Problem 2336: Smallest Number in Infinite Set

Problem Information

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

Problem Description

You have a set which contains all positive integers

[1, 2, 3, 4, 5, ...]

.

Implement the

SmallestInfiniteSet

class:

SmallestInfiniteSet()

Initializes the

SmallestInfiniteSet

object to contain

all

positive integers.

int popSmallest()

Removes

and returns the smallest integer contained in the infinite set.

void addBack(int num)

Adds

a positive integer

num

back into the infinite set, if it is

not

already in the infinite set.

Example 1:

Input

["SmallestInfiniteSet", "addBack", "popSmallest", "popSmallest", "popSmallest", "addBack", "popSmallest", "popSmallest", "popSmallest"] [[], [2], [], [], [], [1], [], [], []]

Output

[null, null, 1, 2, 3, null, 1, 4, 5]

Explanation

SmallestInfiniteSet smallestInfiniteSet = new SmallestInfiniteSet(); smallestInfiniteSet.addBack(2); // 2 is already in the set, so no change is made. smallestInfiniteSet.popSmallest(); // return 1, since 1 is the smallest number, and remove it from the set. smallestInfiniteSet.popSmallest(); // return 2, and remove it from the set. smallestInfiniteSet.popSmallest(); // return 3, and remove it from the set. smallestInfiniteSet.addBack(1); // 1 is added back to the set. smallestInfiniteSet.popSmallest(); // return 1, since 1 was added back to the set and // is the smallest number, and remove it from the set. smallestInfiniteSet.popSmallest(); // return 4, and remove it from the set. smallestInfiniteSet.popSmallest(); // return 5, and remove it from the set.

Constraints:

1 <= num <= 1000

At most

1000

calls will be made

in total

to

popSmallest

and

addBack

.

Code Snippets

C++:

```
class SmallestInfiniteSet {
public:
SmallestInfiniteSet() {

}

int popSmallest() {

}

void addBack(int num) {
}
};

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet* obj = new SmallestInfiniteSet();
* int param_1 = obj->popSmallest();
* obj->addBack(num);
*/
```

Java:

```
class SmallestInfiniteSet {

public SmallestInfiniteSet() {

}

public int popSmallest() {

}

public void addBack(int num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = new SmallestInfiniteSet();
* int param_1 = obj.popSmallest();
* obj.addBack(num);
*/
```

Python3:

```
class SmallestInfiniteSet:

def __init__(self):
```

```
def popSmallest(self) -> int:


def addBack(self, num: int) -> None:



# Your SmallestInfiniteSet object will be instantiated and called as such:
# obj = SmallestInfiniteSet()
# param_1 = obj.popSmallest()
# obj.addBack(num)
```

Python:

```
class SmallestInfiniteSet(object):

def __init__(self):


def popSmallest(self):
"""
:rtype: int
"""


def addBack(self, num):
"""
:type num: int
:rtype: None
"""



# Your SmallestInfiniteSet object will be instantiated and called as such:
# obj = SmallestInfiniteSet()
# param_1 = obj.popSmallest()
# obj.addBack(num)
```

JavaScript:

```
var SmallestInfiniteSet = function() {
};

/**
* @return {number}
*/
SmallestInfiniteSet.prototype.popSmallest = function() {

};

/**
* @param {number} num
* @return {void}
*/
SmallestInfiniteSet.prototype.addBack = function(num) {

};

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = new SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

TypeScript:

```
class SmallestInfiniteSet {
constructor() {

}

popSmallest(): number {

}

addBack(num: number): void {

}
}

/**
```

```
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = new SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

C#:

```
public class SmallestInfiniteSet {

public SmallestInfiniteSet() {

}

public int PopSmallest() {

}

public void AddBack(int num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = new SmallestInfiniteSet();
* int param_1 = obj.PopSmallest();
* obj.AddBack(num);
*/
```

C:

```



typedef struct {

} SmallestInfiniteSet;


SmallestInfiniteSet* smallestInfiniteSetCreate() {
}

int smallestInfiniteSetPopSmallest(SmallestInfiniteSet* obj) {

}

void smallestInfiniteSetAddBack(SmallestInfiniteSet* obj, int num) {

}

void smallestInfiniteSetFree(SmallestInfiniteSet* obj) {

}

/**
* Your SmallestInfiniteSet struct will be instantiated and called as such:
* SmallestInfiniteSet* obj = smallestInfiniteSetCreate();
* int param_1 = smallestInfiniteSetPopSmallest(obj);

* smallestInfiniteSetAddBack(obj, num);

* smallestInfiniteSetFree(obj);
*/
```

Go:

```
type SmallestInfiniteSet struct {

}


func Constructor() SmallestInfiniteSet {

}


func (this *SmallestInfiniteSet) PopSmallest() int {

}


func (this *SmallestInfiniteSet) AddBack(num int) {
}


/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.PopSmallest();
* obj.AddBack(num);
*/
```

Kotlin:

```
class SmallestInfiniteSet() {

fun popSmallest(): Int {

}

fun addBack(num: Int) {

}

}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

Swift:

```
class SmallestInfiniteSet {

init() {

}

func popSmallest() -> Int {
}

func addBack(_ num: Int) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* let obj = SmallestInfiniteSet()
* let ret_1: Int = obj.popSmallest()
* obj.addBack(num)
*/
```

Rust:

```
struct SmallestInfiniteSet {

}



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

fn new() -> Self {

}

fn pop_smallest(&self) -> i32 {

}

fn add_back(&self, num: i32) {

}
}

/**
```

```
* Your SmallestInfiniteSet object will be instantiated and called as such:
* let obj = SmallestInfiniteSet::new();
* let ret_1: i32 = obj.pop_smallest();
* obj.add_back(num);
*/
```

Ruby:

```
class SmallestInfiniteSet
def initialize()

end



=begin
:rtype: Integer
=end
def pop_smallest()

end



=begin
:type num: Integer
:rtype: Void
=end
def add_back(num)

end



end

# Your SmallestInfiniteSet object will be instantiated and called as such:
# obj = SmallestInfiniteSet.new()
# param_1 = obj.pop_smallest()
# obj.add_back(num)
```

PHP:

```
class SmallestInfiniteSet {
/**
*/
function __construct() {

}

/**
* @return Integer
*/
function popSmallest() {

}

/**
* @param Integer $num
* @return NULL
*/
function addBack($num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* $obj = SmallestInfiniteSet();
* $ret_1 = $obj->popSmallest();
* $obj->addBack($num);
*/
```

Dart:

```
class SmallestInfiniteSet {

SmallestInfiniteSet() {

}

int popSmallest() {

}

void addBack(int num) {
}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = SmallestInfiniteSet();
* int param1 = obj.popSmallest();
* obj.addBack(num);
*/
```

Scala:

```
class SmallestInfiniteSet() {

def popSmallest(): Int = {

}

def addBack(num: Int): Unit = {

}

}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* val obj = new SmallestInfiniteSet()
* val param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

Elixir:

```
defmodule SmallestInfiniteSet do
@spec init_() :: any
def init_() do

end

@spec pop_smallest() :: integer
def pop_smallest() do
```

```
end

@spec add_back(num :: integer) :: any
def add_back(num) do

end
end

# Your functions will be called as such:
# SmallestInfiniteSet.init_()
# param_1 = SmallestInfiniteSet.pop_smallest()
# SmallestInfiniteSet.add_back(num)

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

Erlang:

```
-spec smallest_infinite_set_init_() -> any().
smallest_infinite_set_init_() ->
.

-spec smallest_infinite_set_pop_smallest() -> integer().
smallest_infinite_set_pop_smallest() ->
.

-spec smallest_infinite_set_add_back(Num :: integer()) -> any().
smallest_infinite_set_add_back(Num) ->
.


%% Your functions will be called as such:
%% smallest_infinite_set_init_(),
%% Param_1 = smallest_infinite_set_pop_smallest(),
%% smallest_infinite_set_add_back(Num),

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

Racket:

```
(define smallest-infinite-set%
(class object%
(super-new)

(init-field)

; pop-smallest : -> exact-integer?
(define/public (pop-smallest)
)
; add-back : exact-integer? -> void?
(define/public (add-back num)
)))

;; Your smallest-infinite-set% object will be instantiated and called as
such:
;; (define obj (new smallest-infinite-set%))
;; (define param_1 (send obj pop-smallest))
;; (send obj add-back num)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SmallestInfiniteSet {
public:
SmallestInfiniteSet() {

}

int popSmallest() {
}

void addBack(int num) {

}
};

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet* obj = new SmallestInfiniteSet();
* int param_1 = obj->popSmallest();
* obj->addBack(num);
*/
```

Java Solution:

```
/**
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SmallestInfiniteSet {

public SmallestInfiniteSet() {

}

public int popSmallest() {

}

public void addBack(int num) {

}
}
```

```
/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = new SmallestInfiniteSet();
* int param_1 = obj.popSmallest();
* obj.addBack(num);
*/
```

Python3 Solution:

```
"""
Problem: Smallest Number in Infinite Set
Difficulty: Medium
Tags: hash, queue, heap

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class SmallestInfiniteSet:

def __init__(self):



def popSmallest(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SmallestInfiniteSet(object):

def __init__(self):



def popSmallest(self):
"""
:rtype: int
"""
```

```
def addBack(self, num):
"""
:type num: int
:rtype: None
"""



# Your SmallestInfiniteSet object will be instantiated and called as such:
# obj = SmallestInfiniteSet()
# param_1 = obj.popSmallest()
# obj.addBack(num)
```

JavaScript Solution:

```
/**
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/


var SmallestInfiniteSet = function() {

};

/**
* @return {number}
*/
SmallestInfiniteSet.prototype.popSmallest = function() {

};

/**
* @param {number} num
* @return {void}
*/
```

```
SmallestInfiniteSet.prototype.addBack = function(num) {

};

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = new SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

TypeScript Solution:

```
/**
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SmallestInfiniteSet {
constructor() {

}

popSmallest(): number {

}

addBack(num: number): void {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = new SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

C# Solution:

```
/*
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class SmallestInfiniteSet {

public SmallestInfiniteSet() {

}

public int PopSmallest() {

}

public void AddBack(int num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = new SmallestInfiniteSet();
* int param_1 = obj.PopSmallest();
* obj.AddBack(num);
*/
```

C Solution:

```
/*
* Problem: Smallest Number in Infinite Set
* Difficulty: Medium
* Tags: hash, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {

} SmallestInfiniteSet;


SmallestInfiniteSet* smallestInfiniteSetCreate() {

}

int smallestInfiniteSetPopSmallest(SmallestInfiniteSet* obj) {

}

void smallestInfiniteSetAddBack(SmallestInfiniteSet* obj, int num) {

}

void smallestInfiniteSetFree(SmallestInfiniteSet* obj) {

}

/**
* Your SmallestInfiniteSet struct will be instantiated and called as such:
* SmallestInfiniteSet* obj = smallestInfiniteSetCreate();
* int param_1 = smallestInfiniteSetPopSmallest(obj);

* smallestInfiniteSetAddBack(obj, num);

* smallestInfiniteSetFree(obj);
*/
```

Go Solution:

```go
// Problem: Smallest Number in Infinite Set
// Difficulty: Medium
// Tags: hash, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type SmallestInfiniteSet struct {

}


func Constructor() SmallestInfiniteSet {

}


func (this *SmallestInfiniteSet) PopSmallest() int {

}


func (this *SmallestInfiniteSet) AddBack(num int) {

}


/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.PopSmallest();
* obj.AddBack(num);
*/
```

Kotlin Solution:

```kotlin
class SmallestInfiniteSet() {

fun popSmallest(): Int {
}

fun addBack(num: Int) {

}

}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* var obj = SmallestInfiniteSet()
* var param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

Swift Solution:

```
class SmallestInfiniteSet {

init() {

}

func popSmallest() -> Int {

}

func addBack(_ num: Int) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* let obj = SmallestInfiniteSet()
* let ret_1: Int = obj.popSmallest()
* obj.addBack(num)
*/
```

Rust Solution:

```
// Problem: Smallest Number in Infinite Set
// Difficulty: Medium
// Tags: hash, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

struct SmallestInfiniteSet {

}



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

fn new() -> Self {

}

fn pop_smallest(&self) -> i32 {

}

fn add_back(&self, num: i32) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* let obj = SmallestInfiniteSet::new();
* let ret_1: i32 = obj.pop_smallest();
* obj.add_back(num);
*/
```

Ruby Solution:

```
class SmallestInfiniteSet
def initialize()

end



=begin
:rtype: Integer
=end
def pop_smallest()

end



=begin
:type num: Integer
:rtype: Void
=end
def add_back(num)

end



end

# Your SmallestInfiniteSet object will be instantiated and called as such:
# obj = SmallestInfiniteSet.new()
# param_1 = obj.pop_smallest()
# obj.add_back(num)
```

PHP Solution:

```
class SmallestInfiniteSet {
/**
*/
function __construct() {

}

/**
* @return Integer
*/
function popSmallest() {

}

/**
* @param Integer $num
* @return NULL
*/
function addBack($num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* $obj = SmallestInfiniteSet();
* $ret_1 = $obj->popSmallest();
* $obj->addBack($num);
*/
```

Dart Solution:

```
class SmallestInfiniteSet {

SmallestInfiniteSet() {

}

int popSmallest() {

}

void addBack(int num) {

}
}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* SmallestInfiniteSet obj = SmallestInfiniteSet();
* int param1 = obj.popSmallest();
* obj.addBack(num);
*/
```

Scala Solution:

```
class SmallestInfiniteSet() {

def popSmallest(): Int = {

}

def addBack(num: Int): Unit = {

}

}

/**
* Your SmallestInfiniteSet object will be instantiated and called as such:
* val obj = new SmallestInfiniteSet()
* val param_1 = obj.popSmallest()
* obj.addBack(num)
*/
```

Elixir Solution:

```
defmodule SmallestInfiniteSet do
@spec init_() :: any
def init_() do

end

@spec pop_smallest() :: integer
def pop_smallest() do

end

@spec add_back(num :: integer) :: any
def add_back(num) do

end
```

```
end

# Your functions will be called as such:
# SmallestInfiniteSet.init_()
# param_1 = SmallestInfiniteSet.pop_smallest()
# SmallestInfiniteSet.add_back(num)

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

Erlang Solution:

```
-spec smallest_infinite_set_init_() -> any().
smallest_infinite_set_init_() ->
.

-spec smallest_infinite_set_pop_smallest() -> integer().
smallest_infinite_set_pop_smallest() ->
.

-spec smallest_infinite_set_add_back(Num :: integer()) -> any().
smallest_infinite_set_add_back(Num) ->
.



%% Your functions will be called as such:
%% smallest_infinite_set_init_(),
%% Param_1 = smallest_infinite_set_pop_smallest(),
%% smallest_infinite_set_add_back(Num),

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

Racket Solution:

```
(define smallest-infinite-set%
(class object%
(super-new)

(init-field)
; pop-smallest : -> exact-integer?
(define/public (pop-smallest)
)
; add-back : exact-integer? -> void?
(define/public (add-back num)
)))

;; Your smallest-infinite-set% object will be instantiated and called as
such:
;; (define obj (new smallest-infinite-set%))
;; (define param_1 (send obj pop-smallest))
;; (send obj add-back num)
```