Longest Palindromic Substring

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

Solution 1

```
string longestPalindrome(string s) {
    if (s.empty()) return "";
    if (s.size() == 1) return s;
    int min_start = 0, max_len = 1;
    for (int i = 0; i < s.size();) {
      if (s.size() - i <= max_len / 2) break;
      int j = i, k = i;
      while (k < s.size()-1 && s[k+1] == s[k]) ++k; // Skip duplicate characters.
      i = k+1;
      while (k < s.size()-1 && j > 0 && s[k + 1] == s[j - 1]) { ++k; --j; } // Ex
pand.
      int new_len = k - j + 1;
      if (new_len > max_len) { min_start = j; max_len = new_len; }
    }
    return s.substr(min_start, max_len);
}
```

written by hh1985 original link here

Solution 2

The performance is pretty good, surprisingly.

```
public class Solution {
private int lo, maxLen;

public String longestPalindrome(String s) {
    int len = s.length();
    if (len < 2)
        return s;

    for (int i = 0; i < len-1; i++) {
        extendPalindrome(s, i, i);  //assume odd length, try to extend Palindrome
as possible
        extendPalindrome(s, i, i+1); //assume even length.
    }
    return s.substring(lo, lo + maxLen);
}

private void extendPalindrome(String s, int j, int k) {
    while (j >= 0 && k < s.length() && s.charAt(j) == s.charAt(k)) {
        j--;
        k++;
    }
    if (maxLen < k - j - 1) {
        lo = j + 1;
        maxLen = k - j - 1;
    }
}}
```

written by jinwu original link here

Solution 3

```
class Solution {
public:
    std::string longestPalindrome(std::string s) {
        if (s.size() < 2)
            return s;
        int len = s.size(), max_left = 0, max_len = 1, left, right;
        for (int start = 0; start < len && len - start > max_len / 2;) {
            left = right = start;
            while (right < len - 1 && s[right + 1] == s[right])
                ++right;
            start = right + 1;
            while (right < len - 1 && left > 0 && s[right + 1] == s[left - 1]) {
                ++right;
                --left;
            }
            if (max_len < right - left + 1) {
                max_left = left;
                max_len = right - left + 1;
            }
        }
        return s.substr(max_left, max_len);
    }
};
```

written by prime_tang original link here