'A'

Problem 1763: Longest Nice Substring

Problem Information

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

Problem Description

A string

s

is

nice

if, for every letter of the alphabet that

s

contains, it appears

both

in uppercase and lowercase. For example,

"abABB"

is nice because

'A'

and

'a'

appear, and

'B'

and

'b'

appear. However,

"abA"

is not because

'b'

appears, but

'B'

does not.

Given a string

s

, return

the longest

substring

of

s

that is

nice

. If there are multiple, return the substring of the

earliest

occurrence. If there are none, return an empty string

.

Example 1:

Input:

s = "YazaAay"

Output:

"aAa"

Explanation:

"aAa" is a nice string because 'A/a' is the only letter of the alphabet in s, and both 'A' and 'a' appear. "aAa" is the longest nice substring.

Example 2:

Input:

s = "Bb"

Output:

"Bb"

Explanation:

"Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

Example 3:

Input:

s = "c"

Output:

""

Explanation:

There are no nice substrings.

Constraints:

1 <= s.length <= 100

s

consists of uppercase and lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* longestNiceSubstring(char* s) {

}
```

Go:

```
func longestNiceSubstring(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

Python3 Solution:

```
"""
Problem: Longest Nice Substring
Difficulty: Easy
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function longestNiceSubstring(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

C Solution:

```
/*
* Problem: Longest Nice Substring
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* longestNiceSubstring(char* s) {

}
```

Go Solution:

```
// Problem: Longest Nice Substring
// Difficulty: Easy
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func longestNiceSubstring(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Nice Substring
// Difficulty: Easy
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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