Notes on Leetcode

Hanlin He

May 24, 2017

1 Longest Palindromic Substring

1.1 Brute Force

First idea is just brute force.

> Traverse the string.
>
> For each character, find the longest palindromic substring.
>
> In order to do that, traverse the substring began at that character. If a certain substring is palindromic, mark the length and continue.

The biggest problem for this brute force solution is two characters in the string might be repeatedly compared in all n round. In all the effort to optimize this solution, it still exceeds time limits occasionally.

1.2 Expanding

So here comes the second intuitive idea.

> Consider a character at index i in the middle of the string.
>
> If it is the center of a palindromic substring, then all the characters at its left and right must be mirrored. Therefore, we can check the characters at $i \pm k$ recursively.
>
> If we traverse the string, perform the previous operations, we can calculate all the palindromic substrings centered at each character.

This idea hugely reduce comparisons. Since each pair of characters can only be centered at one character. And each character is visited only once. Thus each pair of characters are compared only once. However, it contains an obvious ambiguity: What if a palindromic is even in length? There is no center character for an even length string.

To generalize the solution, here comes a modification: Use a series of same characters as center, rather than an aimless character. Here is the optimized solution.

1. Traverse the string, continue if same character found, stop until finding a different character.

2. View the series of same characters just found as center, expanding at both direction, until find unmatching character. The substring between two ends is a palindromic substring. Mark length and continue at the different character found in 1.

This solution turns out to be even more efficient than the original idea since long series of same character can be stepped over.

1.3 Longest Common Substring

Although the previous solution is quite efficient, we can still view the problem from another aspect.

> If we reverse the original string, the palindromic substring must be a common string of the reversed and original string. In this way, finding the longest palindromic substring is equals to finding the longest common substring between two string.

Nevertheless, the idea is not correct in all circumstances. As an example given in the editorial:

- $S = ''\texttt{abacdfgdcaba}''$
- $S_r = ''\texttt{abacdgfdcaba}''$

Obviously, the common substring "abacd" is not palindromic. A quick fix to this problem is check every time we find a common substring.

To solve the longest common substring problem, we can implement a $\mathcal{O}(n^2)$ dynamic programming solution, using $\mathcal{O}(n^2)$ or $\mathcal{O}(n)$ space.

1.4 To-Do

Besides the solution above, an even more efficient solution named Manacher's Algorithm is available, which is left as to do in the section.