Problem 1286: Iterator for Combination

Problem Information

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

Problem Description

Design the

CombinationIterator

class:

CombinationIterator(string characters, int combinationLength)

Initializes the object with a string

characters

of

sorted distinct

lowercase English letters and a number

combinationLength

as arguments.

next()

Returns the next combination of length

combinationLength

in

lexicographical order

.

hasNext()

Returns

true

if and only if there exists a next combination.

Example 1:

Input

["CombinationIterator", "next", "hasNext", "next", "hasNext", "next", "hasNext"] [["abc", 2], [], [], [], [], [], []]

Output

[null, "ab", true, "ac", true, "bc", false]

Explanation

CombinationIterator itr = new CombinationIterator("abc", 2); itr.next(); // return "ab" itr.hasNext(); // return True itr.next(); // return "ac" itr.hasNext(); // return True itr.next(); // return "bc" itr.hasNext(); // return False

Constraints:

1 <= combinationLength <= characters.length <= 15

All the characters of

characters

are

unique

.

At most

10

4

calls will be made to

next

and

hasNext

.

It is guaranteed that all calls of the function

next

are valid.

Code Snippets

C++:

```
class CombinationIterator {
public:
CombinationIterator(string characters, int combinationLength) {
```

```
}

string next() {

}

bool hasNext() {

}
};

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator* obj = new CombinationIterator(characters,
combinationLength);
* string param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java:

```
class CombinationIterator {

public CombinationIterator(String characters, int combinationLength) {

}

public String next() {

}

public boolean hasNext() {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator obj = new CombinationIterator(characters,
combinationLength);
* String param_1 = obj.next();
* boolean param_2 = obj.hasNext();
```

```
*/
```

Python3:

```
class CombinationIterator:

def __init__(self, characters: str, combinationLength: int):


def next(self) -> str:


def hasNext(self) -> bool:



# Your CombinationIterator object will be instantiated and called as such:
# obj = CombinationIterator(characters, combinationLength)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

Python:

```
class CombinationIterator(object):

def __init__(self, characters, combinationLength):
"""
:type characters: str
:type combinationLength: int
"""


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


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

```
# Your CombinationIterator object will be instantiated and called as such:
# obj = CombinationIterator(characters, combinationLength)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript:

```
/**
* @param {string} characters
* @param {number} combinationLength
*/
var CombinationIterator = function(characters, combinationLength) {

};

/**
* @return {string}
*/
CombinationIterator.prototype.next = function() {

};

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

};

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

TypeScript:

```
class CombinationIterator {
constructor(characters: string, combinationLength: number) {

}

next(): string {

}

hasNext(): boolean {

}
}

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

C#:

```
public class CombinationIterator {

public CombinationIterator(string characters, int combinationLength) {

}

public string Next() {

}

public bool HasNext() {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator obj = new CombinationIterator(characters,
combinationLength);
* string param_1 = obj.Next();
```

```
* bool param_2 = obj.HasNext();
*/
```

C:

```
typedef struct {

} CombinationIterator;


CombinationIterator* combinationIteratorCreate(char* characters, int
combinationLength) {

}

char* combinationIteratorNext(CombinationIterator* obj) {

}

bool combinationIteratorHasNext(CombinationIterator* obj) {

}

void combinationIteratorFree(CombinationIterator* obj) {

}

/**
* Your CombinationIterator struct will be instantiated and called as such:
* CombinationIterator* obj = combinationIteratorCreate(characters,
combinationLength);
* char* param_1 = combinationIteratorNext(obj);

* bool param_2 = combinationIteratorHasNext(obj);

* combinationIteratorFree(obj);
*/
```

Go:

```
type CombinationIterator struct {

}


func Constructor(characters string, combinationLength int)
CombinationIterator {

}


func (this *CombinationIterator) Next() string {

}


func (this *CombinationIterator) HasNext() bool {

}


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

Kotlin:

```
class CombinationIterator(characters: String, combinationLength: Int) {

fun next(): String {

}

fun hasNext(): Boolean {

}

}
```

```
/**
* Your CombinationIterator object will be instantiated and called as such:
* var obj = CombinationIterator(characters, combinationLength)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift:

```
class CombinationIterator {

init(_ characters: String, _ combinationLength: Int) {

}

func next() -> String {

}

func hasNext() -> Bool {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* let obj = CombinationIterator(characters, combinationLength)
* let ret_1: String = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust:

```
struct CombinationIterator {

}


/**
* `&self` means the method takes an immutable reference.
```

```
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl CombinationIterator {

fn new(characters: String, combinationLength: i32) -> Self {

}

fn next(&self) -> String {

}

fn has_next(&self) -> bool {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* let obj = CombinationIterator::new(characters, combinationLength);
* let ret_1: String = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby:

```
class CombinationIterator

=begin
:type characters: String
:type combination_length: Integer
=end
def initialize(characters, combination_length)

end


=begin
:rtype: String
=end
def next()
```

```
end


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

end


end

# Your CombinationIterator object will be instantiated and called as such:
# obj = CombinationIterator.new(characters, combination_length)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP:

```
class CombinationIterator {
/**
* @param String $characters
* @param Integer $combinationLength
*/
function __construct($characters, $combinationLength) {

}

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

}

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

}
```

```
}

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

Dart:

```
class CombinationIterator {

CombinationIterator(String characters, int combinationLength) {

}

String next() {

}

bool hasNext() {

}
}

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

Scala:

```
class CombinationIterator(_characters: String, _combinationLength: Int) {

def next(): String = {

}
```

```
def hasNext(): Boolean = {

}

}

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

Elixir:

```
defmodule CombinationIterator do
@spec init_(characters :: String.t, combination_length :: integer) :: any
def init_(characters, combination_length) do

end

@spec next() :: String.t
def next() do

end

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

end
end

# Your functions will be called as such:
# CombinationIterator.init_(characters, combination_length)
# param_1 = CombinationIterator.next()
# param_2 = CombinationIterator.has_next()

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

Erlang:

```
-spec combination_iterator_init_(Characters :: unicode:unicode_binary(),
CombinationLength :: integer()) -> any().
combination_iterator_init_(Characters, CombinationLength) ->
.

-spec combination_iterator_next() -> unicode:unicode_binary().
combination_iterator_next() ->
.

-spec combination_iterator_has_next() -> boolean().
combination_iterator_has_next() ->
.


%% Your functions will be called as such:
%% combination_iterator_init_(Characters, CombinationLength),
%% Param_1 = combination_iterator_next(),
%% Param_2 = combination_iterator_has_next(),

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

Racket:

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

; characters : string?
; combination-length : exact-integer?
(init-field
characters
combination-length)

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

;; Your combination-iterator% object will be instantiated and called as such:
;; (define obj (new combination-iterator% [characters characters]
```

```
[combination-length combination-length]))
;; (define param_1 (send obj next))
;; (define param_2 (send obj has-next))
```

Solutions

C++ Solution:

```
/*
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 CombinationIterator {
public:
CombinationIterator(string characters, int combinationLength) {

}

string next() {

}

bool hasNext() {

}
};

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator* obj = new CombinationIterator(characters,
combinationLength);
* string param_1 = obj->next();
* bool param_2 = obj->hasNext();
*/
```

Java Solution:

```
/**
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 CombinationIterator {

public CombinationIterator(String characters, int combinationLength) {

}

public String next() {

}

public boolean hasNext() {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator obj = new CombinationIterator(characters,
combinationLength);
* String param_1 = obj.next();
* boolean param_2 = obj.hasNext();
*/
```

Python3 Solution:

```
"""
Problem: Iterator for Combination
Difficulty: Medium
Tags: string, graph, sort

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 CombinationIterator:

def __init__(self, characters: str, combinationLength: int):


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

Python Solution:

```
class CombinationIterator(object):

def __init__(self, characters, combinationLength):
"""
:type characters: str
:type combinationLength: int
"""



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



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




# Your CombinationIterator object will be instantiated and called as such:
# obj = CombinationIterator(characters, combinationLength)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

JavaScript Solution:

```javascript
/**
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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} characters
* @param {number} combinationLength
*/
var CombinationIterator = function(characters, combinationLength) {

};

/**
* @return {string}
*/
CombinationIterator.prototype.next = function() {

};

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

};

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

TypeScript Solution:

```
/**
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 CombinationIterator {
constructor(characters: string, combinationLength: number) {

}

next(): string {

}

hasNext(): boolean {

}
}

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

C# Solution:

```
/*
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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 CombinationIterator {

public CombinationIterator(string characters, int combinationLength) {

}

public string Next() {

}

public bool HasNext() {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator obj = new CombinationIterator(characters,
combinationLength);
* string param_1 = obj.Next();
* bool param_2 = obj.HasNext();
*/
```

C Solution:

```
/*
* Problem: Iterator for Combination
* Difficulty: Medium
* Tags: string, graph, sort
*
* 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
*/




typedef struct {
```

```
} CombinationIterator;


CombinationIterator* combinationIteratorCreate(char* characters, int
combinationLength) {

}

char* combinationIteratorNext(CombinationIterator* obj) {

}

bool combinationIteratorHasNext(CombinationIterator* obj) {

}

void combinationIteratorFree(CombinationIterator* obj) {

}

/**
* Your CombinationIterator struct will be instantiated and called as such:
* CombinationIterator* obj = combinationIteratorCreate(characters,
combinationLength);
* char* param_1 = combinationIteratorNext(obj);

* bool param_2 = combinationIteratorHasNext(obj);

* combinationIteratorFree(obj);
*/
```

Go Solution:

```
// Problem: Iterator for Combination
// Difficulty: Medium
// Tags: string, graph, sort
//
// 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
```

```
type CombinationIterator struct {

}


func Constructor(characters string, combinationLength int)
CombinationIterator {

}


func (this *CombinationIterator) Next() string {

}


func (this *CombinationIterator) HasNext() bool {

}


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

Kotlin Solution:

```
class CombinationIterator(characters: String, combinationLength: Int) {

fun next(): String {

}

fun hasNext(): Boolean {

}

}
```

```
/**
* Your CombinationIterator object will be instantiated and called as such:
* var obj = CombinationIterator(characters, combinationLength)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift Solution:

```
class CombinationIterator {

init(_ characters: String, _ combinationLength: Int) {

}

func next() -> String {

}

func hasNext() -> Bool {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* let obj = CombinationIterator(characters, combinationLength)
* let ret_1: String = obj.next()
* let ret_2: Bool = obj.hasNext()
*/
```

Rust Solution:

```
// Problem: Iterator for Combination
// Difficulty: Medium
// Tags: string, graph, sort
//
// 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

struct CombinationIterator {

}


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

fn new(characters: String, combinationLength: i32) -> Self {

}

fn next(&self) -> String {

}

fn has_next(&self) -> bool {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* let obj = CombinationIterator::new(characters, combinationLength);
* let ret_1: String = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby Solution:

```
class CombinationIterator

=begin
:type characters: String
:type combination_length: Integer
=end
```

```
def initialize(characters, combination_length)

end


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

end


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

end


end

# Your CombinationIterator object will be instantiated and called as such:
# obj = CombinationIterator.new(characters, combination_length)
# param_1 = obj.next()
# param_2 = obj.has_next()
```

PHP Solution:

```
class CombinationIterator {
/**
* @param String $characters
* @param Integer $combinationLength
*/
function __construct($characters, $combinationLength) {

}

/**
* @return String
```

```
*/
function next() {

}

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

}
}

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

Dart Solution:

```
class CombinationIterator {

CombinationIterator(String characters, int combinationLength) {

}

String next() {

}

bool hasNext() {

}
}

/**
* Your CombinationIterator object will be instantiated and called as such:
* CombinationIterator obj = CombinationIterator(characters,
combinationLength);
```

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

Scala Solution:

```
class CombinationIterator(_characters: String, _combinationLength: Int) {

def next(): String = {

}

def hasNext(): Boolean = {

}

}

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

Elixir Solution:

```
defmodule CombinationIterator do
@spec init_(characters :: String.t, combination_length :: integer) :: any
def init_(characters, combination_length) do

end

@spec next() :: String.t
def next() do

end

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

```
end
end

# Your functions will be called as such:
# CombinationIterator.init_(characters, combination_length)
# param_1 = CombinationIterator.next()
# param_2 = CombinationIterator.has_next()

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

Erlang Solution:

```
-spec combination_iterator_init_(Characters :: unicode:unicode_binary(),
CombinationLength :: integer()) -> any().
combination_iterator_init_(Characters, CombinationLength) ->
.

-spec combination_iterator_next() -> unicode:unicode_binary().
combination_iterator_next() ->
.

-spec combination_iterator_has_next() -> boolean().
combination_iterator_has_next() ->
.


%% Your functions will be called as such:
%% combination_iterator_init_(Characters, CombinationLength),
%% Param_1 = combination_iterator_next(),
%% Param_2 = combination_iterator_has_next(),

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

Racket Solution:

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

```
; characters : string?
; combination-length : exact-integer?
(init-field
characters
combination-length)

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

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