Problem 2000: Reverse Prefix of Word

Problem Information

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

Problem Description

Given a

0-indexed

string

word

and a character

ch

,

reverse

the segment of

word

that starts at index

0

and ends at the index of the

first occurrence

of

ch

(

inclusive

). If the character

ch

does not exist in

word

, do nothing.

For example, if

word = "abcdefd"

and

ch = "d"

, then you should

reverse

the segment that starts at

0

and ends at

3

(

inclusive

). The resulting string will be

"

dcba

efd"

.

Return

the resulting string

.

Example 1:

Input:

word = "

abcd

efd", ch = "d"

Output:

"

dcba

efd"

Explanation:

The first occurrence of "d" is at index 3. Reverse the part of word from 0 to 3 (inclusive), the resulting string is "dcbaefd".

Example 2:

Input:

word = "

xyxz

xe", ch = "z"

Output:

"

zxyx

xe"

Explanation:

The first and only occurrence of "z" is at index 3. Reverse the part of word from 0 to 3 (inclusive), the resulting string is "zxyxxe".

Example 3:

Input:

word = "abcd", ch = "z"

Output:

"abcd"

Explanation:

"z" does not exist in word. You should not do any reverse operation, the resulting string is "abcd".

Constraints:

1 <= word.length <= 250

word

consists of lowercase English letters.

ch

is a lowercase English letter.

Code Snippets

C++:

```
class Solution {
public:
string reversePrefix(string word, char ch) {

}
};
```

Java:

```
class Solution {
public String reversePrefix(String word, char ch) {

}
}
```

Python3:

```
class Solution:
def reversePrefix(self, word: str, ch: str) -> str:
```

Python:

```
class Solution(object):
def reversePrefix(self, word, ch):
"""
:type word: str
:type ch: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} word
* @param {character} ch
* @return {string}
*/
var reversePrefix = function(word, ch) {

};
```

TypeScript:

```
function reversePrefix(word: string, ch: string): string {

};
```

C#:

```
public class Solution {
public string ReversePrefix(string word, char ch) {

}
}
```

C:

```
char* reversePrefix(char* word, char ch) {

}
```

Go:

```
func reversePrefix(word string, ch byte) string {

}
```

Kotlin:

```
class Solution {
fun reversePrefix(word: String, ch: Char): String {

}
}
```

Swift:

```
class Solution {
func reversePrefix(_ word: String, _ ch: Character) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn reverse_prefix(word: String, ch: char) -> String {

}
}
```

Ruby:

```
# @param {String} word
# @param {Character} ch
# @return {String}
def reverse_prefix(word, ch)

end
```

PHP:

```
class Solution {

/**
* @param String $word
```

```
* @param String $ch
* @return String
*/
function reversePrefix($word, $ch) {

}
}
```

Dart:

```
class Solution {
String reversePrefix(String word, String ch) {

}
}
```

Scala:

```
object Solution {
def reversePrefix(word: String, ch: Char): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reverse_prefix(word :: String.t, ch :: char) :: String.t
def reverse_prefix(word, ch) do

end
end
```

Erlang:

```
-spec reverse_prefix(Word :: unicode:unicode_binary(), Ch :: char()) ->
unicode:unicode_binary().
reverse_prefix(Word, Ch) ->
.
```

Racket:

```
(define/contract (reverse-prefix word ch)
(-> string? char? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string reversePrefix(string word, char ch) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String reversePrefix(String word, char ch) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Reverse Prefix of Word
Difficulty: Easy
Tags: array, string, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def reversePrefix(self, word: str, ch: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def reversePrefix(self, word, ch):
"""
:type word: str
:type ch: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {string} word
* @param {character} ch
* @return {string}
*/
var reversePrefix = function(word, ch) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function reversePrefix(word: string, ch: string): string {

};
```

C# Solution:

```
/*
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string ReversePrefix(string word, char ch) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Reverse Prefix of Word
* Difficulty: Easy
* Tags: array, string, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* reversePrefix(char* word, char ch) {

}
```

Go Solution:

```
// Problem: Reverse Prefix of Word
// Difficulty: Easy
// Tags: array, string, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func reversePrefix(word string, ch byte) string {

}
```

Kotlin Solution:

```
class Solution {
fun reversePrefix(word: String, ch: Char): String {

}
}
```

Swift Solution:

```
class Solution {
func reversePrefix(_ word: String, _ ch: Character) -> String {

}
}
```

Rust Solution:

```
// Problem: Reverse Prefix of Word
// Difficulty: Easy
// Tags: array, string, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn reverse_prefix(word: String, ch: char) -> String {

}
}
```

Ruby Solution:

```
# @param {String} word
# @param {Character} ch
# @return {String}
def reverse_prefix(word, ch)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @param String $ch
* @return String
*/
function reversePrefix($word, $ch) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String reversePrefix(String word, String ch) {

}
}
```

Scala Solution:

```
object Solution {
def reversePrefix(word: String, ch: Char): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reverse_prefix(word :: String.t, ch :: char) :: String.t
def reverse_prefix(word, ch) do

end
end
```

Erlang Solution:

```
-spec reverse_prefix(Word :: unicode:unicode_binary(), Ch :: char()) ->
unicode:unicode_binary().
reverse_prefix(Word, Ch) ->
.
```

Racket Solution:

```
(define/contract (reverse-prefix word ch)
(-> string? char? string?)
)
```