palindrome counter hackerrank solution

Palindrome counter hackerrank solution is a popular algorithm challenge that tests your ability to identify palindromic substrings within a given string. This problem is not only an excellent exercise for budding programmers but also serves as a valuable lesson in understanding string manipulation and dynamic programming concepts. In this article, we will explore the palindrome counter problem in detail, discuss various approaches to solve it, and provide a comprehensive solution that can be implemented on platforms like HackerRank.

Understanding Palindromes

A palindrome is a sequence of characters that reads the same backward as forward. For example, "madam," "racecar," and "level" are all palindromic words. In the context of the palindrome counter problem, we need to identify all possible palindromic substrings within a given string.

Types of Palindromes

1. Odd-length Palindromes: These palindromes have a single character in the center. For example, "aba" and "racecar."
2. Even-length Palindromes: These consist of pairs of characters centered around the two middle characters. For example, "abba" and "deified."

The Palindrome Counter Problem

The palindrome counter problem typically involves the following steps:

1. Input a string: The user is required to enter a string of characters.
2. Count palindromic substrings: The program must compute the total number of palindromic substrings within the input string.
3. Output the count: Finally, the program outputs the total count of these palindromic substrings.

Example Input and Output

- Input: "abba"
- Output: 6
- Palindromic substrings: "a", "b", "b", "a", "abba", "bb"

Approaches to Solve the Problem

There are several methods to count palindromic substrings, each with different time complexities and implementations. Here are the most common approaches:

1. Naive Approach

The simplest method is to generate all possible substrings and check each one for being a palindrome.

- Time Complexity: O(n^3)
- Space Complexity: O(1)

Steps:
1. Generate all possible substrings of the input string.
2. For each substring, check if it is a palindrome by comparing it to its reverse.
3. Count the palindromic substrings.

While this approach is straightforward, it is inefficient for large strings.

2. Expand Around Center Approach

This method improves efficiency by only checking possible centers of palindromes. For each character (and each pair of characters), we expand outward as long as we continue to find palindromic characters.

- Time Complexity: O(n^2)
- Space Complexity: O(1)

Steps:
1. For each character in the string, treat it as the center of an odd-length palindrome and expand outwards.
2. Then, for each pair of consecutive characters, treat them as the center of an even-length palindrome and expand outwards.
3. Count how many times you can expand for each center.

3. Dynamic Programming Approach

Using dynamic programming can also optimize the palindrome counting process.

- Time Complexity: O(n^2)
- Space Complexity: O(n^2)

Steps:

1. Create a 2D array where dp[i][j] is true if the substring from index i to j is a palindrome.
2. Initialize the table for single characters and two-character substrings.
3. Fill in the table based on previously computed values, then count the palindromic substrings.

Implementing the Solution

Here, we will implement the Expand Around Center approach in Python, as it strikes a balance between clarity and efficiency.

```python
def count_palindromic_substrings(s):
n = len(s)
count = 0

def expand_around_center(left, right):
nonlocal count
while left >= 0 and right < n and s[left] == s[right]:
count += 1
left -= 1
right += 1

for i in range(n):
Odd-length palindromes
expand_around_center(i, i)
Even-length palindromes
expand_around_center(i, i + 1)

return count

Example usage
input_string = "abba"
print(count_palindromic_substrings(input_string)) Output: 6
```

Conclusion

The palindrome counter hackerrank solution showcases the beauty of algorithmic problem-solving. By understanding the characteristics of palindromes and employing efficient strategies, you can tackle this problem effectively. Whether you choose the naive approach, expand around the center, or dynamic programming, the key is to grasp the underlying principles of string manipulation and substring analysis.

Practice is essential to mastering these concepts, so try implementing the solution in various programming languages and experimenting with different inputs. Happy coding!

Frequently Asked Questions

What is a palindrome in the context of strings?

A palindrome is a string that reads the same forwards and backwards, such as 'madam' or 'racecar'.

What is the objective of the Palindrome Counter challenge on HackerRank?

The objective is to count the number of palindrome substrings within a given string.

What are some common approaches to solving the Palindrome Counter problem?

Common approaches include using dynamic programming, expanding around potential centers, or checking all substrings for palindrome properties.

How can you optimize the solution for the Palindrome Counter problem?

You can optimize by avoiding checking all substrings by using the center expansion method, which reduces unnecessary checks.

What is the time complexity of the naive solution for counting palindromes?

The naive solution has a time complexity of O(n^3) due to checking all substrings and verifying each one.

What is a common mistake when implementing the Palindrome Counter solution?

A common mistake is failing to account for even-length palindromes when counting from character centers.

Can you explain how to implement the center expansion method for this problem?

The center expansion method involves iterating through each character in the string and attempting to expand outwards to check for palindromes, counting them as you go.

Palindrome Counter Hackerrank Solution

Palindrome Counter Hackerrank Solution

Related Articles

- paperless employee accord hr
- out of the dust answer key
- paper dolls and paper airplanes

Back to Home