Problem 1081: Smallest Subsequence of Distinct Characters

Problem Information

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

Problem Description

Given a string

s

, return

the

lexicographically smallest

subsequence

of

s

that contains all the distinct characters of

s

exactly once

.

Example 1:

Input:

s = "bcabc"

Output:

"abc"

Example 2:

Input:

s = "cbacdcbc"

Output:

"acdb"

Constraints:

1 <= s.length <= 1000

s

consists of lowercase English letters.

Note:

This question is the same as 316:

https://leetcode.com/problems/remove-duplicate-letters/

Code Snippets

C++:

```
class Solution {
public:
```

```
string smallestSubsequence(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
char* smallestSubsequence(char* s) {

}
```

Go:

```
func smallestSubsequence(s string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (smallest-subsequence s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String smallestSubsequence(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Subsequence of Distinct Characters
Difficulty: Medium
Tags: string, graph, greedy, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function smallestSubsequence(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Smallest Subsequence of Distinct Characters
* Difficulty: Medium
* Tags: string, graph, greedy, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* smallestSubsequence(char* s) {

}
```

Go Solution:

```
// Problem: Smallest Subsequence of Distinct Characters
// Difficulty: Medium
// Tags: string, graph, greedy, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func smallestSubsequence(s string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Smallest Subsequence of Distinct Characters
// Difficulty: Medium
// Tags: string, graph, greedy, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (smallest-subsequence s)
(-> string? string?)
)
```