2396. Strictly Palindromic Number

Medium Brainteaser Math Two Pointers Leetcode Link

Problem Description

An integer is classified as **strictly palindromic** if it adheres to a special rule regarding its representation in different number bases. Specifically, a number `n` is strictly palindromic if, for every base `b` starting from `2` up to `n - 2`, its representation in that base is a palindrome. It's important to note that a palindrome reads the same forward and backward. For instance, the string "121" is a palindrome because it remains "121" even when reversed.

The task is to create a function that takes an integer `n` and returns `true` if `n` is strictly palindromic or `false` otherwise. This means the function needs to verify if `n` is a palindrome in all the bases from `2` to `n - 2`. The challenge is to figure out a way to evaluate this condition effectively or to determine if such a number can exist.

Intuition

Upon initial consideration, you might think that to determine if a number is strictly palindromic, you'd need to convert the number into all the base representations from `2` to `n-2` and check if it's a palindrome in each. This would indeed be the brute-force approach.

However, a closer examination of the problem can lead to the conclusion that such a number cannot exist. Intuitively, the notion of a strictly palindromic number is flawed because when a number `n` is represented in base `n-1`, the result would always be `11` (since `n` base `n-1` is the same as saying `n-1` times `1` with a remainder of `1`, which is how base conversion works). This representation of the number `n` as `11` in base `n-1` is palindromic, but that is just for this single base.

When you get to base `n-2`, the hypothesis falls apart. Regardless of what the number `n` is, it cannot continue to be palindromic in each and every base leading down to 2 from `n-2`. This is due to the way numbers scale in different bases, which inevitably leads to different least significant digits as you go down in base, disrupting the palindromic nature.

Given these considerations, it turns out that the solution to the problem is quite simple: no number greater than `3` can satisfy the condition of being strictly palindromic because it will fail the test when `n` is expressed in base `n-2`. The function, therefore, correctly always returns `False`, simplifying the solution significantly.

Solution Approach

The implementation of the solution does not require complex algorithms, data structures, or patterns. It is based on the mathematical proof that there are no strictly palindromic numbers beyond `3` due to the nature of base conversion, particularly when `n` is represented in base `n - 2`.

Exploring different bases:

- When converting a number to a base that is one less than the number itself (base `n - 1`), the number is always represented as `11`. This is palindromic, but it only works for that specific base.
- Reducing the base further to `n - 2`, the representation changes and cannot maintain the palindromic property across all lower bases down to `2`.

Given this understanding, the reference solution's approach is straightforward:

The `isStrictlyPalindromic` function in Python is defined with a single line of code:

```python
class Solution:
    def isStrictlyPalindromic(self, n: int) -> bool:
        return False
```

This function does not require any loops, checks, or base conversions because it leverages the knowledge that no number `n > 3` can ever be strictly palindromic. Thus, the solution is exceptionally efficient with a constant time complexity of O(1) - it will always return `False` immediately, regardless of the value of `n`.

Example Walkthrough

Let's pick an integer `n` and walk through the solution approach using this example. Suppose we choose `n = 10`. We need to determine if 10 is strictly palindromic, which means it must be a palindrome from base `2` up to base `n - 2`, which would be base `8` in this case.

- For base `2`, the representation of 10 is `1010`, which is not palindromic.
- For base `3` to base `7`, the representation will change for each base, and while we might find it palindromic in one or a few bases, that is irrelevant because we need it to be palindromic in all bases from `2` to `n - 2`.
- For base `n - 1`, which is base `9` for our example, the representation of `10` would be `11`, as `10` in base `9` equals `9 + 1`. This is indeed palindromic but again does not fulfill the strict condition required.
- Finally, for base `n - 2`, which is base `8` in our case, the representation of `10` would no longer be `11`. In base `8`, `10` is equal to `12`, because `10` in base `8` equals `8 + 2`. This violates the condition to be strictly palindromic as it is not the same forward and backward.

From this example, we can observe that as we change the base, the representation of the number changes and does not remain palindromic in all bases leading up to `n - 2`. Hence, by similar logic, we can generalize that for any number `n > 3`, it will not remain strictly palindromic across all the bases from `2` to `n - 2`. The solution below reflects this understanding:

```python
class Solution:
    def isStrictlyPalindromic(self, n: int) -> bool:
        return False
```

In summary, our example demonstrates that regardless of the number `n` we choose (provided `n > 3`), it will not be possible for it to be strictly palindromic. Hence, the function always returns `False`, as established by the solution approach detailed in the content provided.

Python Solution

```python
class Solution:
    def isStrictlyPalindromic(self, n: int) -> bool:
        # Explanation:
        # The following method checks if the given number 'n' is strictly palindromic.
        # However, upon review, it is found that this method simply returns False for any input.
        # The term "strictly palindromic" is not well-defined in this context,
        # because it does not check the number 'n' in any base to determine whether it is a palindrome.
        # It is worth noting that by the traditional definition, no number can be strictly palindromic,
        # as it would have to be a palindrome in every base, which is not possible for integers greater than 0.
        # Thus, this function always returns False, suggesting that no number can be 'strictly' palindromic according to the unwritte

        return False

```

Java Solution

```java
class Solution {

    // This method checks if an integer n is "strictly palindromic"
    // According to the current definition provided, it returns false for all inputs
    public boolean isStrictlyPalindromic(int n) {
        // Always returns false as no integer can be strictly palindromic by the problem's original statement
        return false;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to determine if a number is strictly palindromic
    // Since the problem states to convert the number into all bases
    // between 2 and n - 2 and check if they are palindromes, and no number
    // can satisfy this condition, the function always returns false.
    bool isStrictlyPalindromic(int n) {
        // The implementation of the logic that eventually leads to the conclusion that
        // no number can satisfy the strictly palindromic condition is omitted.
        // Therefore, the function simply returns false.
        return false;
    }
};

```

Typescript Solution

```typescript
/**
 * Checks if a number is strictly palindromic.
 * A number is strictly palindromic if it forms a palindrome in every base from 2 up to n - 1
 *
 * @param {number} n - The number to check.
 * @returns {boolean} - A boolean indicating if the number is strictly palindromic or not.
 */
function isStrictlyPalindromic(n: number): boolean {
    // Iterate through all the bases from 2 to n - 1
    for (let base = 2; base < n; base++) {
        // Convert the number to the current base
        const numberInBase = n.toString(base);

        // Check if the converted number is a palindrome
        if (!isPalindrome(numberInBase)) {
            return false;
        }
    }

    // If number n is palindromic in all bases from 2 to n - 1, return true
    return true;
}

/**
 * Checks if a string is a palindrome.
 *
 * @param {string} str - The string to check.
 * @returns {boolean} - A boolean indicating whether the string is a palindrome or not.
 */
function isPalindrome(str: string): boolean {
    // Define pointers for the start and end of the string
    let start = 0;
    let end = str.length - 1;

    // Check if the string is a palindrome by comparing characters from both ends
    while (start < end) {
        if (str[start] !== str[end]) {
            return false;
        }
        start++;
        end--;
    }

    // If all characters match, the string is a palindrome
    return true;
}

```

Time and Space Complexity

The given code is a Python method within a class `Solution` that always returns `False` regardless of the input `n`.

Time Complexity: `O(1)`

The time complexity of the method `isStrictlyPalindromic` is constant time `O(1)`, because the function performs a single operation (returning `False`) regardless of the size or value of the input parameter `n`.

Space Complexity: `O(1)`

The space complexity of the method is also constant `O(1)`. No additional space is used that grows with the input size; the only space used is for the input itself and the return value, which do not depend on the value or size of `n`.