Problem 306: Additive Number

Problem Information

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

Problem Description

An

additive number

is a string whose digits can form an

additive sequence

.

A valid

additive sequence

should contain

at least

three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits, return

true

if it is an

additive number

or

false

otherwise.

Note:

Numbers in the additive sequence

cannot

have leading zeros, so sequence

1, 2, 03

or

1, 02, 3

is invalid.

Example 1:

Input:

"112358"

Output:

true

Explanation:

The digits can form an additive sequence: 1, 1, 2, 3, 5, 8. 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

Example 2:

Input:

"199100199"

Output:

true

Explanation:

The additive sequence is: 1, 99, 100, 199. 1 + 99 = 100, 99 + 100 = 199

Constraints:

1 <= num.length <= 35

num

consists only of digits.

Follow up:

How would you handle overflow for very large input integers?

Code Snippets

C++:

```
class Solution {
public:
bool isAdditiveNumber(string num) {

}
};
```

Java:

```
class Solution {
public boolean isAdditiveNumber(String num) {

}
}
```

Python3:

```
class Solution:
def isAdditiveNumber(self, num: str) -> bool:
```

Python:

```
class Solution(object):
def isAdditiveNumber(self, num):
"""
:type num: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} num
* @return {boolean}
*/
var isAdditiveNumber = function(num) {

};
```

TypeScript:

```
function isAdditiveNumber(num: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsAdditiveNumber(string num) {

}
}
```

C:

```
bool isAdditiveNumber(char* num) {

}
```

Go:

```
func isAdditiveNumber(num string) bool {

}
```

Kotlin:

```
class Solution {
fun isAdditiveNumber(num: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isAdditiveNumber(_ num: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_additive_number(num: String) -> bool {

}
}
```

Ruby:

```
# @param {String} num
# @return {Boolean}
def is_additive_number(num)

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @return Boolean
*/
function isAdditiveNumber($num) {

}
}
```

Dart:

```
class Solution {
bool isAdditiveNumber(String num) {

}
}
```

Scala:

```
object Solution {
def isAdditiveNumber(num: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_additive_number(num :: String.t) :: boolean
def is_additive_number(num) do

end
end
```

Erlang:

```
-spec is_additive_number(Num :: unicode:unicode_binary()) -> boolean().
is_additive_number(Num) ->
.
```

Racket:

```
(define/contract (is-additive-number num)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Additive Number
* Difficulty: Medium
* 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 isAdditiveNumber(string num) {

}
};
```

Java Solution:

```
/**
* Problem: Additive Number
* Difficulty: Medium
* 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 isAdditiveNumber(String num) {
}
}
```

Python3 Solution:

```
"""
Problem: Additive Number
Difficulty: Medium
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 isAdditiveNumber(self, num: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isAdditiveNumber(self, num):
"""
:type num: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Additive Number
* Difficulty: Medium
* 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} num
* @return {boolean}
*/
var isAdditiveNumber = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Additive Number
* Difficulty: Medium
* 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 isAdditiveNumber(num: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Additive Number
* Difficulty: Medium
* 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 IsAdditiveNumber(string num) {

}
}
```

C Solution:

```c
/*
* Problem: Additive Number
* Difficulty: Medium
* 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 isAdditiveNumber(char* num) {

}
```

Go Solution:

```go
// Problem: Additive Number
// Difficulty: Medium
// 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 isAdditiveNumber(num string) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun isAdditiveNumber(num: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isAdditiveNumber(_ num: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Additive Number
// Difficulty: Medium
// 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 is_additive_number(num: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} num
# @return {Boolean}
def is_additive_number(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @return Boolean
*/
function isAdditiveNumber($num) {

}
}
```

Dart Solution:

```
class Solution {
bool isAdditiveNumber(String num) {

}
}
```

Scala Solution:

```
object Solution {
def isAdditiveNumber(num: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_additive_number(num :: String.t) :: boolean
def is_additive_number(num) do

end
end
```

Erlang Solution:

```
-spec is_additive_number(Num :: unicode:unicode_binary()) -> boolean().
is_additive_number(Num) ->
.
```

Racket Solution:

```
(define/contract (is-additive-number num)
(-> string? boolean?)
)
```