
Problem Statement:

Determine if a given string is a palindrome by first removing all non-alphanumeric characters, converting the remaining characters to lowercase, and then checking if the cleaned string reads the same forwards and backwards.

Solution 1

```python
def valid_palindrome(s: str) -> bool:
    clean_s = ''.join([char.lower() for char in s if char.isalnum()])
    half_way = len(clean_s) // 2
    for i in range(half_way):
        if clean_s[i] != clean_s[len(clean_s) - 1 - i]: return False
    return True
```

Step-by-Step Breakdown

1. **Input:**
    - A string s that may contain letters, numbers, spaces, punctuation, etc.
2. **Intermittent step 1:** Clean the string:

```
1. clean_s = ''.join([char.lower() for char in s if char.isalnum()])
```

This step removes all non-alphanumeric characters from s and converts the remaining characters to lowercase.

3. **Intermittent step 2:**
    - Determine the midpoint:

```
1. half_way = len(clean_s) // 2
```

Compare characters from each end:
Iterate over the first half of clean_s and, for each index i, compare clean_s[i] with clean_s[len(clean_s) - 1 - i].

- If any pair does not match, return False immediately.

4. **Output:**
    - If all corresponding characters match, return True, indicating that the cleaned string is a palindrome.
5. **Efficiency:**
    - Time Complexity: O(n) — The algorithm iterates through half of the cleaned string.

- Space Complexity: O(n) — A new string (clean_s) is created to store the cleaned version of the input.

Visual Flow Diagram

```
Input String: "A man, a plan, a canal: Panama"
                        |
                        ▼
Remove non-alphanumeric characters and convert to lowercase
                        |
                        ▼
Cleaned String: "amanaplanacanalpanama"
                        |
                        ▼
Calculate half_way = len(clean_s) // 2
                        |
                        ▼
For i in range(half_way): (i = 0 to half_way - 1)
                        |
                        ▼
Compare clean_s[i] with clean_s[len(clean_s) - 1 - i]
        ┌──────────────────────────────────────┐
        │ Example: clean_s[0] vs clean_s[20]   │
        │          clean_s[1] vs clean_s[19]   │
        │          ...                         │
        └──────────────────────────────────────┘
                        |
                        ▼
If any pair does not match → return False
                        |
                        ▼
            If all pairs match → return True
```

Solution 2

```python
def valid_palindrome(s: str) -> bool:
    left, right = 0, len(s) - 1
    while left < right:
        # Move left pointer forward if current character is not alphanumeric.
        while left < right and not s[left].isalnum():
            left += 1
        # Move right pointer backward if current character is not alphanumeric.
        while right > left and not s[right].isalnum():
            right -= 1
        # Compare the characters in a case-insensitive manner.
        if s[left].lower() != s[right].lower():
            return False
        left += 1
        right -= 1
    return True
```

Explanation

- **Two-Pointer Approach:**
  Two indices (left and right) start at the beginning and end of the string, respectively.
- **Skipping Non-Alphanumeric Characters:**
  The inner loops increment left or decrement right if the current character is not alphanumeric. This way, only alphanumeric characters are compared.
- **Case-Insensitive Comparison:**
  The characters at the two pointers are compared in lowercase form to ignore case differences.
- **Efficiency:**
  - **Time Complexity:** O(n) — Each character in the string is examined at most once.
  - **Space Complexity:** O(1) — No extra space is needed for storing a cleaned version of the string.

Visual Flow Diagram