Problem 76: Minimum Window Substring

Problem Information

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

Problem Description

Given two strings

s

and

t

of lengths

m

and

n

respectively, return

the

minimum window

substring

of

s

such that every character in

t

(

including duplicates

) is included in the window

. If there is no such substring, return

the empty string

""

.

The testcases will be generated such that the answer is

unique

.

Example 1:

Input:

s = "ADOBECODEBANC", t = "ABC"

Output:

"BANC"

Explanation:

The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input:

s = "a", t = "a"

Output:

"a"

Explanation:

The entire string s is the minimum window.

Example 3:

Input:

s = "a", t = "aa"

Output:

""

Explanation:

Both 'a's from t must be included in the window. Since the largest window of s only has one 'a', return empty string.

Constraints:

m == s.length

n == t.length

1 <= m, n <= 10

5

s

and

t

consist of uppercase and lowercase English letters.

Follow up:

Could you find an algorithm that runs in

O(m + n)

time?

Code Snippets

C++:

```
class Solution {
public:
string minWindow(string s, string t) {

}
};
```

Java:

```
class Solution {
public String minWindow(String s, String t) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @param {string} t
* @return {string}
*/
var minWindow = function(s, t) {

};
```

TypeScript:

```
function minWindow(s: string, t: string): string {

};
```

C#:

```
public class Solution {
public string MinWindow(string s, string t) {

}
}
```

C:

```
char* minWindow(char* s, char* t) {

}
```

Go:

```
func minWindow(s string, t string) string {

}
```

Kotlin:

```
class Solution {
fun minWindow(s: String, t: String): String {

}
}
```

Swift:

```
class Solution {
func minWindow(_ s: String, _ t: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn min_window(s: String, t: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {String}
def min_window(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
```

```
* @param String $t
* @return String
*/
function minWindow($s, $t) {

}
}
```

Dart:

```
class Solution {
String minWindow(String s, String t) {

}
}
```

Scala:

```
object Solution {
def minWindow(s: String, t: String): String = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_window(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> unicode:unicode_binary().
min_window(S, T) ->
.
```

Racket:

```
(define/contract (min-window s t)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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 minWindow(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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 minWindow(String s, String t) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Window Substring
Difficulty: Hard
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 minWindow(self, s: str, t: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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
* @param {string} t
* @return {string}
*/
var minWindow = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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 minWindow(s: string, t: string): string {

};
```

C# Solution:

```
/*
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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 MinWindow(string s, string t) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Window Substring
* Difficulty: Hard
* 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* minWindow(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Minimum Window Substring
// Difficulty: Hard
// 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 minWindow(s string, t string) string {

}
```

Kotlin Solution:

```
class Solution {
fun minWindow(s: String, t: String): String {

}
}
```

Swift Solution:

```
class Solution {
func minWindow(_ s: String, _ t: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Minimum Window Substring
// Difficulty: Hard
// 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 min_window(s: String, t: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {String}
def min_window(s, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return String
*/
function minWindow($s, $t) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String minWindow(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def minWindow(s: String, t: String): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_window(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> unicode:unicode_binary().
min_window(S, T) ->
.
```

Racket Solution:

```
(define/contract (min-window s t)
(-> string? string? string?)
)
```