# Problem 691: Stickers to Spell Word

## Problem Information

**Difficulty:** Hard
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

We are given

n

different types of

stickers

. Each sticker has a lowercase English word on it.

You would like to spell out the given string

target

by cutting individual letters from your collection of stickers and rearranging them. You can use each sticker more than once if you want, and you have infinite quantities of each sticker.

Return

the minimum number of stickers that you need to spell out

target

. If the task is impossible, return

-1

.

Note:

In all test cases, all words were chosen randomly from the

1000

most common US English words, and

target

was chosen as a concatenation of two random words.

Example 1:

Input:

stickers = ["with","example","science"], target = "thehat"

Output:

3

Explanation:

We can use 2 "with" stickers, and 1 "example" sticker. After cutting and rearrange the letters of those stickers, we can form the target "thehat". Also, this is the minimum number of stickers necessary to form the target string.

Example 2:

Input:

stickers = ["notice","possible"], target = "basicbasic"

Output:

-1 Explanation: We cannot form the target "basicbasic" from cutting letters from the given stickers.

Constraints:

n == stickers.length

1 <= n <= 50

1 <= stickers[i].length <= 10

1 <= target.length <= 15

stickers[i]

and

target

consist of lowercase English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
int minStickers(vector<string>& stickers, string target) {

}
};
```

**Java:**

```
class Solution {
public int minStickers(String[] stickers, String target) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function minStickers(stickers: string[], target: string): number {

};
```

**C#:**

```
public class Solution {
public int MinStickers(string[] stickers, string target) {

}
}
```

**C:**

```
int minStickers(char** stickers, int stickersSize, char* target) {


}
```

**Go:**

```
func minStickers(stickers []string, target string) int {


}
```

**Kotlin:**

```
class Solution {
fun minStickers(stickers: Array<String>, target: String): Int {


}
}
```

**Swift:**

```
class Solution {
func minStickers(_ stickers: [String], _ target: String) -> Int {


}
}
```

**Rust:**

```
impl Solution {
pub fn min_stickers(stickers: Vec<String>, target: String) -> i32 {


}
}
```

**Ruby:**

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


end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

```
class Solution {
int minStickers(List<String> stickers, String target) {

}
}
```

**Scala:**

```
object Solution {
def minStickers(stickers: Array[String], target: String): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec min_stickers(stickers :: [String.t], target :: String.t) :: integer
def min_stickers(stickers, target) do

end
end
```

**Erlang:**

```
-spec min_stickers(Stickers :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary()) -> integer().
```

```
min_stickers(Stickers, Target) ->
.
```

### Racket:

```
(define/contract (min-stickers stickers target)
(-> (listof string?) string? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minStickers(vector<string>& stickers, string target) {

}
};
```

### Java Solution:

```
/**
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public int minStickers(String[] stickers, String target) {

}
}
```

### Python3 Solution:

```
"""
Problem: Stickers to Spell Word
Difficulty: Hard
Tags: array, string, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string[]} stickers
* @param {string} target
* @return {number}
*/
var minStickers = function(stickers, target) {

};
```

### TypeScript Solution:

```
/**
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minStickers(stickers: string[], target: string): number {

};
```

### C# Solution:

```
/*
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* 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 MinStickers(string[] stickers, string target) {

}
}
```

**C Solution:**

```
/*
* Problem: Stickers to Spell Word
* Difficulty: Hard
* Tags: array, string, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minStickers(char** stickers, int stickersSize, char* target) {

}
```

**Go Solution:**

```
// Problem: Stickers to Spell Word
// Difficulty: Hard
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minStickers(stickers []string, target string) int {

}
```

**Kotlin Solution:**

```
class Solution {
fun minStickers(stickers: Array<String>, target: String): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func minStickers(_ stickers: [String], _ target: String) -> Int {

}
}
```

**Rust Solution:**

```
// Problem: Stickers to Spell Word
// Difficulty: Hard
// Tags: array, string, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn min_stickers(stickers: Vec<String>, target: String) -> i32 {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

### Dart Solution:

```
class Solution {
int minStickers(List<String> stickers, String target) {

}
}
```

### Scala Solution:

```
object Solution {
def minStickers(stickers: Array[String], target: String): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec min_stickers(stickers :: [String.t], target :: String.t) :: integer
def min_stickers(stickers, target) do

end
end
```

### Erlang Solution:

```
-spec min_stickers(Stickers :: [unicode:unicode_binary()], Target ::
unicode:unicode_binary()) -> integer().
min_stickers(Stickers, Target) ->
```

```
.
```

## Racket Solution:

```
(define/contract (min-stickers stickers target)
(-> (listof string?) string? exact-integer?)
)
```