Problem 1328: Break a Palindrome

Problem Information

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

Problem Description

Given a palindromic string of lowercase English letters

palindrome

, replace

exactly one

character with any lowercase English letter so that the resulting string is

not

a palindrome and that it is the

lexicographically smallest

one possible.

Return

the resulting string. If there is no way to replace a character to make it not a palindrome, return an

empty string

.

A string

a

is lexicographically smaller than a string

b

(of the same length) if in the first position where

a

and

b

differ,

a

has a character strictly smaller than the corresponding character in

b

. For example,

"abcc"

is lexicographically smaller than

"abcd"

because the first position they differ is at the fourth character, and

'c'

is smaller than

'd'

.

Example 1:

Input:

palindrome = "abccba"

Output:

"aaccba"

Explanation:

There are many ways to make "abccba" not a palindrome, such as "

z

bccba", "a

a

ccba", and "ab

a

cba". Of all the ways, "aaccba" is the lexicographically smallest.

Example 2:

Input:

palindrome = "a"

Output:

""

Explanation:

There is no way to replace a single character to make "a" not a palindrome, so return an empty string.

Constraints:

1 <= palindrome.length <= 1000

palindrome

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string breakPalindrome(string palindrome) {

}
};
```

Java:

```
class Solution {
public String breakPalindrome(String palindrome) {

}
}
```

Python3:

```
class Solution:
def breakPalindrome(self, palindrome: str) -> str:
```

Python:

```
class Solution(object):
def breakPalindrome(self, palindrome):
"""
:type palindrome: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} palindrome
* @return {string}
*/
var breakPalindrome = function(palindrome) {

};
```

TypeScript:

```
function breakPalindrome(palindrome: string): string {

};
```

C#:

```
public class Solution {
public string BreakPalindrome(string palindrome) {

}
}
```

C:

```
char* breakPalindrome(char* palindrome) {

}
```

Go:

```
func breakPalindrome(palindrome string) string {

}
```

Kotlin:

```
class Solution {
fun breakPalindrome(palindrome: String): String {

}
}
```

Swift:

```
class Solution {
func breakPalindrome(_ palindrome: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn break_palindrome(palindrome: String) -> String {

}
}
```

Ruby:

```
# @param {String} palindrome
# @return {String}
def break_palindrome(palindrome)

end
```

PHP:

```
class Solution {

/**
* @param String $palindrome
* @return String
*/
function breakPalindrome($palindrome) {

}
```

```
}
```

Dart:

```
class Solution {
String breakPalindrome(String palindrome) {

}
}
```

Scala:

```
object Solution {
def breakPalindrome(palindrome: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec break_palindrome(palindrome :: String.t) :: String.t
def break_palindrome(palindrome) do

end
end
```

Erlang:

```
-spec break_palindrome(Palindrome :: unicode:unicode_binary()) ->
unicode:unicode_binary().
break_palindrome(Palindrome) ->
.
```

Racket:

```
(define/contract (break-palindrome palindrome)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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 breakPalindrome(string palindrome) {

}
};
```

Java Solution:

```
/**
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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 breakPalindrome(String palindrome) {

}
}
```

Python3 Solution:

```
"""
Problem: Break a Palindrome
```

```
Difficulty: Medium
Tags: string, graph, greedy

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 breakPalindrome(self, palindrome: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def breakPalindrome(self, palindrome):
"""
:type palindrome: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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} palindrome
* @return {string}
*/
var breakPalindrome = function(palindrome) {

};
```

TypeScript Solution:

```
/**
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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 breakPalindrome(palindrome: string): string {

};
```

C# Solution:

```
/*
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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 BreakPalindrome(string palindrome) {

}
}
```

C Solution:

```
/*
* Problem: Break a Palindrome
* Difficulty: Medium
* Tags: string, graph, greedy
*
* 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* breakPalindrome(char* palindrome) {

}
```

Go Solution:

```
// Problem: Break a Palindrome
// Difficulty: Medium
// Tags: string, graph, greedy
//
// 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 breakPalindrome(palindrome string) string {

}
```

Kotlin Solution:

```
class Solution {
fun breakPalindrome(palindrome: String): String {

}
}
```

Swift Solution:

```
class Solution {
func breakPalindrome(_ palindrome: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Break a Palindrome
// Difficulty: Medium
// Tags: string, graph, greedy
//
// 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 break_palindrome(palindrome: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} palindrome
# @return {String}
def break_palindrome(palindrome)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $palindrome
* @return String
*/
function breakPalindrome($palindrome) {

}
}
```

Dart Solution:

```
class Solution {
String breakPalindrome(String palindrome) {

}
}
```

Scala Solution:

```
object Solution {
def breakPalindrome(palindrome: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec break_palindrome(palindrome :: String.t) :: String.t
def break_palindrome(palindrome) do

end
end
```

Erlang Solution:

```
-spec break_palindrome(Palindrome :: unicode:unicode_binary()) ->
unicode:unicode_binary().
break_palindrome(Palindrome) ->
.
```

Racket Solution:

```
(define/contract (break-palindrome palindrome)
(-> string? string?)
)
```