810. Chalkboard XOR Game

Hard | Bit Manipulation | Brainteaser | Array | Math | Game Theory

Problem Description

In this problem, we have a game based on numbers written on a chalkboard represented by an array of integers `nums`. There are two players, Alice and Bob, who take turns erasing exactly one number from the chalkboard. Alice always goes first. The goal is to avoid making the bitwise XOR of all elements on the chalkboard equal to `0`. If a player erases a number and the XOR turns to `0`, that player loses. Bitwise XOR of one element is the element itself, while the bitwise XOR of no elements is `0`.

Additionally, there is a win condition right at the start of a player's turn. If the bitwise XOR of all elements before the player takes an action is `0`, then that player immediately wins the game.

The task is to return `true` if Alice wins the game when both players play optimally, otherwise `false`.

Intuition

To solve this problem, we need to analyze the conditions under which a player can win, keeping in mind that both Alice and Bob play optimally (they make the best possible moves).

Firstly, one straightforward condition is if the initial XOR of all numbers is `0`. If Alice begins and the XOR is `0`, she wins without making a move.

Secondly, consider the condition regarding the count of numbers on the chalkboard. If the count is even, Alice can always mirror Bob's moves. Whenever Bob erases a number to avoid losing, Alice can do the same. Since the count began even and decreases by two with each full round (Alice and Bob each erase one), Alice will never be forced into a losing move because she will always respond to Bob's moves and the number of elements will stay even on her turn.

The optimal strategy for both players is to always erase a number that keeps the xor sum non-zero if possible. If not, the player losing is the one forced to make the xor sum `0`. Hence, Alice will only lose if she starts with an odd number of elements on the chalkboard and cannot make the XOR sum `0` on her first move.

Solution Approach

The solution approach utilizes a Python function `xorGame`, which determines the winner of the game using two simple checks based on the rules of the game as described in the problem description.

To implement the solution, we use two Python features:

1. The `reduce` function from the `functools` module.
2. The `xor` function from the `operator` module.

Here is how the solution approach works in terms of these components and logic:

- The `xor` function is a binary operation that performs bit-by-bit exclusive OR on two integers. It is used in conjunction with `reduce` to calculate the cumulative XOR of all elements in the `nums` list.
- `reduce(function, sequence)` applies the `function` cumulatively to the items of the `sequence`, from left to right, so as to reduce the sequence to a single value. For example, `reduce(xor, nums)` applies the `xor` function to the elements of `nums` and gives a single integer result, which is the XOR of all numbers.
- The solution then relies on two conditions:
  - `len(nums) % 2 == 0`: As stated in the intuition, if the number of elements in `nums` is even, Alice can always ensure she won't be the one to cause the XOR sum to become `0`. She can do this by mirroring Bob's moves. Therefore, if the length of the list `nums` is even, the function returns `true` indicating Alice wins.
  - `reduce(xor, nums) == 0`: This condition checks if the initial XOR of all elements is `0`. If it is, Alice wins by the game's rules without having to erase any number, so the function returns `true`.

The `xorGame` function in Python combines these checks with the `or` operator. If either condition is `true`, Alice wins the game; otherwise, Bob does.

Here's the implementation distilled into its logic:

1. Check if the length of `nums` is even; if so, return `true` and Alice wins.
2. Calculate the XOR of all elements in `nums`; if it's `0`, return `true` and Alice wins.
3. If both checks fail, return `false`, indicating Bob wins.

This implementation is short and efficient. It directly applies mathematical properties of the XOR operation and parity checks (even or odd number of elements) in determining the winner of the game.

The provided implementation also assumes optimal play from both participants, meaning they will both do their best to avoid losing. Considering this, the checks cover all scenarios for Alice to win.

Example Walkthrough

Let's use a small example to illustrate the solution approach described above. Suppose we have the following array of integers, which represents the numbers on the chalkboard: `nums = [1, 1, 2]`.

Now, let's walk through the logic:

1. First, we count the number of elements in `nums`. There are three elements, so the count is odd. This means that we can't immediately decide Alice wins based on the even-count condition.
2. Next, we calculate the XOR of all elements: `1 XOR 1 XOR 2`. In binary, this is `01 XOR 01 XOR 10`. The XOR of the first two `1s` cancels each other out, leaving us with `0 XOR 2`, which equals `2`. The result is non-zero, so we now know that the game will not be decided by the initial XOR being `0`.

Because both conditions for Alice to win (an even number of elements and an initial XOR of `0`) are not met, the solution approach will return `false`. This indicates that, under optimal play, Bob will have the winning strategy in this scenario.

Solution Implementation

Python

```python
from functools import reduce
from operator import xor
from typing import List

class Solution:
    def xorGame(self, nums: List[int]) -> bool:
        # The xor game can be won in two scenarios:
        # 1. If the length of the list nums is even, the player starting can always win
        # by pairing up the numbers and xoring each pair.
        if len(nums) % 2 == 0:
            return True
        # 2. If the xor of all numbers in the list nums is 0, the player can also win,
        # because no matter how the other player picks a number, the first player can
        # always choose a number that will reset the xor back to 0.
        else:
            # reduce applies the xor operator cumulatively to the items of nums,
            # from left to right, so as to reduce the iterable nums to a single value.
            return reduce(xor, nums) == 0

# Examole usage:
# sol = Solution()
# print(sol.xor_game([1, 1, 2]))  # Outputs: True, because len(nums) is odd but xor of all numbers is 0
```

Java

```java
class Solution {

    // Method to check if the XOR game can be won.
    public boolean xorGame(int[] nums) {
        // Check for two conditions:
        // 1. If the length of the array is even, the player starting the game can always win.
        // 2. If the XOR of all elements in the array is 0, the player can also win.
        return isEvenLength(nums) || isXorZero(nums);
    }

    // Helper method to check if the length of the array is even.
    private boolean isEvenLength(int[] nums) {
        return nums.length % 2 == 0;
    }

    // Helper method to calculate the XOR of all elements in the array and check if it is 0.
    private boolean isXorZero(int[] nums) {
        int xorResult = Arrays.stream(nums).reduce(0, (a, b) -> a ^ b);
        return xorResult == 0;
    }
}
```

C++

```cpp
#include <vector> // Include the necessary header for vector

class Solution {
public:
    // Function to determine if the player starting the XOR game will always win
    bool xorGame(const std::vector<int>& nums) {
        // If there's an even number of elements, the first player will always win
        if (nums.size() % 2 == 0) {
            return true;
        }

        // Calculate the XOR of all elements in the vector
        int xorSum = 0;
        for (int num : nums) {
            xorSum ^= num;
        }

        // The first player also wins if the XOR sum of all numbers is 0
        return xorSum == 0;
    }
};
```

TypeScript

```typescript
// Function to determine if the player starting the XOR game will always win
function xorGame(nums: number[]): boolean {
    // If there's an even number of elements, the first player will always win
    if (nums.length % 2 === 0) {
        return true;
    }

    // Calculate the XOR of all elements in the array
    let xorSum: number = 0;
    for (let num of nums) {
        xorSum ^= num;
    }

    // The first player also wins if the XOR sum of all numbers is 0
    return xorSum === 0;
}
```

```python
from functools import reduce
from operator import xor
from typing import List

class Solution:
    def xorGame(self, nums: List[int]) -> bool:
        # The xor game can be won in two scenarios:
        # 1. If the length of the list nums is even, the player starting can always win
        # by pairing up the numbers and xoring each pair.
        if len(nums) % 2 == 0:
            return True
        # 2. If the xor of all numbers in the list nums is 0, the player can also win,
        # because no matter how the other player picks a number, the first player can
        # always choose a number that will reset the xor back to 0.
        else:
            # reduce applies the xor operator cumulatively to the items of nums,
            # from left to right, so as to reduce the iterable nums to a single value.
            return reduce(xor, nums) == 0

# Examole usage:
# sol = Solution()
# print(sol.xor_game([1, 1, 2]))  # Outputs: True, because len(nums) is odd but xor of all numbers is 0
```

Time and Space Complexity

The time complexity of the function `xorGame` is `O(N)`, where `N` is the number of elements in the list `nums`. This is because the function computes the xor of all the elements in the list using the `reduce` function along with the xor operator, which takes linear time.

The space complexity of the function is `O(1)`. No additional space is used proportional to the size of the input, as the xor operation is done in-place and the comparison with zero also uses constant space.