Problem 5: Longest Palindromic Substring

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a string

s

, return

the longest

palindromic

substring

in

s

.

Example 1:

Input:

s = "babad"

Output:

"bab"

Explanation:

"aba" is also a valid answer.

Example 2:

Input:

s = "cbbd"

Output:

"bb"

Constraints:

1 <= s.length <= 1000

s

consist of only digits and English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

Python3:

```
class Solution:
def longestPalindrome(self, s: str) -> str:
```

Python:

```
class Solution(object):
def longestPalindrome(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var longestPalindrome = function(s) {

};
```

TypeScript:

```
function longestPalindrome(s: string): string {

};
```

C#:

```
public class Solution {
public string LongestPalindrome(string s) {

}
}
```

C:

```
char* longestPalindrome(char* s) {

}
```

Go:

```
func longestPalindrome(s string) string {

}
```

Kotlin:

```
class Solution {
fun longestPalindrome(s: String): String {

}
}
```

Swift:

```
class Solution {
func longestPalindrome(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn longest_palindrome(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def longest_palindrome(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
function longestPalindrome($s) {

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def longestPalindrome(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_palindrome(s :: String.t) :: String.t
def longest_palindrome(s) do

end
end
```

Erlang:

```
-spec longest_palindrome(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
longest_palindrome(S) ->
```

```
.
```

Racket:

```
(define/contract (longest-palindrome s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindromic Substring
Difficulty: Medium
Tags: array, string, tree, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def longestPalindrome(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestPalindrome(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @return {string}
*/
var longestPalindrome = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function longestPalindrome(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
```

```
public string LongestPalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Longest Palindromic Substring
* Difficulty: Medium
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

char* longestPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Longest Palindromic Substring
// Difficulty: Medium
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestPalindrome(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun longestPalindrome(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func longestPalindrome(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Longest Palindromic Substring
// Difficulty: Medium
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn longest_palindrome(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def longest_palindrome(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function longestPalindrome($s) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def longestPalindrome(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_palindrome(s :: String.t) :: String.t
def longest_palindrome(s) do

end
end
```

Erlang Solution:

```
-spec longest_palindrome(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
longest_palindrome(S) ->
.
```

Racket Solution:

```
(define/contract (longest-palindrome s)
(-> string? string?)
)
```