Problem 3694: Distinct Points Reachable After Substring Removal

Problem Information

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

Problem Description

You are given a string

s

consisting of characters

'U'

,

'D'

,

'L'

, and

'R'

, representing moves on an infinite 2D Cartesian grid.

'U'

: Move from

(x, y)

to

(x, y + 1)

.

'D'

: Move from

(x, y)

to

(x, y - 1)

.

'L'

: Move from

(x, y)

to

(x - 1, y)

.

'R'

: Move from

(x, y)

to

(x + 1, y)

.

You are also given a positive integer

k

.

You

must

choose and remove

exactly one

contiguous substring of length

k

from

s

. Then, start from coordinate

(0, 0)

and perform the remaining moves in order.

Return an integer denoting the number of

distinct

final coordinates reachable.

Example 1:

Input:

s = "LUL", k = 1

Output:

2

Explanation:

After removing a substring of length 1,

s

can be

"UL"

,

"LL"

or

"LU"

. Following these moves, the final coordinates will be

(-1, 1)

,

(-2, 0)

and

(-1, 1)

respectively. There are two distinct points

(-1, 1)

and

(-2, 0)

so the answer is 2.

Example 2:

Input:

s = "UDLR", k = 4

Output:

1

Explanation:

After removing a substring of length 4,

s

can only be the empty string. The final coordinates will be

(0, 0)

. There is only one distinct point

(0, 0)

so the answer is 1.

Example 3:

Input:

s = "UU", k = 1

Output:

1

Explanation:

After removing a substring of length 1,

s

becomes

"U"

, which always ends at

(0, 1)

, so there is only one distinct final coordinate.

Constraints:

1 <= s.length <= 10

5

s

consists of only

'U'

,

'D'

,

'L'

, and

'R'

.

1 <= k <= s.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (distinct-points s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Distinct Points Reachable After Substring Removal
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

C# Solution:

```
/*
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
```

```
}
```

C Solution:

```
/*
* Problem: Distinct Points Reachable After Substring Removal
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int distinctPoints(char* s, int k) {

}
```

Go Solution:

```
// Problem: Distinct Points Reachable After Substring Removal
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func distinctPoints(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Distinct Points Reachable After Substring Removal
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (distinct-points s k)
(-> string? exact-integer? exact-integer?)
)
```