Problem 2370: Longest Ideal Subsequence

Problem Information

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

Problem Description

You are given a string

s

consisting of lowercase letters and an integer

k

. We call a string

t

ideal

if the following conditions are satisfied:

t

is a

subsequence

of the string

s

.

The absolute difference in the alphabet order of every two

adjacent

letters in

t

is less than or equal to

k

.

Return

the length of the

longest

ideal string

.

A

subsequence

is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Note

that the alphabet order is not cyclic. For example, the absolute difference in the alphabet order of

'a'

and

'z'

is

25

, not

1

.

Example 1:

Input:

s = "acfgbd", k = 2

Output:

4

Explanation:

The longest ideal string is "acbd". The length of this string is 4, so 4 is returned. Note that "acfgbd" is not ideal because 'c' and 'f' have a difference of 3 in alphabet order.

Example 2:

Input:

s = "abcd", k = 3

Output:

4

Explanation:

The longest ideal string is "abcd". The length of this string is 4, so 4 is returned.

Constraints:

1 <= s.length <= 10

5

0 <= k <= 25

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $s
```

```
* @param Integer $k
* @return Integer
*/
function longestIdealString($s, $k) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 longestIdealString(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 longestIdealString(String s, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Longest Ideal Subsequence
Difficulty: Medium
Tags: string, dp, hash

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 longestIdealString(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 longestIdealString = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 longestIdealString(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 LongestIdealString(string s, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Longest Ideal Subsequence
* Difficulty: Medium
* Tags: string, dp, hash
*
* 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 longestIdealString(char* s, int k) {

}
```

Go Solution:

```
// Problem: Longest Ideal Subsequence
// Difficulty: Medium
// Tags: string, dp, hash
//
// 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 longestIdealString(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Ideal Subsequence
// Difficulty: Medium
// Tags: string, dp, hash
//
// 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_ideal_string(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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