Problem 1392: Longest Happy Prefix

Problem Information

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

Problem Description

A string is called a

happy prefix

if is a

non-empty

prefix which is also a suffix (excluding itself).

Given a string

s

, return

the

longest happy prefix

of

s

. Return an empty string

""

if no such prefix exists.

Example 1:

Input:

s = "level"

Output:

"l"

Explanation:

s contains 4 prefix excluding itself ("l", "le", "lev", "leve"), and suffix ("l", "el", "vel", "evel"). The largest prefix which is also suffix is given by "l".

Example 2:

Input:

s = "abababab"

Output:

"abab"

Explanation:

"abab" is the largest prefix which is also suffix. They can overlap in the original string.

Constraints:

1 <= s.length <= 10

5

s

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* longestPrefix(char* s) {

}
```

Go:

```
func longestPrefix(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String longestPrefix(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Happy Prefix
Difficulty: Hard
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

function longestPrefix(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string LongestPrefix(string s) {

}
}
```

C Solution:

```
/*
* Problem: Longest Happy Prefix
* Difficulty: Hard
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* longestPrefix(char* s) {

}
```

Go Solution:

```
// Problem: Longest Happy Prefix
// Difficulty: Hard
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func longestPrefix(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Happy Prefix
// Difficulty: Hard
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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