Problem 2042: Check if Numbers Are Ascending in a Sentence

Problem Information

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

Problem Description

A sentence is a list of

tokens

separated by a

single

space with no leading or trailing spaces. Every token is either a

positive number

consisting of digits

0-9

with no leading zeros, or a

word

consisting of lowercase English letters.

For example,

"a puppy has 2 eyes 4 legs"

is a sentence with seven tokens:

"2"

and

"4"

are numbers and the other tokens such as

"puppy"

are words.

Given a string

s

representing a sentence, you need to check if

all

the numbers in

s

are

strictly increasing

from left to right (i.e., other than the last number,

each

number is

strictly smaller

than the number on its

right

in

s

).

Return

true

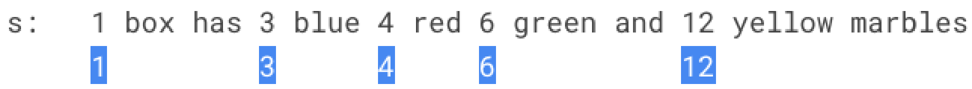
if so, or

false

otherwise

.

Example 1:



```
s:   1 box has 3 blue 4 red 6 green and 12 yellow marbles
     1         3      4     6                12
```

Input:

s = "1 box has 3 blue 4 red 6 green and 12 yellow marbles"

Output:

true

Explanation:

The numbers in s are: 1, 3, 4, 6, 12. They are strictly increasing from left to right: 1 < 3 < 4 < 6 < 12.

Example 2:

Input:

s = "hello world 5 x 5"

Output:

false

Explanation:

The numbers in s are:

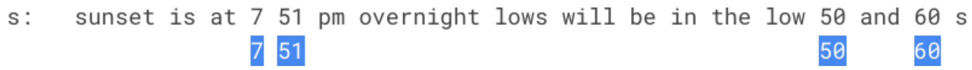
5

,

5

. They are not strictly increasing.

Example 3:



```
s:   sunset is at 7 51 pm overnight lows will be in the low 50 and 60 s
                  7 51                                     50     60
```

Input:

s = "sunset is at 7 51 pm overnight lows will be in the low 50 and 60 s"

Output:

false

Explanation:

The numbers in s are: 7,

51

,

50

, 60. They are not strictly increasing.

Constraints:

3 <= s.length <= 200

s

consists of lowercase English letters, spaces, and digits from

0

to

9

, inclusive.

The number of tokens in

s

is between

2

and

100

, inclusive.

The tokens in

s

are separated by a single space.

There are at least

two

numbers in

s

.

Each number in

s

is a

positive

number

less

than

100

, with no leading zeros.

s

contains no leading or trailing spaces.

Code Snippets

C++:

```
class Solution {
public:
bool areNumbersAscending(string s) {

}
};
```

Java:

```
class Solution {
public boolean areNumbersAscending(String s) {

}
}
```

Python3:

```
class Solution:
def areNumbersAscending(self, s: str) -> bool:
```

Python:

```
class Solution(object):
def areNumbersAscending(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {boolean}
*/
var areNumbersAscending = function(s) {

};
```

TypeScript:

```
function areNumbersAscending(s: string): boolean {

};
```

C#:

```
public class Solution {
public bool AreNumbersAscending(string s) {

}
}
```

C:

```
bool areNumbersAscending(char* s) {

}
```

Go:

```
func areNumbersAscending(s string) bool {

}
```

Kotlin:

```
class Solution {
fun areNumbersAscending(s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func areNumbersAscending(_ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn are_numbers_ascending(s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @return {Boolean}
def are_numbers_ascending(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function areNumbersAscending($s) {

}
}
```

Dart:

```
class Solution {
bool areNumbersAscending(String s) {

}
}
```

Scala:

```
object Solution {
def areNumbersAscending(s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec are_numbers_ascending(s :: String.t) :: boolean
def are_numbers_ascending(s) do

end
end
```

Erlang:

```
-spec are_numbers_ascending(S :: unicode:unicode_binary()) -> boolean().
are_numbers_ascending(S) ->
.
```

Racket:

```
(define/contract (are-numbers-ascending s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool areNumbersAscending(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean areNumbersAscending(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if Numbers Are Ascending in a Sentence
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def areNumbersAscending(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def areNumbersAscending(self, s):
"""
:type s: str
:rtype: bool
```

```

"""
```

JavaScript Solution:

```
/**
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {boolean}
*/
var areNumbersAscending = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function areNumbersAscending(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool AreNumbersAscending(string s) {

}
}
```

C Solution:

```
/*
* Problem: Check if Numbers Are Ascending in a Sentence
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool areNumbersAscending(char* s) {

}
```

Go Solution:

```
// Problem: Check if Numbers Are Ascending in a Sentence
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func areNumbersAscending(s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun areNumbersAscending(s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func areNumbersAscending(_ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if Numbers Are Ascending in a Sentence
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn are_numbers_ascending(s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Boolean}
def are_numbers_ascending(s)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function areNumbersAscending($s) {

}
}
```

Dart Solution:

```
class Solution {
bool areNumbersAscending(String s) {

}
}
```

Scala Solution:

```
object Solution {
def areNumbersAscending(s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec are_numbers_ascending(s :: String.t) :: boolean
def are_numbers_ascending(s) do

end
end
```

Erlang Solution:

```
-spec are_numbers_ascending(S :: unicode:unicode_binary()) -> boolean().
are_numbers_ascending(S) ->
.
```

Racket Solution:

```
(define/contract (are-numbers-ascending s)
(-> string? boolean?)
)
```