Problem 2052: Minimum Cost to Separate Sentence Into Rows

Problem Information

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

Problem Description

You are given a string

sentence

containing words separated by spaces, and an integer

k

. Your task is to separate

sentence

into

rows

where the number of characters in each row is

at most

k

. You may assume that

sentence

does not begin or end with a space, and the words in

sentence

are separated by a single space.

You can split

sentence

into rows by inserting line breaks between words in

sentence

. A word

cannot

be split between two rows. Each word must be used exactly once, and the word order cannot be rearranged. Adjacent words in a row should be separated by a single space, and rows should not begin or end with spaces.

The

cost

of a row with length

n

is

(k - n)

2

, and the

total cost

is the sum of the

costs

for all rows

except

the last one.

For example if

sentence = "i love leetcode"

and

k = 12

:

Separating

sentence

into

"i"

,

"love"

, and

"leetcode"

has a cost of

(12 - 1)

2

+ (12 - 4)

2

= 185

.

Separating

sentence

into

"i love"

, and

"leetcode"

has a cost of

(12 - 6)

2

= 36

.

Separating

sentence

into

"i"

, and

"love leetcode"

is not possible because the length of

"love leetcode"

is greater than

k

.

Return

the

minimum

possible total cost of separating

sentence

into rows.

Example 1:

Input:

sentence = "i love leetcode", k = 12

Output:

36

Explanation:

Separating sentence into "i", "love", and "leetcode" has a cost of (12 - 1)

2

+ (12 - 4)

2

= 185. Separating sentence into "i love", and "leetcode" has a cost of (12 - 6)

2

= 36. Separating sentence into "i", "love leetcode" is not possible because "love leetcode" has length 13. 36 is the minimum possible total cost so return it.

Example 2:

Input:

sentence = "apples and bananas taste great", k = 7

Output:

21

Explanation

Separating sentence into "apples", "and", "bananas", "taste", and "great" has a cost of (7 - 6)

2

+ (7 - 3)

2

+ (7 - 7)

2

+ (7 - 5)

2

= 21. 21 is the minimum possible total cost so return it.

Example 3:

Input:

sentence = "a", k = 5

Output:

0

Explanation:

The cost of the last row is not included in the total cost, and since there is only one row, return 0.

Constraints:

1 <= sentence.length <= 5000

1 <= k <= 5000

The length of each word in

sentence

is at most

k

.

sentence

consists of only lowercase English letters and spaces.

sentence

does not begin or end with a space.

Words in

sentence

are separated by a single space.

Code Snippets

C++:

```
class Solution {
public:
int minimumCost(string sentence, int k) {

}
};
```

Java:

```
class Solution {
public int minimumCost(String sentence, int k) {

}
}
```

Python3:

```
class Solution:
def minimumCost(self, sentence: str, k: int) -> int:
```

Python:

```
class Solution(object):
def minimumCost(self, sentence, k):
"""
:type sentence: str
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} sentence
* @param {number} k
* @return {number}
*/
var minimumCost = function(sentence, k) {

};
```

TypeScript:

```
function minimumCost(sentence: string, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumCost(string sentence, int k) {

}
}
```

C:

```
int minimumCost(char* sentence, int k) {

}
```

Go:

```
func minimumCost(sentence string, k int) int {
}
```

Kotlin:

```
class Solution {
fun minimumCost(sentence: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumCost(_ sentence: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_cost(sentence: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} sentence
# @param {Integer} k
# @return {Integer}
def minimum_cost(sentence, k)

end
```

PHP:

```
class Solution {

/**
* @param String $sentence
* @param Integer $k
* @return Integer
*/
function minimumCost($sentence, $k) {

}
}
```

Dart:

```
class Solution {
int minimumCost(String sentence, int k) {

}
}
```

Scala:

```
object Solution {
def minimumCost(sentence: String, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_cost(sentence :: String.t, k :: integer) :: integer
def minimum_cost(sentence, k) do

end
end
```

Erlang:

```
-spec minimum_cost(Sentence :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_cost(Sentence, K) ->
.
```

Racket:

```
(define/contract (minimum-cost sentence k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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 sentence, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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 sentence, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Separate Sentence Into Rows
Difficulty: Medium
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, sentence: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCost(self, sentence, k):
"""
:type sentence: str
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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} sentence
* @param {number} k
* @return {number}
*/
var minimumCost = function(sentence, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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(sentence: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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 sentence, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Separate Sentence Into Rows
* Difficulty: Medium
* 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* sentence, int k) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Separate Sentence Into Rows
// Difficulty: Medium
// 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(sentence string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCost(sentence: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumCost(_ sentence: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Separate Sentence Into Rows
// Difficulty: Medium
// 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(sentence: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} sentence
# @param {Integer} k
# @return {Integer}
def minimum_cost(sentence, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $sentence
* @param Integer $k
* @return Integer
*/
function minimumCost($sentence, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCost(String sentence, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCost(sentence: String, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_cost(sentence :: String.t, k :: integer) :: integer
def minimum_cost(sentence, k) do

end
end
```

Erlang Solution:

```
-spec minimum_cost(Sentence :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_cost(Sentence, K) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost sentence k)
(-> string? exact-integer? exact-integer?)
)
```