

Problem 752: Open the Lock

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots:

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'

. The wheels can rotate freely and wrap around: for example we can turn

'9'

to be

'0'

, or

'0'

to be

'9'

. Each move consists of turning one wheel one slot.

The lock initially starts at

'0000'

, a string representing the state of the 4 wheels.

You are given a list of

deadends

dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a

target

representing the value of the wheels that will unlock the lock, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

Example 1:

Input:

```
deadends = ["0201", "0101", "0102", "1212", "2002"], target = "0202"
```

Output:

6

Explanation:

A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202". Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would be invalid, because the wheels of the lock become stuck after the display becomes the dead end "0102".

Example 2:

Input:

```
deadends = ["8888"], target = "0009"
```

Output:

1

Explanation:

We can turn the last wheel in reverse to move from "0000" -> "0009".

Example 3:

Input:

```
deadends = ["8887", "8889", "8878", "8898", "8788", "8988", "7888", "9888"], target = "8888"
```

Output:

-1

Explanation:

We cannot reach the target without getting stuck.

Constraints:

$1 \leq \text{deadends.length} \leq 500$

$\text{deadends}[i].length == 4$

$\text{target.length} == 4$

target

will not be

in the list

deadends

.

target

and

deadends[i]

consist of digits only.

Code Snippets

C++:

```
class Solution {  
public:  
    int openLock(vector<string>& deadends, string target) {  
  
    }  
};
```

Java:

```
class Solution {  
public int openLock(String[] deadends, String target) {  
  
}  
}
```

Python3:

```
class Solution:  
    def openLock(self, deadends: List[str], target: str) -> int:
```

Python:

```
class Solution(object):  
    def openLock(self, deadends, target):  
        """  
        :type deadends: List[str]  
        :type target: str  
        :rtype: int
```

```
"""
```

JavaScript:

```
/**  
 * @param {string[]} deadends  
 * @param {string} target  
 * @return {number}  
 */  
var openLock = function(deadends, target) {  
  
};
```

TypeScript:

```
function openLock(deadends: string[], target: string): number {  
  
};
```

C#:

```
public class Solution {  
    public int OpenLock(string[] deadends, string target) {  
  
    }  
}
```

C:

```
int openLock(char** deadends, int deadendsSize, char* target) {  
  
}
```

Go:

```
func openLock(deadends []string, target string) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun openLock(deadends: Array<String>, target: String): Int {  
        }  
        }  
}
```

Swift:

```
class Solution {  
    func openLock(_ deadends: [String], _ target: String) -> Int {  
        }  
        }  
}
```

Rust:

```
impl Solution {  
    pub fn open_lock(deadends: Vec<String>, target: String) -> i32 {  
        }  
        }  
}
```

Ruby:

```
# @param {String[]} deadends  
# @param {String} target  
# @return {Integer}  
def open_lock(deadends, target)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String[] $deadends  
     * @param String $target  
     * @return Integer  
     */  
    function openLock($deadends, $target) {  
  
    }
```

```
}
```

Dart:

```
class Solution {  
    int openLock(List<String> deadends, String target) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def openLock(deadends: Array[String], target: String): Int = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
  @spec open_lock([String.t], String.t) :: integer  
  def open_lock(deadends, target) do  
  
  end  
end
```

Erlang:

```
-spec open_lock([unicode:unicode_binary()], Target ::  
  unicode:unicode_binary()) -> integer().  
open_lock(Deadends, Target) ->  
.
```

Racket:

```
(define/contract (open-lock deadends target)  
  (-> (listof string?) string? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Open the Lock
 * Difficulty: Medium
 * Tags: array, string, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    int openLock(vector<string>& deadends, string target) {
}
```

Java Solution:

```
/**
 * Problem: Open the Lock
 * Difficulty: Medium
 * Tags: array, string, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
    public int openLock(String[] deadends, String target) {
}
```

Python3 Solution:

```
"""
Problem: Open the Lock
```

Difficulty: Medium

Tags: array, string, hash, search

Approach: Use two pointers or sliding window technique

Time Complexity: $O(n)$ or $O(n \log n)$

Space Complexity: $O(n)$ for hash map

"""

```
class Solution:
    def openLock(self, deadends: List[str], target: str) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

```
class Solution(object):
    def openLock(self, deadends, target):
        """
        :type deadends: List[str]
        :type target: str
        :rtype: int
        """
```

JavaScript Solution:

```
/**
 * Problem: Open the Lock
 * Difficulty: Medium
 * Tags: array, string, hash, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity:  $O(n)$  or  $O(n \log n)$ 
 * Space Complexity:  $O(n)$  for hash map
 */

/**
 * @param {string[]} deadends
 * @param {string} target
 * @return {number}
 */
var openLock = function(deadends, target) {
```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Open the Lock  
 * Difficulty: Medium  
 * Tags: array, string, hash, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
function openLock(deadends: string[], target: string): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Open the Lock  
 * Difficulty: Medium  
 * Tags: array, string, hash, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
public class Solution {  
    public int OpenLock(string[] deadends, string target) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Open the Lock
```

```

* Difficulty: Medium
* Tags: array, string, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
}

int openLock(char** deadends, int deadendsSize, char* target) {
}

```

Go Solution:

```

// Problem: Open the Lock
// Difficulty: Medium
// Tags: array, string, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func openLock(deadends []string, target string) int {
}

```

Kotlin Solution:

```

class Solution {
    fun openLock(deadends: Array<String>, target: String): Int {
        }
    }
}
```

Swift Solution:

```

class Solution {
    func openLock(_ deadends: [String], _ target: String) -> Int {
        }
    }
}
```

Rust Solution:

```
// Problem: Open the Lock
// Difficulty: Medium
// Tags: array, string, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn open_lock(deadends: Vec<String>, target: String) -> i32 {
        }

    }
}
```

Ruby Solution:

```
# @param {String[]} deadends
# @param {String} target
# @return {Integer}
def open_lock(deadends, target)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param String[] $deadends
     * @param String $target
     * @return Integer
     */
    function openLock($deadends, $target) {

    }
}
```

Dart Solution:

```
class Solution {  
    int openLock(List<String> deadends, String target) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def openLock(deadends: Array[String], target: String): Int = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec open_lock(deadends :: [String.t], target :: String.t) :: integer  
  def open_lock(deadends, target) do  
  
  end  
end
```

Erlang Solution:

```
-spec open_lock(Deadends :: [unicode:unicode_binary()], Target ::  
  unicode:unicode_binary()) -> integer().  
open_lock(Deadends, Target) ->  
.
```

Racket Solution:

```
(define/contract (open-lock deadends target)  
  (-> (listof string?) string? exact-integer?))  
)
```