Problem 3023: Find Pattern in Infinite Stream I

Problem Information

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

Problem Description

You are given a binary array

pattern

and an object

stream

of class

InfiniteStream

representing a

0-indexed

infinite stream of bits.

The class

InfiniteStream

contains the following function:

int next()

: Reads a

single

bit (which is either

0

or

1

) from the stream and returns it.

Return

the

first starting

index where the pattern matches the bits read from the stream

. For example, if the pattern is

[1, 0]

, the first match is the highlighted part in the stream

[0,

1, 0

, 1, ...]

.

Example 1:

Input:

stream = [1,1,1,0,1,1,1,...], pattern = [0,1]

Output:

3

Explanation:

The first occurrence of the pattern [0,1] is highlighted in the stream [1,1,1,

0,1

,...], which starts at index 3.

Example 2:

Input:

stream = [0,0,0,0,...], pattern = [0]

Output:

0

Explanation:

The first occurrence of the pattern [0] is highlighted in the stream [

0

,...], which starts at index 0.

Example 3:

Input:

stream = [1,0,1,1,0,1,1,0,1,...], pattern = [1,1,0,1]

Output:

2

Explanation:

The first occurrence of the pattern [1,1,0,1] is highlighted in the stream [1,0,

1,1,0,1

,...], which starts at index 2.

Constraints:

1 <= pattern.length <= 100

pattern

consists only of

0

and

1

.

stream

consists only of

0

and

1

.

The input is generated such that the pattern's start index exists in the first

10

5

bits of the stream.

Code Snippets

C++:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* public:
* InfiniteStream(vector<int> bits);
* int next();
* };
*/
class Solution {
public:
int findPattern(InfiniteStream* stream, vector<int>& pattern) {

}
};
```

Java:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* public InfiniteStream(int[] bits);
* public int next();
* }
*/
class Solution {
public int findPattern(InfiniteStream infiniteStream, int[] pattern) {

}
}
```

Python3:

```
# Definition for an infinite stream.
# class InfiniteStream:
# def next(self) -> int:
# pass
class Solution:
def findPattern(self, stream: Optional['InfiniteStream'], pattern: List[int])
-> int:
```

Python:

```
# Definition for an infinite stream.
# class InfiniteStream:
# def next(self):
# pass
class Solution(object):
def findPattern(self, stream, pattern):
"""
:type stream: InfiniteStream
:type pattern: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* @param {number[]} bits
* constructor(bits);
*
* @return {number}
* next();
* }
*/
/**
* @param {InfiniteStream} stream
* @param {number[]} pattern
* @return {number}
```

```
*/
var findPattern = function(stream, pattern) {

};
```

TypeScript:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* constructor(bits: number[]);
* public next(): number;
* }
*/
function findPattern(stream: InfiniteStream, pattern: number[]): number {

};
```

C#:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* public InfiniteStream(int[] bits);
* public int Next();
* }
*/
public class Solution {
public int FindPattern(InfiniteStream stream, int[] pattern) {

}
}
```

C:

```
/**
* Definition for an infinite stream.
*
* YOU HAVE TO PASS THE OBJECT ITSELF AS THE FIRST PARAMETER
*
* struct InfiniteStream {
* int (*next)(struct InfiniteStream*);
* };
*/
int findPattern(struct InfiniteStream* stream, int* pattern, int patternSize)
{

}
```

Go:

```
/**
* Definition for an infinite stream.
* type InfiniteStream interface {
* Next() int
* }
*/
func findPattern(stream InfiniteStream, pattern []int) int {

}
```

Kotlin:

```
/**
* Definition for an infinite stream.
* class InfiniteStream(bits: IntArray) {
* fun next(): Int
* }
*/
class Solution {
fun findPattern(stream: InfiniteStream, pattern: IntArray): Int {

}
}
```

Swift:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* init(bits: [Int]) {}
* func next() -> Int {}
* }
*/
```

```
class Solution {
func findPattern(_ stream: InfiniteStream, _ pattern: [Int]) -> Int {

}
}
```

Rust:

```
/**
* Definition for an infinite stream.
* impl InfiniteStream {
* pub fn new(bits: Vec<i32>) -> Self {}
* pub fn next(&mut self) -> i32 {}
* }
*/
impl Solution {
pub fn find_pattern(mut stream: InfiniteStream, pattern: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# Definition for an infinite stream.
# class InfiniteStream
# def initialize(bits)
# end
# def next
# end
# end
# @param {InfiniteStream} stream
# @param {Integer[]} pattern
# @return {Integer}
def find_pattern(stream, pattern)

end
```

PHP:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
```

```
* function __construct(bits);
* function next();
* * @return Integer
* }
*/
class Solution {

/**
* @param InfiniteStream $stream
* @param Integer[] $pattern
* @return Integer
*/
function findPattern($stream, $pattern) {

}
}
```

Dart:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* InfiniteStream(List<int> bits);
* int next();
* }
*/
class Solution {
int findPattern(InfiniteStream stream, List<int> pattern) {

}
}
```

Scala:

```
/**
* Definition for an infinite stream.
* class InfiniteStream(bits: Array[Int]) {
* def next(): Int
* }
*/
object Solution {
def findPattern(stream: InfiniteStream, pattern: Array[Int]): Int = {
}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
* class InfiniteStream {
* public:
* InfiniteStream(vector<int> bits);
* int next();
* };
*/
class Solution {
public:
int findPattern(InfiniteStream* stream, vector<int>& pattern) {

}
};
```

Java Solution:

```
/**
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
* class InfiniteStream {
* public InfiniteStream(int[] bits);
* public int next();
* }
*/
class Solution {
public int findPattern(InfiniteStream infiniteStream, int[] pattern) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Pattern in Infinite Stream I
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

# Definition for an infinite stream.
# class InfiniteStream:
# def next(self) -> int:
# pass
class Solution:
def findPattern(self, stream: Optional['InfiniteStream'], pattern: List[int])
-> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for an infinite stream.
# class InfiniteStream:
# def next(self):
# pass
class Solution(object):
def findPattern(self, stream, pattern):
"""
:type stream: InfiniteStream
:type pattern: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
* class InfiniteStream {
* @param {number[]} bits
* constructor(bits);
*
* @return {number}
* next();
* }
*/
/**
* @param {InfiniteStream} stream
* @param {number[]} pattern
* @return {number}
*/
var findPattern = function(stream, pattern) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
* class InfiniteStream {
* constructor(bits: number[]);
* public next(): number;
* }
*/
function findPattern(stream: InfiniteStream, pattern: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
* class InfiniteStream {
* public InfiniteStream(int[] bits);
* public int Next();
* }
*/
```

```
public class Solution {
public int FindPattern(InfiniteStream stream, int[] pattern) {

}
}
```

C Solution:

```
/*
* Problem: Find Pattern in Infinite Stream I
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Definition for an infinite stream.
*
* YOU HAVE TO PASS THE OBJECT ITSELF AS THE FIRST PARAMETER
*
* struct InfiniteStream {
* int (*next)(struct InfiniteStream*);
* };
*/
int findPattern(struct InfiniteStream* stream, int* pattern, int patternSize)
{

}
```

Go Solution:

```
// Problem: Find Pattern in Infinite Stream I
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
/**
* Definition for an infinite stream.
* type InfiniteStream interface {
* Next() int
* }
*/
func findPattern(stream InfiniteStream, pattern []int) int {

}
```

Kotlin Solution:

```
/**
* Definition for an infinite stream.
* class InfiniteStream(bits: IntArray) {
* fun next(): Int
* }
*/
class Solution {
fun findPattern(stream: InfiniteStream, pattern: IntArray): Int {

}
}
```

Swift Solution:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* init(bits: [Int]) {}
* func next() -> Int {}
* }
*/
class Solution {
func findPattern(_ stream: InfiniteStream, _ pattern: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Pattern in Infinite Stream I
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

/**
* Definition for an infinite stream.
* impl InfiniteStream {
* pub fn new(bits: Vec<i32>) -> Self {}
* pub fn next(&mut self) -> i32 {}
* }
*/
impl Solution {
pub fn find_pattern(mut stream: InfiniteStream, pattern: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# Definition for an infinite stream.
# class InfiniteStream
# def initialize(bits)
# end
# def next
# end
# end
# @param {InfiniteStream} stream
# @param {Integer[]} pattern
# @return {Integer}
def find_pattern(stream, pattern)

end
```

PHP Solution:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
```

```
* function __construct(bits);
* function next();
* * @return Integer
* }
*/
class Solution {

/**
* @param InfiniteStream $stream
* @param Integer[] $pattern
* @return Integer
*/
function findPattern($stream, $pattern) {

}
}
```

Dart Solution:

```
/**
* Definition for an infinite stream.
* class InfiniteStream {
* InfiniteStream(List<int> bits);
* int next();
* }
*/
class Solution {
int findPattern(InfiniteStream stream, List<int> pattern) {

}
}
```

Scala Solution:

```
/**
* Definition for an infinite stream.
* class InfiniteStream(bits: Array[Int]) {
* def next(): Int
* }
*/
object Solution {
```

```
def findPattern(stream: InfiniteStream, pattern: Array[Int]): Int = {

}
}
```