Problem 3213: Construct String with Minimum Cost

Problem Information

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

Problem Description

You are given a string

target

, an array of strings

words

, and an integer array

costs

, both arrays of the same length.

Imagine an empty string

s

.

You can perform the following operation any number of times (including

zero

):

Choose an index

i

in the range

[0, words.length - 1]

.

Append

words[i]

to

s

.

The cost of operation is

costs[i]

.

Return the

minimum

cost to make

s

equal to

target

. If it's not possible, return

-1

.

Example 1:

Input:

target = "abcdef", words = ["abdef","abc","d","def","ef"], costs = [100,1,1,10,5]

Output:

7

Explanation:

The minimum cost can be achieved by performing the following operations:

Select index 1 and append

"abc"

to

s

at a cost of 1, resulting in

s = "abc"

.

Select index 2 and append

"d"

to

s

at a cost of 1, resulting in

s = "abcd"

.

Select index 4 and append

"ef"

to

s

at a cost of 5, resulting in

s = "abcdef"

.

Example 2:

Input:

target = "aaaa", words = ["z","zz","zzz"], costs = [1,10,100]

Output:

-1

Explanation:

It is impossible to make

s

equal to

target

, so we return -1.

Constraints:

1 <= target.length <= 5 * 10

4

1 <= words.length == costs.length <= 5 * 10

4

1 <= words[i].length <= target.length

The total sum of

words[i].length

is less than or equal to

5 * 10

4

.

target

and

words[i]

consist only of lowercase English letters.

1 <= costs[i] <= 10

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} target
* @param {string[]} words
* @param {number[]} costs
* @return {number}
*/
var minimumCost = function(target, words, costs) {

};
```

TypeScript:

```
function minimumCost(target: string, words: string[], costs: number[]):
number {

};
```

C#:

```
public class Solution {
public int MinimumCost(string target, string[] words, int[] costs) {

}
}
```

C:

```
int minimumCost(char* target, char** words, int wordsSize, int* costs, int
costsSize) {

}
```

Go:

```
func minimumCost(target string, words []string, costs []int) int {

}
```

Kotlin:

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

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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



}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-cost target words costs)
(-> string? (listof string?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 minimumCost(string target, vector<string>& words, vector<int>& costs) {

}
};
```

Java Solution:

```
/**
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 minimumCost(String target, String[] words, int[] costs) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct String with Minimum Cost
Difficulty: Hard
Tags: array, string, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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} target
* @param {string[]} words
* @param {number[]} costs
* @return {number}
*/
var minimumCost = function(target, words, costs) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 minimumCost(target: string, words: string[], costs: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 MinimumCost(string target, string[] words, int[] costs) {
}
}
```

C Solution:

```
/*
* Problem: Construct String with Minimum Cost
* Difficulty: Hard
* Tags: array, string, dp
*
* 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 minimumCost(char* target, char** words, int wordsSize, int* costs, int
costsSize) {

}
```

Go Solution:

```
// Problem: Construct String with Minimum Cost
// Difficulty: Hard
// Tags: array, string, dp
//
// 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 minimumCost(target string, words []string, costs []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Construct String with Minimum Cost
// Difficulty: Hard
// Tags: array, string, dp
//
// 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 minimum_cost(target: String, words: Vec<String>, costs: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

```
(define/contract (minimum-cost target words costs)
(-> string? (listof string?) (listof exact-integer?) exact-integer?)
)
```