Problem 842: Split Array into Fibonacci Sequence

Problem Information

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

Problem Description

You are given a string of digits

num

, such as

"123456579"

. We can split it into a Fibonacci-like sequence

[123, 456, 579]

.

Formally, a

Fibonacci-like

sequence is a list

f

of non-negative integers such that:

0 <= f[i] < 2

31

, (that is, each integer fits in a

32-bit

signed integer type),

f.length >= 3

, and

f[i] + f[i + 1] == f[i + 2]

for all

0 <= i < f.length - 2

.

Note that when splitting the string into pieces, each piece must not have extra leading zeroes, except if the piece is the number

0

itself.

Return any Fibonacci-like sequence split from

num

, or return

[]

if it cannot be done.

Example 1:

Input:

num = "1101111"

Output:

[11,0,11,11]

Explanation:

The output [110, 1, 111] would also be accepted.

Example 2:

Input:

num = "112358130"

Output:

[]

Explanation:

The task is impossible.

Example 3:

Input:

num = "0123"

Output:

[]

Explanation:

Leading zeroes are not allowed, so "01", "2", "3" is not valid.

Constraints:

1 <= num.length <= 200

num

contains only digits.

Code Snippets

C++:

```
class Solution {
public:
vector<int> splitIntoFibonacci(string num) {

}
};
```

Java:

```
class Solution {
public List<Integer> splitIntoFibonacci(String num) {

}
}
```

Python3:

```
class Solution:
def splitIntoFibonacci(self, num: str) -> List[int]:
```

Python:

```
class Solution(object):
def splitIntoFibonacci(self, num):
"""
:type num: str
```

```
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} num
* @return {number[]}
*/
var splitIntoFibonacci = function(num) {

};
```

TypeScript:

```
function splitIntoFibonacci(num: string): number[] {

};
```

C#:

```
public class Solution {
public IList<int> SplitIntoFibonacci(string num) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* splitIntoFibonacci(char* num, int* returnSize) {

}
```

Go:

```
func splitIntoFibonacci(num string) []int {

}
```

Kotlin:

```
class Solution {
fun splitIntoFibonacci(num: String): List<Int> {

}
}
```

Swift:

```
class Solution {
func splitIntoFibonacci(_ num: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn split_into_fibonacci(num: String) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} num
# @return {Integer[]}
def split_into_fibonacci(num)

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @return Integer[]
*/
function splitIntoFibonacci($num) {

}
}
```

Dart:

```
class Solution {
List<int> splitIntoFibonacci(String num) {

}
}
```

Scala:

```
object Solution {
def splitIntoFibonacci(num: String): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec split_into_fibonacci(num :: String.t) :: [integer]
def split_into_fibonacci(num) do

end
end
```

Erlang:

```
-spec split_into_fibonacci(Num :: unicode:unicode_binary()) -> [integer()].
split_into_fibonacci(Num) ->
.
```

Racket:

```
(define/contract (split-into-fibonacci num)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Array into Fibonacci Sequence
* 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:
vector<int> splitIntoFibonacci(string num) {

}
};
```

Java Solution:

```
/**
* Problem: Split Array into Fibonacci Sequence
* 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 List<Integer> splitIntoFibonacci(String num) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Array into Fibonacci Sequence
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 splitIntoFibonacci(self, num: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def splitIntoFibonacci(self, num):
"""
:type num: str
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Split Array into Fibonacci Sequence
* 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 {number[]}
*/
var splitIntoFibonacci = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Array into Fibonacci Sequence
* 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 splitIntoFibonacci(num: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Split Array into Fibonacci Sequence
* 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 IList<int> SplitIntoFibonacci(string num) {

}
}
```

C Solution:

```
/*
* Problem: Split Array into Fibonacci Sequence
* 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
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* splitIntoFibonacci(char* num, int* returnSize) {

}
```

Go Solution:

```
// Problem: Split Array into Fibonacci Sequence
// 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 splitIntoFibonacci(num string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun splitIntoFibonacci(num: String): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func splitIntoFibonacci(_ num: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Split Array into Fibonacci Sequence
// 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 split_into_fibonacci(num: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} num
# @return {Integer[]}
def split_into_fibonacci(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @return Integer[]
*/
function splitIntoFibonacci($num) {

}
}
```

Dart Solution:

```
class Solution {
List<int> splitIntoFibonacci(String num) {

}
}
```

Scala Solution:

```
object Solution {
def splitIntoFibonacci(num: String): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec split_into_fibonacci(num :: String.t) :: [integer]
def split_into_fibonacci(num) do

end
end
```

Erlang Solution:

```
-spec split_into_fibonacci(Num :: unicode:unicode_binary()) -> [integer()].
split_into_fibonacci(Num) ->
.
```

Racket Solution:

```
(define/contract (split-into-fibonacci num)
(-> string? (listof exact-integer?))
)
```