Maximum XOR For Each Query

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	1829
tags	Bit Manipulation
language	C++
solved on	@08/11/2024
link	https://leetcode.com/problems/maximum-xor-for-each-query/description/
Completion	☑

Intuition

The problem revolves around calculating the maximum XOR value for each prefix of the given list of integers, where the XOR is computed with a mask that keeps the bits within a specified bit range (`maximumBit`). The result for each position is the maximum XOR value that can be obtained using the given numbers with respect to the bit limit.

- The approach focuses on maintaining a running XOR (`XOR`) of the elements in the array.
- The mask is used to limit the size of the XOR result to fit within the given `maximumBit`.
- The key insight is that for each element, the XOR operation flips bits, and by taking the complement (`~XOR`), the goal is to maximize the XOR value.

Approach

1. **Mask Calculation**: First, calculate a mask based on the given `maximumBit`. The mask ensures that we only consider the lower `maximumBit` bits when calculating the XOR result.
2. **Iterate Over the Array**:
   - Maintain a running XOR of the array's elements.
   - For each element in the array, the result for that element is `(~XOR) & mask`. The result is stored in the corresponding position, starting from the end of the array.
3. **Why Reverse the Iteration**: The result is filled in reverse order because, for each position `i`, the maximum XOR up to that position needs to be calculated. Since the last element's result depends on all previous ones, iterating backward ensures we respect the prefix XOR.
4. **Return the Answer**: After populating the results in reverse, return the list.

Complexity

Time Complexity:

- The time complexity is **O(n)**, where `n` is the size of the input array `nums`. The loop runs once over the entire array, performing constant-time operations like XOR and bitwise operations.

Space Complexity:

- The space complexity is **O(n)** due to the `answer` vector storing the results. No additional significant space is used beyond this.

Code

```
class Solution {
public:
    vector<int> getMaximumXor(vector<int>& nums, int maximumBit) {
        vector<int> answer(nums.size());
        int mask = (1 << maximumBit) - 1;

        int XOR = 0;

        for (int i = 0; i < nums.size(); i++) {
            XOR ^= nums[i];
            answer[nums.size() - 1 - i] = (~XOR) & mask;
        }

        return answer;
    }
};
```