; i--){
        int bit = val & (1<<i);
        if(bit==0){
            if(root.left == null)
                root.left= new Node();
            root= root.left;
        }else{
            if(root.right== null)
                root.right= new Node();
            root= root.right;
        }
    }
}

public int search(Node root, int val){
    int maxXor=0;
    for(int i=31; i>=0; i--){
        int bit= val &(1<<i);
        if(bit ==0){
            if(root.right != null){
                root= root.right;
                maxXor= maxXor | (1<<i);
            }else{
                root = root.left;
            }
```

```
        } else {
            if (root.left != null) {
                root = root.left;
                maxXor = maxXor | (1<<i);
            } else {
                root = root.right;
            }
        }
    }
    return maxXor;
}

public int findMaximumXOR(int[] nums) {
    Node root = new Node();
    int ans = 0;
    for (int val : nums) {
        insert(root, val);
        ans = Math.max(ans, search(root, val));
    }
    return ans;
}
```

② Unique rows in boolean matrix.

No. of unique rows = No. of nodes in the deepest level of trie

③ Maximum XOR with an Element from an Array

Eg: [5, 9, 1, 3, 7]

queries
{6, 7}
{4, 5}
{8, 9}
{7, 3}

6 ⇒ 0110 , 7 ⇒ 0111

App search ① Search (Trie except nodes 7, 6)

for 6 = 0110
Ideal 1001
but 1001 > 0111 ∉ 7
So
Ideal 2 0001 < 0111 ∈ 7

→ This needs offline queries.

Online queries ⇒ applying queries in given order.
Offline queries ⇒ apply queries in order of least deletions.
↳ or apply queries when values smaller than query limits have been inserted

⇒ Sort Queries wrt second element.

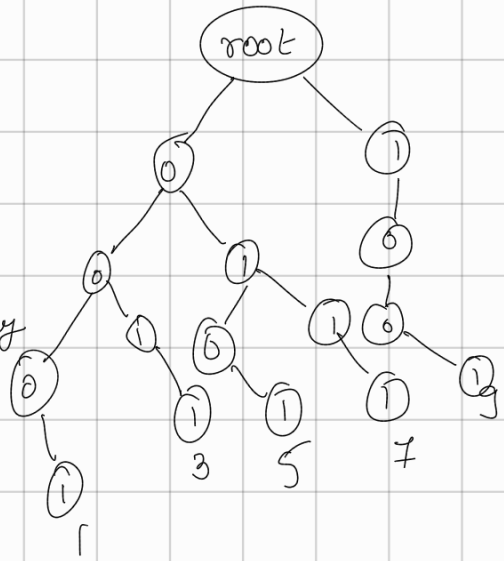
{7, 3} {4, 5} {6, 7} {8, 9}
idx 3 1 0 2

answers {1, 3, 5, 7, 9}

before {7, 3}
insert {1, 3}
search (7) ∵ trie has ≤ 3 only

before {4, 5}
insert (5)
search (4)

before {6, 7}
insert (7)



search (6)

before [8, 9]

insert (9)

search (8).