Problem 1206: Design Skiplist

Problem Information

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

Problem Description

Design a

Skiplist

without using any built-in libraries.

A

skiplist

is a data structure that takes

O(log(n))

time to add, erase and search. Comparing with treap and red-black tree which has the same function and performance, the code length of Skiplist can be comparatively short and the idea behind Skiplists is just simple linked lists.

For example, we have a Skiplist containing

[30,40,50,60,70,90]

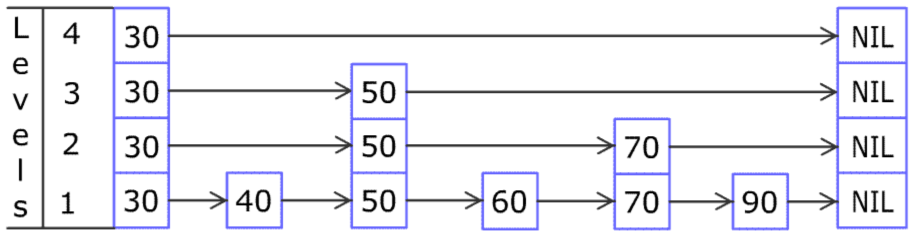
and we want to add

80

and

45

into it. The Skiplist works this way:



Artyom Kalinin [CC BY-SA 3.0], via

Wikimedia Commons

You can see there are many layers in the Skiplist. Each layer is a sorted linked list. With the help of the top layers, add, erase and search can be faster than

O(n)

. It can be proven that the average time complexity for each operation is

O(log(n))

and space complexity is

O(n)

.

See more about Skiplist:

https://en.wikipedia.org/wiki/Skip_list

Implement the

Skiplist

class:

Skiplist()

Initializes the object of the skiplist.

bool search(int target)

Returns

true

if the integer

target

exists in the Skiplist or

false

otherwise.

void add(int num)

Inserts the value

num

into the SkipList.

bool erase(int num)

Removes the value

num

from the Skiplist and returns

true

. If

num

does not exist in the Skiplist, do nothing and return

false

. If there exist multiple

num

values, removing any one of them is fine.

Note that duplicates may exist in the Skiplist, your code needs to handle this situation.

Example 1:

Input

["Skiplist", "add", "add", "add", "search", "add", "search", "erase", "erase", "search"] [[], [1], [2], [3], [0], [4], [1], [0], [1], [1]]

Output

[null, null, null, null, false, null, true, false, true, false]

Explanation

Skiplist skiplist = new Skiplist(); skiplist.add(1); skiplist.add(2); skiplist.add(3); skiplist.search(0); // return False skiplist.add(4); skiplist.search(1); // return True skiplist.erase(0); // return False, 0 is not in skiplist. skiplist.erase(1); // return True skiplist.search(1); // return False, 1 has already been erased.

Constraints:

0 <= num, target <= 2 * 10

4

At most

5 * 10

4

calls will be made to

search

,

add

, and

erase

.

Code Snippets

C++:

```
class Skiplist {
public:
Skiplist() {

}

bool search(int target) {

}

void add(int num) {

}
bool erase(int num) {

}
};

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist* obj = new Skiplist();
* bool param_1 = obj->search(target);
* obj->add(num);
* bool param_3 = obj->erase(num);
*/
```

Java:

```
class Skiplist {

public Skiplist() {

}

public boolean search(int target) {

}

public void add(int num) {

}

public boolean erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = new Skiplist();
* boolean param_1 = obj.search(target);
* obj.add(num);
* boolean param_3 = obj.erase(num);
*/
```

Python3:

```
class Skiplist:

def __init__(self):



def search(self, target: int) -> bool:



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



def erase(self, num: int) -> bool:




# Your Skiplist object will be instantiated and called as such:
# obj = Skiplist()
# param_1 = obj.search(target)
# obj.add(num)
# param_3 = obj.erase(num)
```

Python:

```
class Skiplist(object):

def __init__(self):



def search(self, target):
"""
:type target: int
:rtype: bool
"""



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

```
def erase(self, num):
"""
:type num: int
:rtype: bool
"""



# Your Skiplist object will be instantiated and called as such:
# obj = Skiplist()
# param_1 = obj.search(target)
# obj.add(num)
# param_3 = obj.erase(num)
```

JavaScript:

```
var Skiplist = function() {

};

/**
* @param {number} target
* @return {boolean}
*/
Skiplist.prototype.search = function(target) {

};

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

};

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

```
*/
Skiplist.prototype.erase = function(num) {

};

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = new Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

TypeScript:

```
class Skiplist {
constructor() {

}

search(target: number): boolean {

}

add(num: number): void {

}

erase(num: number): boolean {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = new Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

C#:

```
public class Skiplist {

public Skiplist() {

}

public bool Search(int target) {

}

public void Add(int num) {

}

public bool Erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = new Skiplist();
* bool param_1 = obj.Search(target);
* obj.Add(num);
* bool param_3 = obj.Erase(num);
*/
```

C:

```



typedef struct {

} Skiplist;


Skiplist* skiplistCreate() {

}
```

```
bool skiplistSearch(Skiplist* obj, int target) {

}

void skiplistAdd(Skiplist* obj, int num) {

}

bool skiplistErase(Skiplist* obj, int num) {

}

void skiplistFree(Skiplist* obj) {

}

/**
* Your Skiplist struct will be instantiated and called as such:
* Skiplist* obj = skiplistCreate();
* bool param_1 = skiplistSearch(obj, target);

* skiplistAdd(obj, num);

* bool param_3 = skiplistErase(obj, num);

* skiplistFree(obj);
*/
```

Go:

```
type Skiplist struct {

}


func Constructor() Skiplist {

}


func (this *Skiplist) Search(target int) bool {
}


func (this *Skiplist) Add(num int) {

}


func (this *Skiplist) Erase(num int) bool {

}


/**
* Your Skiplist object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Search(target);
* obj.Add(num);
* param_3 := obj.Erase(num);
*/
```

Kotlin:

```
class Skiplist() {

fun search(target: Int): Boolean {

}

fun add(num: Int) {

}

fun erase(num: Int): Boolean {

}

}

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

Swift:

```
class Skiplist {

init() {

}

func search(_ target: Int) -> Bool {

}

func add(_ num: Int) {

}

func erase(_ num: Int) -> Bool {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* let obj = Skiplist()
* let ret_1: Bool = obj.search(target)
* obj.add(num)
* let ret_3: Bool = obj.erase(num)
*/
```

Rust:

```
struct Skiplist {

}
```

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

fn new() -> Self {

}

fn search(&self, target: i32) -> bool {

}

fn add(&self, num: i32) {

}

fn erase(&self, num: i32) -> bool {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* let obj = Skiplist::new();
* let ret_1: bool = obj.search(target);
* obj.add(num);
* let ret_3: bool = obj.erase(num);
*/
```

Ruby:

```
class Skiplist
def initialize()

end



=begin
:type target: Integer
:rtype: Boolean
```

```
=end
def search(target)

end



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

end



=begin
:type num: Integer
:rtype: Boolean
=end
def erase(num)

end



end

# Your Skiplist object will be instantiated and called as such:
# obj = Skiplist.new()
# param_1 = obj.search(target)
# obj.add(num)
# param_3 = obj.erase(num)
```

PHP:

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

}

/**
* @param Integer $target
* @return Boolean
*/
function search($target) {

}

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

}

/**
* @param Integer $num
* @return Boolean
*/
function erase($num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* $obj = Skiplist();
* $ret_1 = $obj->search($target);
* $obj->add($num);
* $ret_3 = $obj->erase($num);
*/
```

Dart:

```
class Skiplist {

Skiplist() {

}

bool search(int target) {
}

void add(int num) {

}

bool erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = Skiplist();
* bool param1 = obj.search(target);
* obj.add(num);
* bool param3 = obj.erase(num);
*/
```

Scala:

```
class Skiplist() {

def search(target: Int): Boolean = {

}

def add(num: Int): Unit = {

}

def erase(num: Int): Boolean = {

}

}

/**
* Your Skiplist object will be instantiated and called as such:
* val obj = new Skiplist()
* val param_1 = obj.search(target)
* obj.add(num)
* val param_3 = obj.erase(num)
*/
```

Elixir:

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

end

@spec search(target :: integer) :: boolean
def search(target) do

end

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

end

@spec erase(num :: integer) :: boolean
def erase(num) do

end
end

# Your functions will be called as such:
# Skiplist.init_()
# param_1 = Skiplist.search(target)
# Skiplist.add(num)
# param_3 = Skiplist.erase(num)

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

Erlang:

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

```
-spec skiplist_search(Target :: integer()) -> boolean().
skiplist_search(Target) ->
  .

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

-spec skiplist_erase(Num :: integer()) -> boolean().
skiplist_erase(Num) ->
  .



%% Your functions will be called as such:
%% skiplist_init_(),
%% Param_1 = skiplist_search(Target),
%% skiplist_add(Num),
%% Param_3 = skiplist_erase(Num),

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

Racket:

```
(define skiplist%
(class object%
(super-new)

(init-field)

; search : exact-integer? -> boolean?
(define/public (search target)
)
; add : exact-integer? -> void?
(define/public (add num)
)
; erase : exact-integer? -> boolean?
(define/public (erase num)
)))

;; Your skiplist% object will be instantiated and called as such:
;; (define obj (new skiplist%))
;; (define param_1 (send obj search target))
;; (send obj add num)
;; (define param_3 (send obj erase num))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Skiplist {
public:
Skiplist() {

}

bool search(int target) {

}

void add(int num) {

}

bool erase(int num) {

}
};

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist* obj = new Skiplist();
* bool param_1 = obj->search(target);
* obj->add(num);
* bool param_3 = obj->erase(num);
*/
```

Java Solution:

```
/**
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Skiplist {

public Skiplist() {

}

public boolean search(int target) {

}

public void add(int num) {

}

public boolean erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = new Skiplist();
* boolean param_1 = obj.search(target);
* obj.add(num);
* boolean param_3 = obj.erase(num);
*/
```

Python3 Solution:

```
"""
Problem: Design Skiplist
Difficulty: Hard
Tags: tree, sort, search, linked_list

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

class Skiplist:

def __init__(self):



def search(self, target: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Skiplist(object):

def __init__(self):



def search(self, target):
"""
:type target: int
:rtype: bool
"""



def add(self, num):
"""
:type num: int
```

```
:rtype: None
"""


def erase(self, num):
"""
:type num: int
:rtype: bool
"""



# Your Skiplist object will be instantiated and called as such:
# obj = Skiplist()
# param_1 = obj.search(target)
# obj.add(num)
# param_3 = obj.erase(num)
```

JavaScript Solution:

```
/**
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/


var Skiplist = function() {

};

/**
* @param {number} target
* @return {boolean}
*/
Skiplist.prototype.search = function(target) {

};

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

};

/**
* @param {number} num
* @return {boolean}
*/
Skiplist.prototype.erase = function(num) {

};

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = new Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Skiplist {
constructor() {
}

search(target: number): boolean {

}

add(num: number): void {

}

erase(num: number): boolean {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = new Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

C# Solution:

```
/*
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Skiplist {

public Skiplist() {

}
public bool Search(int target) {

}

public void Add(int num) {

}

public bool Erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = new Skiplist();
* bool param_1 = obj.Search(target);
* obj.Add(num);
* bool param_3 = obj.Erase(num);
*/
```

C Solution:

```
/*
* Problem: Design Skiplist
* Difficulty: Hard
* Tags: tree, sort, search, linked_list
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} Skiplist;
```

```
Skiplist* skiplistCreate() {

}

bool skiplistSearch(Skiplist* obj, int target) {

}

void skiplistAdd(Skiplist* obj, int num) {

}

bool skiplistErase(Skiplist* obj, int num) {

}

void skiplistFree(Skiplist* obj) {

}

/**
* Your Skiplist struct will be instantiated and called as such:
* Skiplist* obj = skiplistCreate();
* bool param_1 = skiplistSearch(obj, target);

* skiplistAdd(obj, num);

* bool param_3 = skiplistErase(obj, num);

* skiplistFree(obj);
*/
```

Go Solution:

```
// Problem: Design Skiplist
// Difficulty: Hard
// Tags: tree, sort, search, linked_list
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height
```

```
type Skiplist struct {

}


func Constructor() Skiplist {

}


func (this *Skiplist) Search(target int) bool {

}


func (this *Skiplist) Add(num int) {

}


func (this *Skiplist) Erase(num int) bool {

}


/**
* Your Skiplist object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Search(target);
* obj.Add(num);
* param_3 := obj.Erase(num);
*/
```

Kotlin Solution:

```
class Skiplist() {

fun search(target: Int): Boolean {


}
fun add(num: Int) {

}

fun erase(num: Int): Boolean {

}

}

/**
* Your Skiplist object will be instantiated and called as such:
* var obj = Skiplist()
* var param_1 = obj.search(target)
* obj.add(num)
* var param_3 = obj.erase(num)
*/
```

Swift Solution:

```
class Skiplist {

init() {

}

func search(_ target: Int) -> Bool {

}

func add(_ num: Int) {

}

func erase(_ num: Int) -> Bool {

}
}
```

```
/**
* Your Skiplist object will be instantiated and called as such:
* let obj = Skiplist()
* let ret_1: Bool = obj.search(target)
* obj.add(num)
* let ret_3: Bool = obj.erase(num)
*/
```

Rust Solution:

```
// Problem: Design Skiplist
// Difficulty: Hard
// Tags: tree, sort, search, linked_list
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

struct Skiplist {

}



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

fn new() -> Self {

}

fn search(&self, target: i32) -> bool {

}

fn add(&self, num: i32) {

}
fn erase(&self, num: i32) -> bool {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* let obj = Skiplist::new();
* let ret_1: bool = obj.search(target);
* obj.add(num);
* let ret_3: bool = obj.erase(num);
*/
```

Ruby Solution:

```
class Skiplist
def initialize()

end



=begin
:type target: Integer
:rtype: Boolean
=end
def search(target)

end



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

end



=begin
:type num: Integer
```

```
:rtype: Boolean
=end
def erase(num)

end


end

# Your Skiplist object will be instantiated and called as such:
# obj = Skiplist.new()
# param_1 = obj.search(target)
# obj.add(num)
# param_3 = obj.erase(num)
```

PHP Solution:

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

}

/**
* @param Integer $target
* @return Boolean
*/
function search($target) {

}

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

}

/**
* @param Integer $num
* @return Boolean
*/
function erase($num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* $obj = Skiplist();
* $ret_1 = $obj->search($target);
* $obj->add($num);
* $ret_3 = $obj->erase($num);
*/
```

Dart Solution:

```
class Skiplist {

Skiplist() {

}

bool search(int target) {

}

void add(int num) {

}

bool erase(int num) {

}
}

/**
* Your Skiplist object will be instantiated and called as such:
* Skiplist obj = Skiplist();
* bool param1 = obj.search(target);
* obj.add(num);
* bool param3 = obj.erase(num);
*/
```

Scala Solution:

```
class Skiplist() {

def search(target: Int): Boolean = {

}

def add(num: Int): Unit = {

}

def erase(num: Int): Boolean = {

}

}

/**
* Your Skiplist object will be instantiated and called as such:
* val obj = new Skiplist()
* val param_1 = obj.search(target)
* obj.add(num)
* val param_3 = obj.erase(num)
*/
```

Elixir Solution:

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

end

@spec search(target :: integer) :: boolean
def search(target) do
```

```
end

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

end

@spec erase(num :: integer) :: boolean
def erase(num) do

end
end

# Your functions will be called as such:
# Skiplist.init_()
# param_1 = Skiplist.search(target)
# Skiplist.add(num)
# param_3 = Skiplist.erase(num)

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

Erlang Solution:

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

-spec skiplist_search(Target :: integer()) -> boolean().
skiplist_search(Target) ->
.

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

-spec skiplist_erase(Num :: integer()) -> boolean().
skiplist_erase(Num) ->
.
```

```
%% Your functions will be called as such:
%% skiplist_init_(),
%% Param_1 = skiplist_search(Target),
%% skiplist_add(Num),
%% Param_3 = skiplist_erase(Num),

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

Racket Solution:

```
(define skiplist%
(class object%
(super-new)

(init-field)

; search : exact-integer? -> boolean?
(define/public (search target)
)
; add : exact-integer? -> void?
(define/public (add num)
)
; erase : exact-integer? -> boolean?
(define/public (erase num)
)))

;; Your skiplist% object will be instantiated and called as such:
;; (define obj (new skiplist%))
;; (define param_1 (send obj search target))
;; (send obj add num)
;; (define param_3 (send obj erase num))
```