Problem 727: Minimum Window Subsequence

Problem Information

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

Problem Description

Given strings

s1

and

s2

, return

the minimum contiguous substring part of

s1

, so that

s2

is a subsequence of the part

.

If there is no such window in

s1

that covers all characters in

s2

, return the empty string

""

. If there are multiple such minimum-length windows, return the one with the

left-most starting index

.

Example 1:

Input:

s1 = "abcdebdde", s2 = "bde"

Output:

"bcde"

Explanation:

"bcde" is the answer because it occurs before "bdde" which has the same length. "deb" is not a smaller window because the elements of s2 in the window must occur in order.

Example 2:

Input:

s1 = "jmeqksfrsdcmsiwvaovztaqenprpvnbstl", s2 = "u"

Output:

""

Constraints:

1 <= s1.length <= 2 * 10

4

1 <= s2.length <= 100

s1

and

s2

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string minWindow(string s1, string s2) {

}
};
```

Java:

```
class Solution {
public String minWindow(String s1, String s2) {

}
}
```

Python3:

```
class Solution:
def minWindow(self, s1: str, s2: str) -> str:
```

Python:

```
class Solution(object):
def minWindow(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {string} s2
* @return {string}
*/
var minWindow = function(s1, s2) {

};
```

TypeScript:

```
function minWindow(s1: string, s2: string): string {

};
```

C#:

```
public class Solution {
public string MinWindow(string s1, string s2) {

}
}
```

C:

```
char* minWindow(char* s1, char* s2) {

}
```

Go:

```
func minWindow(s1 string, s2 string) string {

}
```

Kotlin:

```
class Solution {
fun minWindow(s1: String, s2: String): String {

}
}
```

Swift:

```
class Solution {
func minWindow(_ s1: String, _ s2: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn min_window(s1: String, s2: String) -> String {

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @return {String}
def min_window(s1, s2)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
```

```
* @param String $s2
* @return String
*/
function minWindow($s1, $s2) {

}
}
```

Dart:

```
class Solution {
String minWindow(String s1, String s2) {

}
}
```

Scala:

```
object Solution {
def minWindow(s1: String, s2: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_window(s1 :: String.t, s2 :: String.t) :: String.t
def min_window(s1, s2) do

end
end
```

Erlang:

```
-spec min_window(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
min_window(S1, S2) ->
.
```

Racket:

```
(define/contract (min-window s1 s2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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 minWindow(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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 minWindow(String s1, String s2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Window Subsequence
Difficulty: Hard
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 minWindow(self, s1: str, s2: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minWindow(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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} s1
* @param {string} s2
* @return {string}
*/
var minWindow = function(s1, s2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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 minWindow(s1: string, s2: string): string {

};
```

C# Solution:

```
/*
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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 MinWindow(string s1, string s2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Window Subsequence
* Difficulty: Hard
* 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* minWindow(char* s1, char* s2) {

}
```

Go Solution:

```
// Problem: Minimum Window Subsequence
// Difficulty: Hard
// 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 minWindow(s1 string, s2 string) string {

}
```

Kotlin Solution:

```
class Solution {
fun minWindow(s1: String, s2: String): String {

}
}
```

Swift Solution:

```
class Solution {
func minWindow(_ s1: String, _ s2: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Minimum Window Subsequence
// Difficulty: Hard
// 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 min_window(s1: String, s2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @return {String}
def min_window(s1, s2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return String
*/
function minWindow($s1, $s2) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String minWindow(String s1, String s2) {

}
}
```

Scala Solution:

```
object Solution {
def minWindow(s1: String, s2: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_window(s1 :: String.t, s2 :: String.t) :: String.t
def min_window(s1, s2) do

end
end
```

Erlang Solution:

```
-spec min_window(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
min_window(S1, S2) ->
.
```

Racket Solution:

```
(define/contract (min-window s1 s2)
(-> string? string? string?)
)
```