Problem 895: Maximum Frequency Stack

Problem Information

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

Problem Description

Design a stack-like data structure to push elements to the stack and pop the most frequent element from the stack.

Implement the

FreqStack

class:

FreqStack()

constructs an empty frequency stack.

void push(int val)

pushes an integer

val

onto the top of the stack.

int pop()

removes and returns the most frequent element in the stack.

If there is a tie for the most frequent element, the element closest to the stack's top is removed and returned.

Example 1:

Input

["FreqStack", "push", "push", "push", "push", "push", "push", "pop", "pop", "pop", "pop"] [[], [5], [7], [5], [7], [4], [5], [], [], [], []]

Output

[null, null, null, null, null, null, null, 5, 7, 5, 4]

Explanation

FreqStack freqStack = new FreqStack(); freqStack.push(5); // The stack is [5] freqStack.push(7); // The stack is [5,7] freqStack.push(5); // The stack is [5,7,5] freqStack.push(7); // The stack is [5,7,5,7] freqStack.push(4); // The stack is [5,7,5,7,4] freqStack.push(5); // The stack is [5,7,5,7,4,5] freqStack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,5,7,4]. freqStack.pop(); // return 7, as 5 and 7 is the most frequent, but 7 is closest to the top. The stack becomes [5,7,5,4]. freqStack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,4]. freqStack.pop(); // return 4, as 4, 5 and 7 is the most frequent, but 4 is closest to the top. The stack becomes [5,7].

Constraints:

0 <= val <= 10

9

At most

2 * 10

4

calls will be made to

push

and

pop

.

It is guaranteed that there will be at least one element in the stack before calling

pop

.

Code Snippets

C++:

```
class FreqStack {
public:
FreqStack() {

}

void push(int val) {

}

int pop() {

}
};

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack* obj = new FreqStack();
* obj->push(val);
* int param_2 = obj->pop();
*/
```

Java:

```
class FreqStack {

public FreqStack() {

}

public void push(int val) {

}

public int pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = new FreqStack();
* obj.push(val);
* int param_2 = obj.pop();
*/
```

Python3:

```
class FreqStack:

def __init__(self):



def push(self, val: int) -> None:



def pop(self) -> int:




# Your FreqStack object will be instantiated and called as such:
# obj = FreqStack()
# obj.push(val)
# param_2 = obj.pop()
```

Python:

```
class FreqStack(object):

def __init__(self):



def push(self, val):
"""
:type val: int
:rtype: None
"""



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




# Your FreqStack object will be instantiated and called as such:
# obj = FreqStack()
# obj.push(val)
# param_2 = obj.pop()
```

JavaScript:

```

var FreqStack = function() {

};

/**
* @param {number} val
* @return {void}
*/
FreqStack.prototype.push = function(val) {

};

/**
* @return {number}
*/
FreqStack.prototype.pop = function() {
};

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = new FreqStack()
* obj.push(val)
* var param_2 = obj.pop()
*/
```

TypeScript:

```
class FreqStack {
constructor() {

}

push(val: number): void {

}

pop(): number {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = new FreqStack()
* obj.push(val)
* var param_2 = obj.pop()
*/
```

C#:

```
public class FreqStack {

public FreqStack() {

}

public void Push(int val) {
}

public int Pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = new FreqStack();
* obj.Push(val);
* int param_2 = obj.Pop();
*/
```

C:

```
typedef struct {

} FreqStack;


FreqStack* freqStackCreate() {

}

void freqStackPush(FreqStack* obj, int val) {

}

int freqStackPop(FreqStack* obj) {

}

void freqStackFree(FreqStack* obj) {

}

/**
```

```
* Your FreqStack struct will be instantiated and called as such:
* FreqStack* obj = freqStackCreate();
* freqStackPush(obj, val);

* int param_2 = freqStackPop(obj);

* freqStackFree(obj);
*/
```

Go:

```
type FreqStack struct {

}



func Constructor() FreqStack {

}



func (this *FreqStack) Push(val int) {

}



func (this *FreqStack) Pop() int {

}



/**
* Your FreqStack object will be instantiated and called as such:
* obj := Constructor();
* obj.Push(val);
* param_2 := obj.Pop();
*/
```

Kotlin:

```
class FreqStack() {
```

```
fun push(`val`: Int) {

}

fun pop(): Int {

}

}

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = FreqStack()
* obj.push(`val`)
* var param_2 = obj.pop()
*/
```

Swift:

```
class FreqStack {

init() {

}

func push(_ val: Int) {

}

func pop() -> Int {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* let obj = FreqStack()
* obj.push(val)
* let ret_2: Int = obj.pop()
*/
```

Rust:

```
struct FreqStack {

}



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

fn new() -> Self {

}

fn push(&self, val: i32) {

}

fn pop(&self) -> i32 {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* let obj = FreqStack::new();
* obj.push(val);
* let ret_2: i32 = obj.pop();
*/
```

Ruby:

```
class FreqStack
def initialize()

end



=begin
```

```
:type val: Integer
:rtype: Void
=end
def push(val)

end


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

end


end

# Your FreqStack object will be instantiated and called as such:
# obj = FreqStack.new()
# obj.push(val)
# param_2 = obj.pop()
```

PHP:

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

}

/**
* @param Integer $val
* @return NULL
*/
function push($val) {

}

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

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* $obj = FreqStack();
* $obj->push($val);
* $ret_2 = $obj->pop();
*/
```

Dart:

```
class FreqStack {

FreqStack() {

}

void push(int val) {

}

int pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = FreqStack();
* obj.push(val);
* int param2 = obj.pop();
*/
```

Scala:

```
class FreqStack() {

def push(`val`: Int): Unit = {
}

def pop(): Int = {

}

}

/**
* Your FreqStack object will be instantiated and called as such:
* val obj = new FreqStack()
* obj.push(`val`)
* val param_2 = obj.pop()
*/
```

Elixir:

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

end

@spec push(val :: integer) :: any
def push(val) do

end

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

end
end

# Your functions will be called as such:
# FreqStack.init_()
# FreqStack.push(val)
# param_2 = FreqStack.pop()

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

Erlang:

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

-spec freq_stack_push(Val :: integer()) -> any().
freq_stack_push(Val) ->
.

-spec freq_stack_pop() -> integer().
freq_stack_pop() ->
.


%% Your functions will be called as such:
%% freq_stack_init_(),
%% freq_stack_push(Val),
%% Param_2 = freq_stack_pop(),

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

Racket:

```
(define freq-stack%
(class object%
(super-new)

(init-field)

; push : exact-integer? -> void?
(define/public (push val)
)
; pop : -> exact-integer?
(define/public (pop)
)))

;; Your freq-stack% object will be instantiated and called as such:
;; (define obj (new freq-stack%))
;; (send obj push val)
;; (define param_2 (send obj pop))
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
*
* 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 FreqStack {
public:
FreqStack() {

}

void push(int val) {

}

int pop() {

}
};

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack* obj = new FreqStack();
* obj->push(val);
* int param_2 = obj->pop();
*/
```

Java Solution:

```
/**
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
*
* 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 FreqStack {

public FreqStack() {

}

public void push(int val) {

}

public int pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = new FreqStack();
* obj.push(val);
* int param_2 = obj.pop();
*/
```

Python3 Solution:

```
"""
Problem: Maximum Frequency Stack
Difficulty: Hard
Tags: hash, stack

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 FreqStack:

def __init__(self):



def push(self, val: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class FreqStack(object):

def __init__(self):



def push(self, val):
"""
:type val: int
:rtype: None
"""



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




# Your FreqStack object will be instantiated and called as such:
# obj = FreqStack()
# obj.push(val)
# param_2 = obj.pop()
```

JavaScript Solution:

```
/**
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
```

```
*
* 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 FreqStack = function() {

};

/**
* @param {number} val
* @return {void}
*/
FreqStack.prototype.push = function(val) {

};

/**
* @return {number}
*/
FreqStack.prototype.pop = function() {

};

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = new FreqStack()
* obj.push(val)
* var param_2 = obj.pop()
*/
```

TypeScript Solution:

```
/**
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
*
* 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 FreqStack {
constructor() {

}

push(val: number): void {

}

pop(): number {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = new FreqStack()
* obj.push(val)
* var param_2 = obj.pop()
*/
```

C# Solution:

```
/*
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
*
* 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 FreqStack {

public FreqStack() {
}

public void Push(int val) {

}

public int Pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = new FreqStack();
* obj.Push(val);
* int param_2 = obj.Pop();
*/
```

C Solution:

```
/*
* Problem: Maximum Frequency Stack
* Difficulty: Hard
* Tags: hash, stack
*
* 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 {

} FreqStack;


FreqStack* freqStackCreate() {

}
```

```
void freqStackPush(FreqStack* obj, int val) {

}

int freqStackPop(FreqStack* obj) {

}

void freqStackFree(FreqStack* obj) {

}

/**
* Your FreqStack struct will be instantiated and called as such:
* FreqStack* obj = freqStackCreate();
* freqStackPush(obj, val);

* int param_2 = freqStackPop(obj);

* freqStackFree(obj);
*/
```

Go Solution:

```
// Problem: Maximum Frequency Stack
// Difficulty: Hard
// Tags: hash, stack
//
// 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 FreqStack struct {

}



func Constructor() FreqStack {

}
```

```
func (this *FreqStack) Push(val int) {

}


func (this *FreqStack) Pop() int {

}


/**
* Your FreqStack object will be instantiated and called as such:
* obj := Constructor();
* obj.Push(val);
* param_2 := obj.Pop();
*/
```

Kotlin Solution:

```
class FreqStack() {

fun push(`val`: Int) {

}

fun pop(): Int {

}

}

/**
* Your FreqStack object will be instantiated and called as such:
* var obj = FreqStack()
* obj.push(`val`)
* var param_2 = obj.pop()
*/
```

Swift Solution:

```

class FreqStack {

init() {

}

func push(_ val: Int) {

}

func pop() -> Int {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* let obj = FreqStack()
* obj.push(val)
* let ret_2: Int = obj.pop()
*/
```

Rust Solution:

```
// Problem: Maximum Frequency Stack
// Difficulty: Hard
// Tags: hash, stack
//
// 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 FreqStack {

}



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

```
impl FreqStack {

fn new() -> Self {

}

fn push(&self, val: i32) {

}

fn pop(&self) -> i32 {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* let obj = FreqStack::new();
* obj.push(val);
* let ret_2: i32 = obj.pop();
*/
```

Ruby Solution:

```
class FreqStack
def initialize()

end



=begin
:type val: Integer
:rtype: Void
=end
def push(val)

end



=begin
:rtype: Integer
```

```
=end
def pop()

end



end

# Your FreqStack object will be instantiated and called as such:
# obj = FreqStack.new()
# obj.push(val)
# param_2 = obj.pop()
```

PHP Solution:

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

}

/**
* @param Integer $val
* @return NULL
*/
function push($val) {

}

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

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* $obj = FreqStack();
* $obj->push($val);
* $ret_2 = $obj->pop();
*/
```

Dart Solution:

```
class FreqStack {

FreqStack() {

}

void push(int val) {

}

int pop() {

}
}

/**
* Your FreqStack object will be instantiated and called as such:
* FreqStack obj = FreqStack();
* obj.push(val);
* int param2 = obj.pop();
*/
```

Scala Solution:

```
class FreqStack() {

def push(`val`: Int): Unit = {

}

def pop(): Int = {

}

}
```

```
/**
* Your FreqStack object will be instantiated and called as such:
* val obj = new FreqStack()
* obj.push(`val`)
* val param_2 = obj.pop()
*/
```

Elixir Solution:

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

end

@spec push(val :: integer) :: any
def push(val) do

end

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

end
end

# Your functions will be called as such:
# FreqStack.init_()
# FreqStack.push(val)
# param_2 = FreqStack.pop()

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

Erlang Solution:

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

```
-spec freq_stack_push(Val :: integer()) -> any().
freq_stack_push(Val) ->
.

-spec freq_stack_pop() -> integer().
freq_stack_pop() ->
.


%% Your functions will be called as such:
%% freq_stack_init_(),
%% freq_stack_push(Val),
%% Param_2 = freq_stack_pop(),

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

Racket Solution:

```
(define freq-stack%
(class object%
(super-new)

(init-field)

; push : exact-integer? -> void?
(define/public (push val)
)
; pop : -> exact-integer?
(define/public (pop)
)))

;; Your freq-stack% object will be instantiated and called as such:
;; (define obj (new freq-stack%))
;; (send obj push val)
;; (define param_2 (send obj pop))
```