Problem 3443: Maximum Manhattan Distance After K Changes

Problem Information

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

Problem Description

You are given a string

s

consisting of the characters

'N'

,

'S'

,

'E'

, and

'W'

, where

s[i]

indicates movements in an infinite grid:

'N'

: Move north by 1 unit.

'S'

: Move south by 1 unit.

'E'

: Move east by 1 unit.

'W'

: Move west by 1 unit.

Initially, you are at the origin

(0, 0)

. You can change

at most

k

characters to any of the four directions.

Find the

maximum

Manhattan distance

from the origin that can be achieved

at any time

while performing the movements

in order

.

The

Manhattan Distance

between two cells

(x

i

, y

i

)

and

(x

j

, y

j

)

is

|x

i

- x

j

| + |y

i

- y

j

|

.

Example 1:

Input:

s = "NWSE", k = 1

Output:

3

Explanation:

Change

s[2]

from

'S'

to

'N'

. The string

s

becomes

"NWNE"

.

Movement

Position (x, y)

Manhattan Distance

Maximum

s[0] == 'N'

(0, 1)

0 + 1 = 1

1

s[1] == 'W'

(-1, 1)

1 + 1 = 2

2

s[2] == 'N'

(-1, 2)

1 + 2 = 3

3

s[3] == 'E'

(0, 2)

0 + 2 = 2

3

The maximum Manhattan distance from the origin that can be achieved is 3. Hence, 3 is the output.

Example 2:

Input:

s = "NSWWEW", k = 3

Output:

6

Explanation:

Change

s[1]

from

'S'

to

'N'

, and

s[4]

from

'E'

to

'W'

. The string

s

becomes

"NNWWWW"

.

The maximum Manhattan distance from the origin that can be achieved is 6. Hence, 6 is the output.

Constraints:

1 <= s.length <= 10

5

0 <= k <= s.length

s

consists of only

'N'

,

'S'

,

'E'

, and

'W'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-distance s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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:
int maxDistance(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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 int maxDistance(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Manhattan Distance After K Changes
Difficulty: Medium
Tags: string, 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 maxDistance(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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 {number}
*/
var maxDistance = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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 maxDistance(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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 int MaxDistance(string s, int k) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Manhattan Distance After K Changes
* Difficulty: Medium
* Tags: string, 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
*/

int maxDistance(char* s, int k) {

}
```

Go Solution:

```
// Problem: Maximum Manhattan Distance After K Changes
// Difficulty: Medium
// Tags: string, 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 maxDistance(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Manhattan Distance After K Changes
// Difficulty: Medium
// Tags: string, 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 max_distance(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-distance s k)
(-> string? exact-integer? exact-integer?)
)
```