Maximum XOR of Two Numbers in an Array

Given a **non-empty** array of numbers, $a_0, a_1, a_2, \ldots, a_{n-1}$, where $0 \le a_i < 2^{31}$.

Find the maximum result of a_i XOR a_j, where $0 \le i, j < n$.

Could you do this in $O(n)$ runtime?

Example:

```
Input: [3, 10, 5, 25, 2, 8]

Output: 28

Explanation: The maximum result is 5 ^ 25 = 28.
```

Solution 1

```
public class Solution {
    public int findMaximumXOR(int[] nums) {
        int max = 0, mask = 0;
        for(int i = 31; i >= 0; i--){
            mask = mask | (1 << i);
            Set<Integer> set = new HashSet<>();
            for(int num : nums){
                set.add(num & mask);
            }
            int tmp = max | (1 << i);
            for(int prefix : set){
                if(set.contains(tmp ^ prefix)) {
                    max = tmp;
                    break;
                }
            }
        }
        return max;
    }
}
```

written by tangx668 original link here

Solution 2

```
def findMaximumXOR(self, nums):
    answer = 0
    for i in range(32)[::-1]:
        answer <<= 1
        prefixes = {num >> i for num in nums}
        answer += any(answer^1 ^ p in prefixes for p in prefixes)
    return answer
```

Build the answer bit by bit from left to right. Let's say we already know the largest first seven bits we can create. How to find the largest first eight bits we can create? Well it's that maximal seven-bits prefix followed by 0 or 1. Append 0 and then try to create the 1 one (i.e., `answer ^ 1`) from two eight-bits prefixes from `nums`. If we can, then change that 0 to 1.

written by StefanPochmann original link here

Solution 3

```
class Trie {
    Trie[] children;
    public Trie() {
        children = new Trie[2];
    }
}

public int findMaximumXOR(int[] nums) {
    if(nums == null || nums.length == 0) {
        return 0;
    }
    // Init Trie.
    Trie root = new Trie();
    for(int num: nums) {
        Trie curNode = root;
        for(int i = 31; i >= 0; i --) {
            int curBit = (num >>> i) & 1;
            if(curNode.children[curBit] == null) {
                curNode.children[curBit] = new Trie();
            }
            curNode = curNode.children[curBit];
        }
    }
    int max = Integer.MIN_VALUE;
    for(int num: nums) {
        Trie curNode = root;
        int curSum = 0;
        for(int i = 31; i >= 0; i --) {
            int curBit = (num >>> i) & 1;
            if(curNode.children[curBit ^ 1] != null) {
                curSum += (1 << i);
                curNode = curNode.children[curBit ^ 1];
            }else {
                curNode = curNode.children[curBit];
            }
        }
        max = Math.max(curSum, max);
    }
    return max;
}
```

written by mywen1234 original link here