Problem 1898: Maximum Number of Removable Characters

Problem Information

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

Problem Description

You are given two strings

s

and

p

where

p

is a

subsequence

of

s

. You are also given a

distinct 0-indexed

integer array

removable

containing a subset of indices of

s

(

s

is also

0-indexed

).

You want to choose an integer

k

(

0 <= k <= removable.length

) such that, after removing

k

characters from

s

using the

first

k

indices in

removable

,

p

is still a

subsequence

of

s

. More formally, you will mark the character at

s[removable[i]]

for each

0 <= i < k

, then remove all marked characters and check if

p

is still a subsequence.

Return

the

maximum

k

you can choose such that

p

is still a

subsequence

of

s

after the removals

.

A

subsequence

of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

Example 1:

Input:

s = "abcacb", p = "ab", removable = [3,1,0]

Output:

2

Explanation

: After removing the characters at indices 3 and 1, "a

b

c

a

cb" becomes "accb". "ab" is a subsequence of "

a

cc

b

". If we remove the characters at indices 3, 1, and 0, "

ab

c

a

cb" becomes "ccb", and "ab" is no longer a subsequence. Hence, the maximum k is 2.

Example 2:

Input:

s = "abcbddddd", p = "abcd", removable = [3,2,1,4,5,6]

Output:

1

Explanation

: After removing the character at index 3, "abc

b

ddddd" becomes "abcddddd". "abcd" is a subsequence of "

abcd

dddd".

Example 3:

Input:

s = "abcab", p = "abc", removable = [0,1,2,3,4]

Output:

0

Explanation

: If you remove the first index in the array removable, "abc" is no longer a subsequence.

Constraints:

1 <= p.length <= s.length <= 10

5

0 <= removable.length < s.length

0 <= removable[i] < s.length

p

is a

subsequence

of

s

.

s

and

p

both consist of lowercase English letters.

The elements in

removable

are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
int maximumRemovals(string s, string p, vector<int>& removable) {

}
};
```

Java:

```
class Solution {
public int maximumRemovals(String s, String p, int[] removable) {

}
}
```

Python3:

```
class Solution:
def maximumRemovals(self, s: str, p: str, removable: List[int]) -> int:
```

Python:

```
class Solution(object):
def maximumRemovals(self, s, p, removable):
"""
:type s: str
:type p: str
:type removable: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} p
* @param {number[]} removable
* @return {number}
*/
var maximumRemovals = function(s, p, removable) {

};
```

TypeScript:

```
function maximumRemovals(s: string, p: string, removable: number[]): number {

};
```

C#:

```
public class Solution {
public int MaximumRemovals(string s, string p, int[] removable) {

}
}
```

C:

```
int maximumRemovals(char* s, char* p, int* removable, int removableSize) {

}
```

Go:

```
func maximumRemovals(s string, p string, removable []int) int {

}
```

Kotlin:

```
class Solution {
fun maximumRemovals(s: String, p: String, removable: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumRemovals(_ s: String, _ p: String, _ removable: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_removals(s: String, p: String, removable: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} p
# @param {Integer[]} removable
# @return {Integer}
def maximum_removals(s, p, removable)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $p
* @param Integer[] $removable
* @return Integer
*/
function maximumRemovals($s, $p, $removable) {

}
}
```

Dart:

```
class Solution {
int maximumRemovals(String s, String p, List<int> removable) {

}
}
```

Scala:

```
object Solution {
def maximumRemovals(s: String, p: String, removable: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_removals(s :: String.t, p :: String.t, removable :: [integer])
:: integer
def maximum_removals(s, p, removable) do

end
end
```

Erlang:

```
-spec maximum_removals(S :: unicode:unicode_binary(), P ::
unicode:unicode_binary(), Removable :: [integer()]) -> integer().
maximum_removals(S, P, Removable) ->
.
```

Racket:

```
(define/contract (maximum-removals s p removable)
(-> string? string? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Removable Characters
* Difficulty: Medium
* Tags: array, string, search
*
* 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:
int maximumRemovals(string s, string p, vector<int>& removable) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Removable Characters
* Difficulty: Medium
* Tags: array, string, search
*
* 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 int maximumRemovals(String s, String p, int[] removable) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Removable Characters
Difficulty: Medium
Tags: array, string, search

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 maximumRemovals(self, s: str, p: str, removable: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumRemovals(self, s, p, removable):
"""
:type s: str
:type p: str
:type removable: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Removable Characters
```

```
* Difficulty: Medium
* Tags: array, string, search
*
* 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} s
* @param {string} p
* @param {number[]} removable
* @return {number}
*/
var maximumRemovals = function(s, p, removable) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Removable Characters
* Difficulty: Medium
* Tags: array, string, search
*
* 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 maximumRemovals(s: string, p: string, removable: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Removable Characters
* Difficulty: Medium
* Tags: array, string, search
*
```

```
* 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 int MaximumRemovals(string s, string p, int[] removable) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Removable Characters
* Difficulty: Medium
* Tags: array, string, search
*
* 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
*/

int maximumRemovals(char* s, char* p, int* removable, int removableSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Removable Characters
// Difficulty: Medium
// Tags: array, string, search
//
// 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 maximumRemovals(s string, p string, removable []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumRemovals(s: String, p: String, removable: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumRemovals(_ s: String, _ p: String, _ removable: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Removable Characters
// Difficulty: Medium
// Tags: array, string, search
//
// 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 maximum_removals(s: String, p: String, removable: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} p
# @param {Integer[]} removable
# @return {Integer}
def maximum_removals(s, p, removable)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $p
* @param Integer[] $removable
* @return Integer
*/
function maximumRemovals($s, $p, $removable) {

}
}
```

Dart Solution:

```
class Solution {
int maximumRemovals(String s, String p, List<int> removable) {

}
}
```

Scala Solution:

```
object Solution {
def maximumRemovals(s: String, p: String, removable: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_removals(s :: String.t, p :: String.t, removable :: [integer])
:: integer
def maximum_removals(s, p, removable) do

end
end
```

Erlang Solution:

```
-spec maximum_removals(S :: unicode:unicode_binary(), P ::
unicode:unicode_binary(), Removable :: [integer()]) -> integer().
maximum_removals(S, P, Removable) ->
.
```

Racket Solution:

```
(define/contract (maximum-removals s p removable)
(-> string? string? (listof exact-integer?) exact-integer?)
)
```