

Problem 514: Freedom Trail

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

In the video game Fallout 4, the quest

"Road to Freedom"

requires players to reach a metal dial called the

"Freedom Trail Ring"

and use the dial to spell a specific keyword to open the door.

Given a string

ring

that represents the code engraved on the outer ring and another string

key

that represents the keyword that needs to be spelled, return

the minimum number of steps to spell all the characters in the keyword

.

Initially, the first character of the ring is aligned at the

"12:00"

direction. You should spell all the characters in

key

one by one by rotating

ring

clockwise or anticlockwise to make each character of the string key aligned at the

"12:00"

direction and then by pressing the center button.

At the stage of rotating the ring to spell the key character

key[i]

:

You can rotate the ring clockwise or anticlockwise by one place, which counts as

one step

. The final purpose of the rotation is to align one of

ring

's characters at the

"12:00"

direction, where this character must equal

key[i]

.

If the character

`key[i]`

has been aligned at the

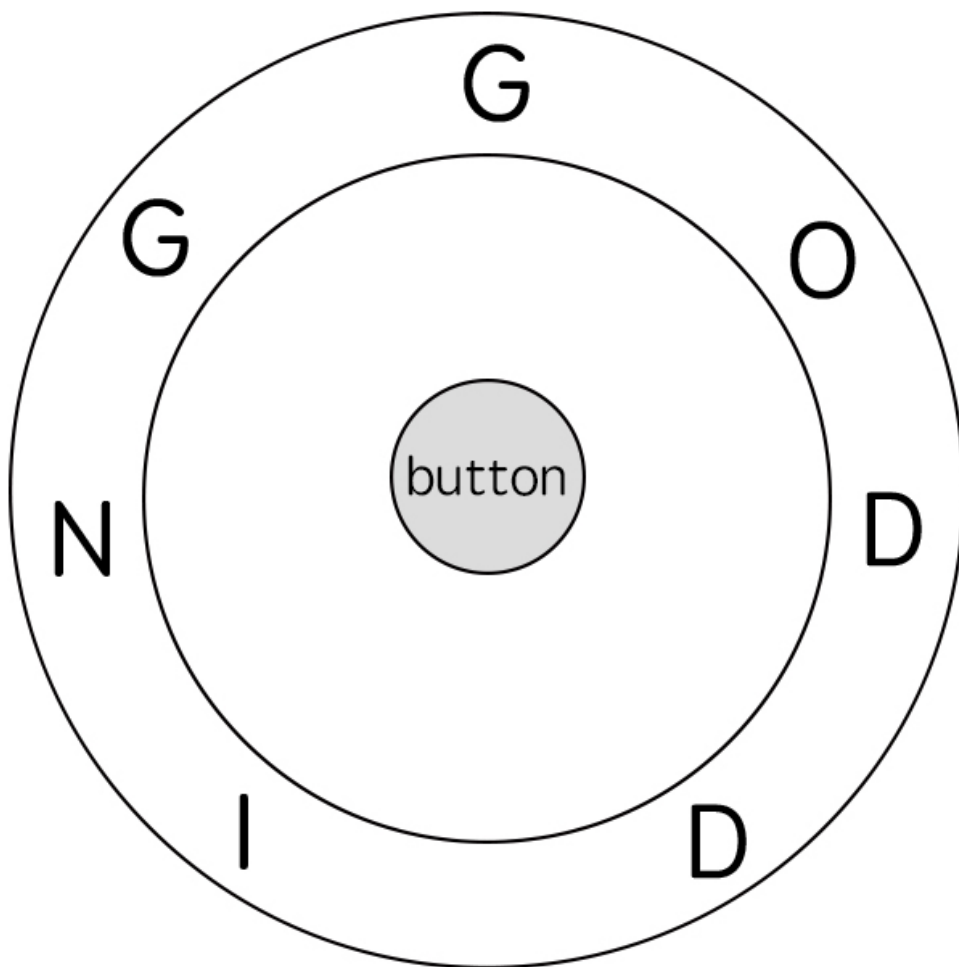
"12:00"

direction, press the center button to spell, which also counts as

one step

. After the pressing, you could begin to spell the next character in the key (next stage).
Otherwise, you have finished all the spelling.

Example 1:



Input:

ring = "godding", key = "gd"

Output:

4

Explanation:

For the first key character 'g', since it is already in place, we just need 1 step to spell this character. For the second key character 'd', we need to rotate the ring "godding" anticlockwise by two steps to make it become "ddinggo". Also, we need 1 more step for spelling. So the final output is 4.

Example 2:

Input:

ring = "godding", key = "godding"

Output:

13

Constraints:

$1 \leq \text{ring.length}, \text{key.length} \leq 100$

ring

and

key

consist of only lower case English letters.

It is guaranteed that

key

could always be spelled by rotating

ring

.

Code Snippets

C++:

```
class Solution {  
public:  
    int findRotateSteps(string ring, string key) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int findRotateSteps(String ring, String key) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def findRotateSteps(self, ring: str, key: str) -> int:
```

Python:

```
class Solution(object):  
    def findRotateSteps(self, ring, key):  
        """  
        :type ring: str  
        :type key: str  
        :rtype: int  
        """
```

JavaScript:

```
/**
 * @param {string} ring
 * @param {string} key
 * @return {number}
 */
var findRotateSteps = function(ring, key) {

};
```

TypeScript:

```
function findRotateSteps(ring: string, key: string): number {

};
```

C#:

```
public class Solution {
    public int FindRotateSteps(string ring, string key) {

    }
}
```

C:

```
int findRotateSteps(char* ring, char* key) {

}
```

Go:

```
func findRotateSteps(ring string, key string) int {

}
```

Kotlin:

```
class Solution {
    fun findRotateSteps(ring: String, key: String): Int {

    }
}
```

```
}
```

Swift:

```
class Solution {  
    func findRotateSteps(_ ring: String, _ key: String) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn find_rotate_steps(ring: String, key: String) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {String} ring  
# @param {String} key  
# @return {Integer}  
def find_rotate_steps(ring, key)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String $ring  
     * @param String $key  
     * @return Integer  
     */  
    function findRotateSteps($ring, $key) {  
  
    }  
}
```

Dart:

```

class Solution {
  int findRotateSteps(String ring, String key) {

  }
}

```

Scala:

```

object Solution {
  def findRotateSteps(ring: String, key: String): Int = {

  }
}

```

Elixir:

```

defmodule Solution do
  @spec find_rotate_steps(ring :: String.t, key :: String.t) :: integer
  def find_rotate_steps(ring, key) do

  end
end

```

Erlang:

```

-spec find_rotate_steps(Ring :: unicode:unicode_binary(), Key ::
unicode:unicode_binary()) -> integer().
find_rotate_steps(Ring, Key) ->
.

```

Racket:

```

(define/contract (find-rotate-steps ring key)
  (-> string? string? exact-integer?)
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Freedom Trail
 * Difficulty: Hard
 * Tags: string, dp, search
 *
 * 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 findRotateSteps(string ring, string key) {

    }

};

```

Java Solution:

```

/**
 * Problem: Freedom Trail
 * Difficulty: Hard
 * Tags: string, dp, search
 *
 * 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 findRotateSteps(String ring, String key) {

    }

}

```

Python3 Solution:

```

"""
Problem: Freedom Trail
Difficulty: Hard
Tags: string, dp, search
"""

```

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 findRotateSteps(self, ring: str, key: str) -> int:
```

```
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
```

```
def findRotateSteps(self, ring, key):
```

```
"""
```

```
:type ring: str
```

```
:type key: str
```

```
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
```

```
 * Problem: Freedom Trail
```

```
 * Difficulty: Hard
```

```
 * Tags: string, dp, search
```

```
 *
```

```
 * 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} ring
```

```
 * @param {string} key
```

```
 * @return {number}
```

```
 */
```

```
var findRotateSteps = function(ring, key) {
```

```
};
```

TypeScript Solution:

```
/**
 * Problem: Freedom Trail
 * Difficulty: Hard
 * Tags: string, dp, search
 *
 * 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 findRotateSteps(ring: string, key: string): number {

};
```

C# Solution:

```
/*
 * Problem: Freedom Trail
 * Difficulty: Hard
 * Tags: string, dp, search
 *
 * 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 FindRotateSteps(string ring, string key) {

    }
}
```

C Solution:

```
/*
 * Problem: Freedom Trail
 * Difficulty: Hard
 * Tags: string, dp, search
 *
 * 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 findRotateSteps(char* ring, char* key) {

}

```

Go Solution:

```

// Problem: Freedom Trail
// Difficulty: Hard
// Tags: string, dp, search
//
// 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 findRotateSteps(ring string, key string) int {

}

```

Kotlin Solution:

```

class Solution {
    fun findRotateSteps(ring: String, key: String): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func findRotateSteps(_ ring: String, _ key: String) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Freedom Trail
// Difficulty: Hard

```

```
// Tags: string, dp, search
//
// 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 find_rotate_steps(ring: String, key: String) -> i32 {

    }
}
```

Ruby Solution:

```
# @param {String} ring
# @param {String} key
# @return {Integer}
def find_rotate_steps(ring, key)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param String $ring
     * @param String $key
     * @return Integer
     */
    function findRotateSteps($ring, $key) {

    }
}
```

Dart Solution:

```
class Solution {
    int findRotateSteps(String ring, String key) {

    }
}
```

```
}
```

Scala Solution:

```
object Solution {  
  def findRotateSteps(ring: String, key: String): Int = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec find_rotate_steps(ring :: String.t, key :: String.t) :: integer  
  def find_rotate_steps(ring, key) do  
  
  end  
end
```

Erlang Solution:

```
-spec find_rotate_steps(Ring :: unicode:unicode_binary(), Key ::  
unicode:unicode_binary()) -> integer().  
find_rotate_steps(Ring, Key) ->  
.
```

Racket Solution:

```
(define/contract (find-rotate-steps ring key)  
  (-> string? string? exact-integer?)  
)
```