Problem 1816: Truncate Sentence

Problem Information

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

Problem Description

A

sentence

is a list of words that are separated by a single space with no leading or trailing spaces. Each of the words consists of

only

uppercase and lowercase English letters (no punctuation).

For example,

"Hello World"

,

"HELLO"

, and

"hello world hello world"

are all sentences.

You are given a sentence

s

and an integer

k

. You want to

truncate

s

such that it contains only the

first

k

words. Return

s

after

truncating

it.

Example 1:

Input:

s = "Hello how are you Contestant", k = 4

Output:

"Hello how are you"

Explanation:

The words in s are ["Hello", "how" "are", "you", "Contestant"]. The first 4 words are ["Hello", "how", "are", "you"]. Hence, you should return "Hello how are you".

Example 2:

Input:

s = "What is the solution to this problem", k = 4

Output:

"What is the solution"

Explanation:

The words in s are ["What", "is" "the", "solution", "to", "this", "problem"]. The first 4 words are ["What", "is", "the", "solution"]. Hence, you should return "What is the solution".

Example 3:

Input:

s = "chopper is not a tanuki", k = 5

Output:

"chopper is not a tanuki"

Constraints:

1 <= s.length <= 500

k

is in the range

[1, the number of words in s]

.

s

consist of only lowercase and uppercase English letters and spaces.

The words in

s

are separated by a single space.

There are no leading or trailing spaces.

Code Snippets

C++:

```
class Solution {
public:
string truncateSentence(string s, int k) {

}
};
```

Java:

```
class Solution {
public String truncateSentence(String s, int k) {

}
}
```

Python3:

```
class Solution:
def truncateSentence(self, s: str, k: int) -> str:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function truncateSentence(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string TruncateSentence(string s, int k) {

}
}
```

C:

```
char* truncateSentence(char* s, int k) {

}
```

Go:

```
func truncateSentence(s string, k int) string {
```

```
}
```

Kotlin:

```
class Solution {
fun truncateSentence(s: String, k: Int): String {

}
}
```

Swift:

```
class Solution {
func truncateSentence(_ s: String, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn truncate_sentence(s: String, k: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def truncate_sentence(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
```

```
* @return String
*/
function truncateSentence($s, $k) {

}
}
```

Dart:

```
class Solution {
String truncateSentence(String s, int k) {

}
}
```

Scala:

```
object Solution {
def truncateSentence(s: String, k: Int): String = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec truncate_sentence(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
truncate_sentence(S, K) ->
.
```

Racket:

```
(define/contract (truncate-sentence s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string truncateSentence(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String truncateSentence(String s, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Truncate Sentence
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def truncateSentence(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function truncateSentence(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string TruncateSentence(string s, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Truncate Sentence
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* truncateSentence(char* s, int k) {

}
```

Go Solution:

```
// Problem: Truncate Sentence
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func truncateSentence(s string, k int) string {

}
```

Kotlin Solution:

```
class Solution {
fun truncateSentence(s: String, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func truncateSentence(_ s: String, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Truncate Sentence
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn truncate_sentence(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def truncate_sentence(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function truncateSentence($s, $k) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String truncateSentence(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def truncateSentence(s: String, k: Int): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec truncate_sentence(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
truncate_sentence(S, K) ->
.
```

Racket Solution:

```
(define/contract (truncate-sentence s k)
(-> string? exact-integer? string?)
)
```