Problem 3518: Smallest Palindromic Rearrangement II

Problem Information

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

Problem Description

You are given a

palindromic

string

s

and an integer

k

.

Return the

k-th

lexicographically smallest

palindromic

permutation

of

s

. If there are fewer than

k

distinct palindromic permutations, return an empty string.

Note:

Different rearrangements that yield the same palindromic string are considered identical and are counted once.

Example 1:

Input:

s = "abba", k = 2

Output:

"baab"

Explanation:

The two distinct palindromic rearrangements of

"abba"

are

"abba"

and

"baab"

.

Lexicographically,

"abba"

comes before

"baab"

. Since

k = 2

, the output is

"baab"

.

Example 2:

Input:

s = "aa", k = 2

Output:

""

Explanation:

There is only one palindromic rearrangement:

"aa"

.

The output is an empty string since

k = 2

exceeds the number of possible rearrangements.

Example 3:

Input:

s = "bacab", k = 1

Output:

"abcba"

Explanation:

The two distinct palindromic rearrangements of

"bacab"

are

"abcba"

and

"bacab"

.

Lexicographically,

"abcba"

comes before

"bacab"

. Since

k = 1

, the output is

"abcba"

.

Constraints:

1 <= s.length <= 10

4

s

consists of lowercase English letters.

s

is guaranteed to be palindromic.

1 <= k <= 10

6

Code Snippets

C++:

```
class Solution {
public:
string smallestPalindrome(string s, int k) {

}
};
```

Java:

```
class Solution {
public String smallestPalindrome(String s, int k) {

}
}
```

Python3:

```
class Solution:
def smallestPalindrome(self, s: str, k: int) -> str:
```

Python:

```
class Solution(object):
def smallestPalindrome(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var smallestPalindrome = function(s, k) {

};
```

TypeScript:

```
function smallestPalindrome(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string SmallestPalindrome(string s, int k) {
```

```
}
}
```

C:

```
char* smallestPalindrome(char* s, int k) {

}
```

Go:

```
func smallestPalindrome(s string, k int) string {

}
```

Kotlin:

```
class Solution {
fun smallestPalindrome(s: String, k: Int): String {

}
}
```

Swift:

```
class Solution {
func smallestPalindrome(_ s: String, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_palindrome(s: String, k: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def smallest_palindrome(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function smallestPalindrome($s, $k) {

}
}
```

Dart:

```
class Solution {
String smallestPalindrome(String s, int k) {

}
}
```

Scala:

```
object Solution {
def smallestPalindrome(s: String, k: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_palindrome(s :: String.t, k :: integer) :: String.t
def smallest_palindrome(s, k) do
```

```
end
end
```

Erlang:

```
-spec smallest_palindrome(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
smallest_palindrome(S, K) ->
.
```

Racket:

```
(define/contract (smallest-palindrome s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string smallestPalindrome(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String smallestPalindrome(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Palindromic Rearrangement II
Difficulty: Hard
Tags: string, graph, math, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def smallestPalindrome(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var smallestPalindrome = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function smallestPalindrome(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
```

```
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string SmallestPalindrome(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Palindromic Rearrangement II
* Difficulty: Hard
* Tags: string, graph, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* smallestPalindrome(char* s, int k) {

}
```

Go Solution:

```
// Problem: Smallest Palindromic Rearrangement II
// Difficulty: Hard
// Tags: string, graph, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func smallestPalindrome(s string, k int) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun smallestPalindrome(s: String, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func smallestPalindrome(_ s: String, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Smallest Palindromic Rearrangement II
// Difficulty: Hard
// Tags: string, graph, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn smallest_palindrome(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def smallest_palindrome(s, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function smallestPalindrome($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
String smallestPalindrome(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def smallestPalindrome(s: String, k: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_palindrome(s :: String.t, k :: integer) :: String.t
def smallest_palindrome(s, k) do

end
end
```

Erlang Solution:

```
-spec smallest_palindrome(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
smallest_palindrome(S, K) ->
.
```

Racket Solution:

```
(define/contract (smallest-palindrome s k)
(-> string? exact-integer? string?)
)
```