

# Problem 3291: Minimum Number of Valid Strings to Form Target I

## Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given an array of strings

words

and a string

target

.

A string

x

is called

valid

if

x

is a

prefix

of

any

string in

words

.

Return the

minimum

number of

valid

strings that can be

concatenated

to form

target

. If it is

not

possible to form

target

, return

-1

.

Example 1:

Input:

words = ["abc", "aaaaa", "bcdef"], target = "aabcdabc"

Output:

3

Explanation:

The target string can be formed by concatenating:

Prefix of length 2 of

words[1]

, i.e.

"aa"

.

Prefix of length 3 of

words[2]

, i.e.

"bcd"

.

Prefix of length 3 of

words[0]

, i.e.

"abc"

.

Example 2:

Input:

words = ["abababab","ab"], target = "ababaababa"

Output:

2

Explanation:

The target string can be formed by concatenating:

Prefix of length 5 of

words[0]

, i.e.

"ababa"

.

Prefix of length 5 of

words[0]

, i.e.

"ababa"

.

Example 3:

Input:

words = ["abcdef"], target = "xyz"

Output:

-1

Constraints:

$1 \leq \text{words.length} \leq 100$

$1 \leq \text{words}[i].\text{length} \leq 5 * 10$

3

The input is generated such that

$\text{sum}(\text{words}[i].\text{length}) \leq 10$

5

.

words[i]

consists only of lowercase English letters.

$1 \leq \text{target.length} \leq 5 * 10$

3

target

consists only of lowercase English letters.

## Code Snippets

### C++:

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

    }
};
```

### Java:

```
class Solution {
    public int minValidStrings(String[] words, String target) {

    }
}
```

### Python3:

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

### Python:

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

### JavaScript:

```
/**
 * @param {string[]} words
 * @param {string} target
 * @return {number}
 */
```

```
var minValidStrings = function(words, target) {  
  
};
```

### TypeScript:

```
function minValidStrings(words: string[], target: string): number {  
  
};
```

### C#:

```
public class Solution {  
    public int MinValidStrings(string[] words, string target) {  
  
    }  
}
```

### C:

```
int minValidStrings(char** words, int wordsSize, char* target) {  
  
}
```

### Go:

```
func minValidStrings(words []string, target string) int {  
  
}
```

### Kotlin:

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

### Swift:

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

```
}  
}
```

### Rust:

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

### Ruby:

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

### PHP:

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

### Dart:

```
class Solution {  
    int minValidStrings(List<String> words, String target) {  
  
    }  
}
```



### Scala:

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

### Elixir:

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

### Erlang:

```
-spec min_valid_strings(Words :: [unicode:unicode_binary()], Target ::  
unicode:unicode_binary()) -> integer().  
min_valid_strings(Words, Target) ->  
.
```

### Racket:

```
(define/contract (min-valid-strings words target)  
  (-> (listof string?) string? exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Minimum Number of Valid Strings to Form Target I  
 * Difficulty: Medium  
 * Tags: array, string, tree, dp, hash, search  
 *  
 * 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 minValidStrings(vector<string>& words, string target) {

    }
};

```

### Java Solution:

```

/**
 * Problem: Minimum Number of Valid Strings to Form Target I
 * Difficulty: Medium
 * Tags: array, string, tree, dp, hash, search
 *
 * 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 minValidStrings(String[] words, String target) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Minimum Number of Valid Strings to Form Target I
Difficulty: Medium
Tags: array, string, tree, dp, hash, search

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 minValidStrings(self, words: List[str], target: str) -> int:
    # TODO: Implement optimized solution
    pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
 * Problem: Minimum Number of Valid Strings to Form Target I
 * Difficulty: Medium
 * Tags: array, string, tree, dp, hash, search
 *
 * 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[]} words
 * @param {string} target
 * @return {number}
 */
var minValidStrings = function(words, target) {

};
```

### TypeScript Solution:

```
/**
 * Problem: Minimum Number of Valid Strings to Form Target I
 * Difficulty: Medium
 * Tags: array, string, tree, dp, hash, search
```

```

*
* 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 minValidStrings(words: string[], target: string): number {

};

```

### C# Solution:

```

/*
* Problem: Minimum Number of Valid Strings to Form Target I
* Difficulty: Medium
* Tags: array, string, tree, dp, hash, search
*
* 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 MinValidStrings(string[] words, string target) {

    }
}

```

### C Solution:

```

/*
* Problem: Minimum Number of Valid Strings to Form Target I
* Difficulty: Medium
* Tags: array, string, tree, dp, hash, search
*
* 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 minValidStrings(char** words, int wordsSize, char* target) {

```

```
}
```

### Go Solution:

```
// Problem: Minimum Number of Valid Strings to Form Target I
// Difficulty: Medium
// Tags: array, string, tree, dp, hash, search
//
// 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 minValidStrings(words []string, target string) int {

}
```

### Kotlin Solution:

```
class Solution {
    fun minValidStrings(words: Array<String>, target: String): Int {

    }
}
```

### Swift Solution:

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

    }
}
```

### Rust Solution:

```
// Problem: Minimum Number of Valid Strings to Form Target I
// Difficulty: Medium
// Tags: array, string, tree, dp, hash, search
//
// 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_valid_strings(words: Vec<String>, target: String) -> i32 {

    }
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

    }

}
```

### Dart Solution:

```
class Solution {
    int minValidStrings(List<String> words, String target) {

    }
}
```

### Scala Solution:

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

### Elixir Solution:

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

### Erlang Solution:

```
-spec min_valid_strings(Words :: [unicode:unicode_binary()], Target ::  
unicode:unicode_binary()) -> integer().  
min_valid_strings(Words, Target) ->  
.
```

### Racket Solution:

```
(define/contract (min-valid-strings words target)  
  (-> (listof string?) string? exact-integer?)  
)
```