Solution: Strobogrammatic Number

Let’s solve the Strobogrammatic Number problem using the Two Pointers pattern.

We'll cover the following

Statement

Given a string `num` representing an integer, determine whether it is a strobogrammatic number. Return TRUE if the number is strobogrammatic or FALSE if it is not.

> **Note:** A **strobogrammatic number** appears the same when rotated 180 degrees (viewed upside down). For example, “69” is strobogrammatic because it looks the same when flipped upside down,

> while “962” is not.

Constraints:

- $1 <= \texttt{num.length} <= 50$
- `num` contains only digits.
- `num` has no leading zeros except when the number itself is zero.

Solution

The solution uses a two pointer approach to determine whether a given string `num` is a strobogrammatic number by checking its digits from both ends toward the center. It uses a set of valid digit mappings that remain unchanged when rotated 180 degrees or transform into each other when flipped (such as $'0'$ to $'0'$, $'1'$ to $'1'$, $'8'$ to $'8'$, $'6'$ to $'9'$, and $'9'$ to $'6'$). The key idea is to verify that each digit on the left side of the string correctly matches its mirrored counterpart on the right side. This means checking if the digit at the start aligns correctly with the corresponding digit at the end when viewed upside down. If all such pairs match according to the defined mappings, the number is considered strobogrammatic.

Now, let’s walk through the steps of the solution:

1. We initialize a map, `dict`, to store the valid mappings of digits that either remain the same or transform correctly when rotated 180 degrees:
    I. `'0'` maps to `'0'`

II. `'1'` maps to `'1'`

III. `'8'` maps to `'8'`

IV. `'6'` maps to `'9'`

V. `'9'` maps to `'6'`

2. We set two pointers:

   I. `left` starts at the beginning of the string.

   II. `right` starts at the end of the string.

3. We iterate from both ends of the string using `left` and `right` pointers toward the middle. In each iteration, we compare the pair of digits pointed by `left` and `right` pointers. For each pair, we do the following:

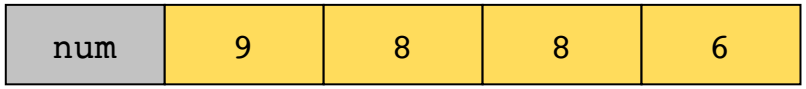
   I. We check whether `num[left]` exists in `dict`. If not, we return FALSE because it is not a valid strobogrammatic digit. Otherwise, we retrieve the expected rotated value of `num[left]` from `dict`. If this expected value does not match `num[right]` pointer, we return FALSE because it means the number is not strobogrammatic.

   II. We increment the `left` pointer by 1 and decrement the `right` pointer by 1, moving toward the center of the string.

4. We keep iterating until the `left` pointer crosses the `right` pointer.

5. After iterating, if all pairs are valid according to the strobogrammatic rules, we return TRUE, indicating tha the number is strobogrammatic.

Let's look at the following illustration to get a better understanding of the solution:



We check whether the number "9886" is strobogrammatic or not.

1 of 12

Let’s look at the code for the solution we just discussed.

Solution | Commented Solution

```
import java.util.HashMap;
import java.util.Map;

public class Solution {

    // Function to check if a number is strobogrammatic
```

```
    public static boolean isStrobogrammatic(String num) {
        Map<Character, Character> dict = new HashMap<>();
        dict.put('0', '0');
        dict.put('1', '1');
        dict.put('8', '8');
        dict.put('6', '9');
        dict.put('9', '6');

        int left = 0;
        int right = num.length() - 1;

        while (left <= right) {
            if (!dict.containsKey(num.charAt(left)) || dict.get(num.charAt(left)) != num.charAt(righ
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    // Driver code
```

Run Save Reset

Strobogrammatic Number

Time complexity

The time complexity of the solution is $O(n)$, where n is the length of the input string `num`. This is because we iterate through the string once, comparing each digit pair from both ends toward the center.

Space complexity

The space complexity is $O(1)$ because the solution uses a fixed-size map to store the strobogrammatic digit mappings, regardless of the input size.

Back lesson

Mark As Completed Next

Strobogrammatic Number

Minimum Number of Moves to Make Palindrome