LeetCode 132

https://leetcode.com/problems/palindrome-partitioning-ii/description/

Yifeng Zeng

Description

132. Palindrome Partitioning II

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given s = "aab",

Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

Idea Report

We can have a int[] f, where f[i] means the minumum cuts needed for a palindrome partitioning of s.substring(0, i+1). So we can just return f[len-1]. We can assume the maximum cuts, so the base cases are f[i] = i. (Inspring by LC 5. Longest Palindromic Substring), we can expand from the center to left and right. If s.charAt(left) == s.charAt(right), that means the s.substring.(left, right + 1) is palindrom, then f[right] = f[left - 1] + 1, adding 1 because the substring between left and right needed that 1 cut. And if left == 0, that means s.substring(0, right + 1) is palindrom, so no cut will be needed, so f[right] = 0. We need to consider both even and odd length of the string so we expand from [i, i] and [i, i+1]

Code:

```
class Solution {
    public int minCut(String s) {
        int len = s.length();
        int[] f = new int[len];
        for (int i = 0; i < len; i++) {
            f[i] = i;
        }

        for (int i = 0; i < len; i++) {
            search(s, f, i, i);
            search(s, f, i, i + 1);
        }

        // System.out.println(Arrays.toString(f));
        return f[len - 1];
    }

    private void search(String s, int[] f, int left, int right) {
        while (left >= 0 && right < s.length()
               && s.charAt(left) == s.charAt(right)) {
            if (left == 0) {
                // left == 0, substring(0, right+1) is palindrom, no cut needed
                f[right] = 0;
            } else {
                f[right] = Math.min(f[right], f[left - 1] + 1);
            }
            left--;
            right++;
        }
    }
}
```

Summary

- 2n time to traverse the center, n time to expand, so O(n^2) time complexity
- Similar to LC 5. Longest Palindromic Substring