5. Longest Palindromic Substring

April 8, 2016 | 922K views

Average Rating: 4.67 (475 votes)

Given a string **s**, find the longest palindromic substring in **s**. You may assume that the maximum length of **s** is 1000.

Example 1:

```
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: "cbbd"
Output: "bb"
```

Summary

This article is for intermediate readers. It introduces the following ideas: Palindrome, Dynamic Programming and String Manipulation. Make sure you understand what a palindrome means. A palindrome is a string which reads the same in both directions. For example, S = "aba" is a palindrome, S = "abc" is not.

Solution

Approach 1: Longest Common Substring

Common mistake

Some people will be tempted to come up with a quick solution, which is unfortunately flawed (however can be corrected easily):

> Reverse S and become S'. Find the longest common substring between S and S', which must also be the longest palindromic substring.

This seemed to work, let's see some examples below.

For example, S = "caba", S' = "abac".

The longest common substring between S and S' is "aba", which is the answer.

Let's try another example: S = "abacdfgdcaba", S' = "abacdgfdcaba".

The longest common substring between S and S' is "abacd". Clearly, this is not a valid palindrome.

Algorithm

We could see that the longest common substring method fails when there exists a reversed copy of a non-palindromic substring in some other part of S. To rectify this, each time we find a longest common substring candidate, we check if the substring's indices are the same as the reversed substring's original indices. If it is, then we attempt to update the longest palindrome found so far; if not, we skip this and find the next candidate.

This gives us an $O(n^2)$ Dynamic Programming solution which uses $O(n^2)$ space (could be improved to use $O(n)$ space). Please read more about Longest Common Substring here.

Approach 2: Brute Force

The obvious brute force solution is to pick all possible starting and ending positions for a substring, and verify if it is a palindrome.

Complexity Analysis

- Time complexity : $O(n^3)$. Assume that n is the length of the input string, there are a total of $\binom{n}{2} = \frac{n(n-1)}{2}$ such substrings (excluding the trivial solution where a character itself is a palindrome). Since verifying each substring takes $O(n)$ time, the run time complexity is $O(n^3)$.
- Space complexity : $O(1)$.

Approach 3: Dynamic Programming

To improve over the brute force solution, we first observe how we can avoid unnecessary re-computation while validating palindromes. Consider the case "ababa". If we already know that "bab" is a palindrome, it is obvious that "ababa" must be a palindrome since the two left and right end letters are the same.

We define $P(i, j)$ as following:

$$P(i,j) = \begin{cases} \text{true}, & \text{if the substring } S_i \dots S_j \text{ is a palindrome} \\ \text{false}, & \text{otherwise.} \end{cases}$$

Therefore,

$$P(i, j) = (P(i+1, j-1) \text{ and } S_i == S_j)$$

The base cases are:

$$P(i, i) = true$$

$$P(i, i+1) = (S_i == S_{i+1})$$

This yields a straight forward DP solution, which we first initialize the one and two letters palindromes, and work our way up finding all three letters palindromes, and so on...

Complexity Analysis

- Time complexity : $O(n^2)$. This gives us a runtime complexity of $O(n^2)$.
- Space complexity : $O(n^2)$. It uses $O(n^2)$ space to store the table.

Additional Exercise

Could you improve the above space complexity further and how?

Approach 4: Expand Around Center

In fact, we could solve it in $O(n^2)$ time using only constant space.

We observe that a palindrome mirrors around its center. Therefore, a palindrome can be expanded from its center, and there are only $2n - 1$ such centers.

You might be asking why there are $2n - 1$ but not n centers? The reason is the center of a palindrome can be in between two letters. Such palindromes have even number of letters (such as "abba") and its center are between the two 'b's.

Java

```
public String longestPalindrome(String s) {
    if (s == null || s.length() < 1) return "";
    int start = 0, end = 0;
    for (int i = 0; i < s.length(); i++) {
        int len1 = expandAroundCenter(s, i, i);
        int len2 = expandAroundCenter(s, i, i + 1);
        int len = Math.max(len1, len2);
        if (len > end - start) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }
    return s.substring(start, end + 1);
}

private int expandAroundCenter(String s, int left, int right) {
    int L = left, R = right;
    while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {
        L--;
        R++;
    }
    return R - L - 1;
}
```

Copy

Complexity Analysis

- Time complexity : $O(n^2)$. Since expanding a palindrome around its center could take $O(n)$ time, the overall complexity is $O(n^2)$.
- Space complexity : $O(1)$.

Approach 5: Manacher's Algorithm

There is even an $O(n)$ algorithm called Manacher's algorithm, explained here in detail. However, it is a non-trivial algorithm, and no one expects you to come up with this algorithm in a 45 minutes coding session. But, please go ahead and understand it, I promise it will be a lot of fun.

Rate this article:

Previous Next

Comments: 388

Sort By

Type comment here... (Markdown is supported)

Preview Post

ChristoPalas ★ 366 March 12, 2019 6:52 PM — Report

The Link for Manacher's Algorithm is broken.

328 Share Reply

SHOW 9 REPLIES

junchen424 ★ 57 June 6, 2018 7:57 AM

If "we check if the substring's indices are the same as the reversed substring's original indices" could be clarified more explicitly, that'd be great.

57 Share Reply

SHOW 5 REPLIES

ozarwork ★ 109 March 31, 2019 10:31 PM

Python3.
If I have to solve this in one hour, perhaps I may come up with this idea.
Approach 4: `Expand Around Center` is much better than this.

Vote Up Read More

72 Share Reply

SHOW 6 REPLIES

user4658O ★ 75 September 15, 2018 3:07 PM

C++:

```
class Solution {
public:
    string longestPalindrome(string s) {
```

Read More

35 Share Reply

SHOW 3 REPLIES

oakleaf ★ 48 July 24, 2018 9:34 PM — Report

My solution in Python 3:

```
class Solution:
    def longestPalindrome(self, s):
        """
```

Read More

48 Share Reply

SHOW 4 REPLIES

TaviJin ★ 94 November 2, 2018 6:07 PM

approach 3 in java

```
class Solution {
    public String longestPalindrome(String s) {
```

Read More

30 Share Reply

SHOW 4 REPLIES

zhengzhicong ★ 294 November 7, 2018 2:24 PM

python3:

```
class Solution:
    def longestPalindrome(self, s):
        """
```

Read More

42 Share Reply

haoyangfan ★ 897 February 22, 2019 1:53 PM

Approach 5: Link to the Manacher's algorithm is dead: 02/22/2019

25 Share Reply

SHOW 1 REPLY

metaalgorithmus ★ 21 March 18, 2019 10:24 PM

I am getting "Error establishing a database connection" for the Manacher's algorithm link

21 Share Reply

SHOW 1 REPLY

coder-coder ★ 71 May 4, 2019 7:01 AM — Report

@1337c0d3r Can you please fix the link in Approach 5. I am getting database error while accessing the link.

15 Share Reply

SHOW 2 REPLIES