5. Longest Palindromic Substring

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 (http://en.wikipedia.org/wiki/Longest_common_substring).

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 knew 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 Copy

```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;
}
```

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 (http://articles.leetcode.com/longest-palindromic-substring-part-ii/). 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.

Comments: 143

Sort By

Login to Comment
(/accounts/login/?next=/problems/longest-palindromic-substring/solution/)

(/msadler)

msadler (/msadler) ★ 0 September 6, 2018 9:00 PM

My Python Solution:

```python
    def longestPalindrome(self, s):
        """
        :type s: str
```

Read More

0 Share

(/water1)

water1 (/water1) ★ 0 August 23, 2018 4:20 AM

javascript实现
请帮我看看 还可以怎么 优化？谢谢

var longestPalindrome = function(s) {
if (s === s.split('').reverse().join('')) {

Read More

0 Share

(/abhigenie92)

abhigenie92 (/abhigenie92) ★ 4 August 15, 2018 1:20 AM

Can't you reverse and and then check it?

1 Share

SHOW 1 REPLY

(/user2925)

user2925 (/user2925) ★ 4 August 12, 2018 2:11 AM

optimize expand around center by exiting early...

```
function longestPalindrome(s) {
  var longest = '', c;
  for (c=0; c<s.length; c++) {
```

Read More

4 Share

(/windliang)

windliang (/windliang) ★ 47 August 10, 2018 12:09 PM

把 5 种算法都总结实现了一遍，http://windliang.cc/2018/08/05/leetCode-5-Longest-Palindromic-Substring/ (http://windliang.cc/2018/08/05/leetCode-5-Longest-Palindromic-Substring/)，大家可以参考下。

17 Share

SHOW 4 REPLIES

(/nathan_xiao)

Nathan_xiao (/nathan_xiao) ★ 0 August 9, 2018 10:20 AM

My soulution in Java：

```
class Solution {
public String longestPalindrome(String s) {
    if (s.length()<1) return s;
```

Read More

0 Share

(/tamazian)

tamazian (/tamazian) ★ 5 August 5, 2018 5:55 AM

"Expand Around Center" solution space complexity is O(N) because it uses stack

0 Share

SHOW 1 REPLY

(/jpizzo)

jpizzo (/jpizzo) ★ 1 August 4, 2018 12:24 AM

My solution in javascript:

```
var longestPalindrome = function(s) {
    let palindromes = [];
```

Read More

1 Share

(/loser_wang)

loser_wang (/loser_wang) ★ 0 August 3, 2018 3:35 AM

My solution in javascript:

Read More

0 Share

(/oakleaf)

oakleaf (/oakleaf) ★ 13 July 24, 2018 6:04 PM

My solution in Python 3:

Read More

13 Share

SHOW 1 REPLY

< 1 2 3 4 5 6 ... 14 15 >