292. Nim Game

Easy Brainteaser Math Game Theory Leetcode Link

Problem Description

In the Nim Game, you are up against a friend with a pile of stones on the table. Players take turns, with you in the lead, each turn allowing the removal of 1 to 3 stones from the pile. The main objective is to be the person who takes the last stone, as this means victory. The challenge posed to you is a strategy one: given the total number `n` of stones, can you determine whether success is in your reach assuming both players adopt an optimal playing strategy? Your task is to compute a function that inputs the number of stones `n` and outputs `true` if winning is possible, or `false` otherwise.

Intuition

To intuitively reason why the provided solution works, let's break down the game for different initial amounts of stones:

- If `n` is between 1 and 3, you can win by taking all the stones.
- If `n` is 4, no matter how many you take, your friend can always take the remainder and win.
- If `n` is between 5 and 7, you can always take enough stones to leave your friend with 4 stones, which, as stated above, leads to a guaranteed win for you.
- If `n` is 8, any move you make will leave a number between 5 and 7 for your friend, which allows them to leave you with 4 stones, leading to their win.

Noticing this pattern, we can see that if `n` is a multiple of 4, you will inevitably lose, because no matter how many stones you take, your friend can always take an amount that will leave a multiple of 4 for your next turn, putting you at a disadvantage until the end when you are left with the fatal 4 stones.

Therefore, the key observation is that the number 4 is a "losing" number. If the starting number `n` is not a multiple of 4, you can win by taking enough stones to bring the total down to a multiple of 4. After that, you can mirror your friend's moves to ensure you always leave them with a multiple of 4 stones, thereby securing your victory.

This boils down the solution to a simple modulo operation. If `n % 4 != 0`, you can win Nim Game by making the first move that leaves your friend with a multiple of 4 stones; otherwise, you cannot guarantee a win if both of you play optimally.

Solution Approach

The implementation of the solution for the Nim Game problem is succinct and relies on an arithmetic operation known as the modulo operation. The entire solution is encapsulated in a single line of code within the method `canWinNim`, which takes an integer `n` as an argument and returns a boolean value.

Here's how the solution method is implemented:

```python
class Solution:
    def canWinNim(self, n: int) -> bool:
        return n % 4 != 0
```

The algorithm behind this approach is straightforward: since we've established that 4 is the "losing" number for the game (i.e., if you start your turn with 4 stones, you cannot win if your opponent plays optimally), we want to check if the current number of stones `n` is a losing number. We do this by calculating `n % 4` — the remainder when `n` is divided by 4.

If the remainder is not equal to zero (`n % 4 != 0`), this means that `n` is not a multiple of 4, and therefore, you can make the first move to leave a multiple of 4 stones to your opponent. Consequently, you have a winning strategy.

No additional data structures or complex patterns are needed; the modulo operation is the sole driver of this logic, evidencing the elegance and efficiency of the solution - a true demonstration of understanding the problem's inherent mathematical pattern, which simplifies the algorithm to a single, decisive operation that runs in constant time with constant space complexity.

In summary, whenever confronted with this particular instance of the Nim Game:

- If `n % 4 == 0`, this signals your defeat - hence, return `false`.
- If `n % 4 != 0`, victory is in your grasp if you use the correct strategy, making the appropriate removals on each turn - hence, return `true`.

Example Walkthrough

Let's assume we have a pile of 6 stones (`n = 6`). Using the solution approach described, let's walk through the game:

1. Check the total number of stones using the modulo operation: `n % 4`.
2. If `n % 4 != 0`, that means you have a winning strategy.
3. In our case, `6 % 4` equals `2`, which is not equal to zero. Therefore, you can win.
4. To implement the winning strategy, you need to take stones in such a way that you leave a multiple of 4 stones for your opponent. Since `6` is not a multiple of 4, you should take `2` stones first.
5. Now there are 4 stones left, which is a multiple of 4. This is the "losing" number for whoever's turn it is to play.
6. Your opponent can take 1, 2, or 3 stones, but no matter what they take, you will be able to take the remainder and win the game.
7. Your opponent takes 1 stone (best move for them), leaving 3 stones.
8. You take all 3 remaining stones, winning the game.

According to the solution approach, as long as you start with a number that is not a multiple of 4, you can always force a win. If the number had been 8 (which is a multiple of 4), you would be at a losing position assuming optimal play from your opponent. This walkthrough illustrates the simplicity and effectiveness of the modulo-based solution where the key to winning is avoiding leaving a pile of 4 stones for your turn.

Python Solution

```python
class Solution:
    def canWinNim(self, n: int) -> bool:
        # In the game of Nim, you can win if the number of stones 'n'
        # is not a multiple of 4. If 'n' % 4 == 0, your opponent can
        # always play optimally and leave you in a position where you'll
        # eventually lose. Therefore, you can only win if 'n' % 4 != 0.
        return n % 4 != 0

```

Java Solution

```java
class Solution {
    // Method to determine if you can win the Nim game given the number of stones.
    public boolean canWinNim(int n) {
        // In the game of Nim, you can always win if the number of stones
        // is not a multiple of 4. If it is a multiple of 4, you will
        // inevitably lose if your opponent plays optimally, because no matter
        // how many stones you take (1 to 3), your opponent can always take
        // a number that sums up to 4 with your move, eventually leaving you
        // with the last 4 stones, which you will be forced to take.
        return n % 4 != 0;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Determines if the player can win the Nim game given the number of stones.
    // The player can win the game unless the number of stones is a multiple of four.
    bool canWinNim(int numberOfStones) {
        // If the remainder of 'numberOfStones' divided by 4 is not zero,
        // the player can win by making a strategic move.
        return numberOfStones % 4 != 0;
    }
};

```

Typescript Solution

```typescript
/**
 * Determines if you can win the game of Nim given the total number of stones.
 * The player can win if the number of stones is not a multiple of 4.
 *
 * @param {number} totalStones - The total number of stones in the game.
 * @returns {boolean} - True if the player can win, false otherwise.
 */
function canWinNim(totalStones: number): boolean {
    // A player can win if the remaining stones are not a multiple of 4.
    // If the number of stones is a multiple of 4, no matter how the player plays,
    // the opponent can always leave a multiple of 4 stones after their turn,
    // eventually leaving the player with exactly 4 stones on their turn, which
    // forces them to lose. Therefore, the player can only win if the starting
    // number of stones is not a multiple of 4.
    return totalStones % 4 !== 0;
}

```

Time and Space Complexity

The time complexity of the code is `O(1)` as the operation performed is a single modulus calculation, which takes constant time regardless of the value of `n`.

The space complexity of the code is also `O(1)` because the algorithm only uses a fixed amount of space; no additional space is utilized that is dependent on the input size `n`.