Problem 604: Design Compressed String Iterator

Problem Information

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

Problem Description

Design and implement a data structure for a compressed string iterator. The given compressed string will be in the form of each letter followed by a positive integer representing the number of this letter existing in the original uncompressed string.

Implement the StringIterator class:

next()

Returns

the next character

if the original string still has uncompressed characters, otherwise returns a

white space

.

hasNext()

Returns true if there is any letter needs to be uncompressed in the original string, otherwise returns

false

.

Example 1:

Input

["StringIterator", "next", "next", "next", "next", "next", "next", "hasNext", "next", "hasNext"]
[["L1e2t1C1o1d1e1"], [], [], [], [], [], [], [], [], []]

Output

[null, "L", "e", "e", "t", "C", "o", true, "d", true]

Explanation

StringIterator stringIterator = new StringIterator("L1e2t1C1o1d1e1"); stringIterator.next(); // return "L" stringIterator.next(); // return "e" stringIterator.next(); // return "e" stringIterator.next(); // return "t" stringIterator.next(); // return "C" stringIterator.next(); // return "o" stringIterator.hasNext(); // return True stringIterator.next(); // return "d" stringIterator.hasNext(); // return True

Constraints:

1 <= compressedString.length <= 1000

compressedString

consists of lower-case an upper-case English letters and digits.

The number of a single character repetitions in

compressedString

is in the range

[1, 10^9]

At most

100

calls will be made to

next

and

hasNext

.

Code Snippets

C++:

```
class StringIterator {
public:
StringIterator(string compressedString) {

}

char next() {

}

bool hasNext() {

}
};

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator* obj = new StringIterator(compressedString);
* char param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java:

```
class StringIterator {

public StringIterator(String compressedString) {

}

public char next() {

}

public boolean hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = new StringIterator(compressedString);
* char param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3:

```
class StringIterator:

def __init__(self, compressedString: str):


def next(self) -> str:


def hasNext(self) -> bool:




# Your StringIterator object will be instantiated and called as such:
# obj = StringIterator(compressedString)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

Python:

```
class StringIterator(object):

def __init__(self, compressedString):
"""
:type compressedString: str
"""



def next(self):
"""
:rtype: str
"""



def hasNext(self):
"""
:rtype: bool
"""




# Your StringIterator object will be instantiated and called as such:
# obj = StringIterator(compressedString)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript:

```
/**
* @param {string} compressedString
*/
var StringIterator = function(compressedString) {

};

/**
* @return {character}
*/
StringIterator.prototype.next = function() {

};

/**
```

```
* @return {boolean}
*/
StringIterator.prototype.hasNext = function() {

};

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = new StringIterator(compressedString)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

TypeScript:

```
class StringIterator {
constructor(compressedString: string) {

}

next(): string {

}

hasNext(): boolean {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = new StringIterator(compressedString)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

C#:

```
public class StringIterator {

public StringIterator(string compressedString) {
```

```
}

public char Next() {

}

public bool HasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = new StringIterator(compressedString);
* char param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C:

```
typedef struct {

} StringIterator;


StringIterator* stringIteratorCreate(char* compressedString) {

}

char stringIteratorNext(StringIterator* obj) {

}

bool stringIteratorHasNext(StringIterator* obj) {

}

void stringIteratorFree(StringIterator* obj) {
```

```
}

/**
* Your StringIterator struct will be instantiated and called as such:
* StringIterator* obj = stringIteratorCreate(compressedString);
* char param_1 = stringIteratorNext(obj);

* bool param_2 = stringIteratorHasNext(obj);

* stringIteratorFree(obj);
*/
```

Go:

```
type StringIterator struct {

}


func Constructor(compressedString string) StringIterator {

}


func (this *StringIterator) Next() byte {

}


func (this *StringIterator) HasNext() bool {

}


/**
* Your StringIterator object will be instantiated and called as such:
* obj := Constructor(compressedString);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

Kotlin:

```
class StringIterator(compressedString: String) {

fun next(): Char {

}

fun hasNext(): Boolean {

}

}

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = StringIterator(compressedString)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift:

```

class StringIterator {

init(_ compressedString: String) {

}

func next() -> Character {

}

func hasNext() -> Bool {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* let obj = StringIterator(compressedString)
* let ret_1: Character = obj.next()
```

```
* let ret_2: Bool = obj.hasNext()
*/
```

Rust:

```
struct StringIterator {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl StringIterator {

fn new(compressedString: String) -> Self {

}

fn next(&self) -> char {

}

fn has_next(&self) -> bool {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* let obj = StringIterator::new(compressedString);
* let ret_1: char = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby:

```
class StringIterator

=begin
:type compressed_string: String
```

```
=end
def initialize(compressed_string)

end


=begin
:rtype: Character
=end
def next()

end


=begin
:rtype: Boolean
=end
def has_next()

end


end

# Your StringIterator object will be instantiated and called as such:
# obj = StringIterator.new(compressed_string)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP:

```
class StringIterator {
/**
* @param String $compressedString
*/
function __construct($compressedString) {

}

/**
* @return String
*/
```

```
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* $obj = StringIterator($compressedString);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
*/
```

Dart:

```
class StringIterator {

StringIterator(String compressedString) {

}

String next() {

}

bool hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = StringIterator(compressedString);
* String param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

Scala:

```
class StringIterator(_compressedString: String) {

def next(): Char = {

}

def hasNext(): Boolean = {

}

}

/**
* Your StringIterator object will be instantiated and called as such:
* val obj = new StringIterator(compressedString)
* val param_1 = obj.next()
* val param_2 = obj.hasNext()
*/
```

Elixir:

```
defmodule StringIterator do
@spec init_(compressed_string :: String.t) :: any
def init_(compressed_string) do

end

@spec next() :: char
def next() do

end

@spec has_next() :: boolean
def has_next() do

end
end

# Your functions will be called as such:
# StringIterator.init_(compressed_string)
```

```
# param_1 = StringIterator.next()
# param_2 = StringIterator.has_next()

# StringIterator.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang:

```
-spec string_iterator_init_(CompressedString :: unicode:unicode_binary()) ->
any().
string_iterator_init_(CompressedString) ->
.

-spec string_iterator_next() -> char().
string_iterator_next() ->
.

-spec string_iterator_has_next() -> boolean().
string_iterator_has_next() ->
.


%% Your functions will be called as such:
%% string_iterator_init_(CompressedString),
%% Param_1 = string_iterator_next(),
%% Param_2 = string_iterator_has_next(),

%% string_iterator_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
(define string-iterator%
(class object%
(super-new)

; compressed-string : string?
(init-field
compressed-string)

; next : -> char?
(define/public (next)
```

```
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your string-iterator% object will be instantiated and called as such:
;; (define obj (new string-iterator% [compressed-string compressed-string]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Compressed String Iterator
* 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 StringIterator {
public:
StringIterator(string compressedString) {

}

char next() {

}

bool hasNext() {

}
};

/**
```

```
* Your StringIterator object will be instantiated and called as such:
* StringIterator* obj = new StringIterator(compressedString);
* char param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java Solution:

```
/**
* Problem: Design Compressed String Iterator
* 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 StringIterator {

public StringIterator(String compressedString) {

}

public char next() {

}

public boolean hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = new StringIterator(compressedString);
* char param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3 Solution:

```
"""
Problem: Design Compressed String Iterator
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 StringIterator:

def __init__(self, compressedString: str):



def next(self) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class StringIterator(object):

def __init__(self, compressedString):
"""
:type compressedString: str
"""



def next(self):
"""
:rtype: str
"""



def hasNext(self):
"""
:rtype: bool
"""
```

```
# Your StringIterator object will be instantiated and called as such:
# obj = StringIterator(compressedString)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript Solution:

```
/**
* Problem: Design Compressed String Iterator
* 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} compressedString
*/
var StringIterator = function(compressedString) {

};

/**
* @return {character}
*/
StringIterator.prototype.next = function() {

};

/**
* @return {boolean}
*/
StringIterator.prototype.hasNext = function() {

};

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = new StringIterator(compressedString)
```

```
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Compressed String Iterator
* 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 StringIterator {
constructor(compressedString: string) {

}

next(): string {

}

hasNext(): boolean {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = new StringIterator(compressedString)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

C# Solution:

```
/*
* Problem: Design Compressed String Iterator
```

```
* 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 StringIterator {

public StringIterator(string compressedString) {

}

public char Next() {

}

public bool HasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = new StringIterator(compressedString);
* char param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C Solution:

```
/*
* Problem: Design Compressed String Iterator
* 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
*/
```

```
typedef struct {

} StringIterator;


StringIterator* stringIteratorCreate(char* compressedString) {

}

char stringIteratorNext(StringIterator* obj) {

}

bool stringIteratorHasNext(StringIterator* obj) {

}

void stringIteratorFree(StringIterator* obj) {

}

/**
* Your StringIterator struct will be instantiated and called as such:
* StringIterator* obj = stringIteratorCreate(compressedString);
* char param_1 = stringIteratorNext(obj);

* bool param_2 = stringIteratorHasNext(obj);

* stringIteratorFree(obj);
*/
```

Go Solution:

```
// Problem: Design Compressed String Iterator
// 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

type StringIterator struct {

}


func Constructor(compressedString string) StringIterator {

}


func (this *StringIterator) Next() byte {

}


func (this *StringIterator) HasNext() bool {

}


/**
* Your StringIterator object will be instantiated and called as such:
* obj := Constructor(compressedString);
* param_1 := obj.Next();
* param_2 := obj.HasNext();
*/
```

Kotlin Solution:

```
class StringIterator(compressedString: String) {

fun next(): Char {

}

fun hasNext(): Boolean {
```

```
}

}

/**
* Your StringIterator object will be instantiated and called as such:
* var obj = StringIterator(compressedString)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift Solution:

```
class StringIterator {

init(_ compressedString: String) {

}

func next() -> Character {

}

func hasNext() -> Bool {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* let obj = StringIterator(compressedString)
* let ret_1: Character = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust Solution:

```
// Problem: Design Compressed String Iterator
// 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

struct StringIterator {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl StringIterator {

fn new(compressedString: String) -> Self {

}

fn next(&self) -> char {

}

fn has_next(&self) -> bool {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* let obj = StringIterator::new(compressedString);
* let ret_1: char = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby Solution:

```
class StringIterator

=begin
```

```
:type compressed_string: String
=end
def initialize(compressed_string)

end


=begin
:rtype: Character
=end
def next()

end


=begin
:rtype: Boolean
=end
def has_next()

end


end

# Your StringIterator object will be instantiated and called as such:
# obj = StringIterator.new(compressed_string)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP Solution:

```
class StringIterator {
/**
* @param String $compressedString
*/
function __construct($compressedString) {

}

/**
```

```
* @return String
*/
function next() {

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* $obj = StringIterator($compressedString);
* $ret_1 = $obj->next();
* $ret_2 = $obj->hasNext();
*/
```

Dart Solution:

```
class StringIterator {

StringIterator(String compressedString) {

}

String next() {

}

bool hasNext() {

}
}

/**
* Your StringIterator object will be instantiated and called as such:
* StringIterator obj = StringIterator(compressedString);
```

```
* String param1 = obj.next();
* bool param2 = obj.hasNext();
*/
```

Scala Solution:

```
class StringIterator(_compressedString: String) {

def next(): Char = {

}

def hasNext(): Boolean = {

}

}

/**
* Your StringIterator object will be instantiated and called as such:
* val obj = new StringIterator(compressedString)
* val param_1 = obj.next()
* val param_2 = obj.hasNext()
*/
```

Elixir Solution:

```
defmodule StringIterator do
@spec init_(compressed_string :: String.t) :: any
def init_(compressed_string) do

end

@spec next() :: char
def next() do

end

@spec has_next() :: boolean
def has_next() do
```

```
end
end

# Your functions will be called as such:
# StringIterator.init_(compressed_string)
# param_1 = StringIterator.next()
# param_2 = StringIterator.has_next()

# StringIterator.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang Solution:

```
-spec string_iterator_init_(CompressedString :: unicode:unicode_binary()) ->
any().
string_iterator_init_(CompressedString) ->
.

-spec string_iterator_next() -> char().
string_iterator_next() ->
.

-spec string_iterator_has_next() -> boolean().
string_iterator_has_next() ->
.


%% Your functions will be called as such:
%% string_iterator_init_(CompressedString),
%% Param_1 = string_iterator_next(),
%% Param_2 = string_iterator_has_next(),

%% string_iterator_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
(define string-iterator%
(class object%
(super-new)
```

```
; compressed-string : string?
(init-field
compressed-string)

; next : -> char?
(define/public (next)
)
; has-next : -> boolean?
(define/public (has-next)
)))

;; Your string-iterator% object will be instantiated and called as such:
;; (define obj (new string-iterator% [compressed-string compressed-string]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```