Problem 3472: Longest Palindromic Subsequence After at Most K Operations

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

.

In one operation, you can replace the character at any position with the next or previous letter in the alphabet (wrapping around so that

'a'

is after

'z'

). For example, replacing

'a'

with the next letter results in

'b'

, and replacing

'a'

with the previous letter results in

'z'

. Similarly, replacing

'z'

with the next letter results in

'a'

, and replacing

'z'

with the previous letter results in

'y'

.

Return the length of the

longest

palindromic

subsequence

of

s

that can be obtained after performing

at most

k

operations.

Example 1:

Input:

s = "abced", k = 2

Output:

3

Explanation:

Replace

s[1]

with the next letter, and

s

becomes

"acced"

.

Replace

s[4]

with the previous letter, and

s

becomes

"accec"

.

The subsequence

"ccc"

forms a palindrome of length 3, which is the maximum.

Example 2:

Input:

s = "

aaazzz

", k = 4

Output:

6

Explanation:

Replace

s[0]

with the previous letter, and

s

becomes

"zaazzz"

.

Replace

s[4]

with the next letter, and

s

becomes

"zaazaz"

.

Replace

s[3]

with the next letter, and

s

becomes

"zaaaaz"

.

The entire string forms a palindrome of length 6.

Constraints:

1 <= s.length <= 200

1 <= k <= 200

s

consists of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (longest-palindromic-subsequence s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindromic Subsequence After at Most K Operations
Difficulty: Medium
Tags: string, dp
```

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Longest Palindromic Subsequence After at Most K Operations
* Difficulty: Medium
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

int longestPalindromicSubsequence(char* s, int k) {

}
```

Go Solution:

```
// Problem: Longest Palindromic Subsequence After at Most K Operations
// Difficulty: Medium
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestPalindromicSubsequence(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Palindromic Subsequence After at Most K Operations
// Difficulty: Medium
```

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

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (longest-palindromic-subsequence s k)
(-> string? exact-integer? exact-integer?)
)
```