Problem 2166: Design Bitset

Problem Information

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

Problem Description

A

Bitset

is a data structure that compactly stores bits.

Implement the

Bitset

class:

Bitset(int size)

Initializes the Bitset with

size

bits, all of which are

0

.

void fix(int idx)

Updates the value of the bit at the index

idx

to

1

. If the value was already

1

, no change occurs.

void unfix(int idx)

Updates the value of the bit at the index

idx

to

0

. If the value was already

0

, no change occurs.

void flip()

Flips the values of each bit in the Bitset. In other words, all bits with value

0

will now have value

1

and vice versa.

boolean all()

Checks if the value of

each

bit in the Bitset is

1

. Returns

true

if it satisfies the condition,

false

otherwise.

boolean one()

Checks if there is

at least one

bit in the Bitset with value

1

. Returns

true

if it satisfies the condition,

false

otherwise.

int count()

Returns the

total number

of bits in the Bitset which have value

1

.

String toString()

Returns the current composition of the Bitset. Note that in the resultant string, the character at the

i

th

index should coincide with the value at the

i

th

bit of the Bitset.

Example 1:

Input

["Bitset", "fix", "fix", "flip", "all", "unfix", "flip", "one", "unfix", "count", "toString"] [[5], [3], [1], [], [], [0], [], [], [0], [], []]

Output

[null, null, null, null, false, null, null, true, null, 2, "01010"]

Explanation

Bitset bs = new Bitset(5); // bitset = "00000". bs.fix(3); // the value at idx = 3 is updated to 1, so bitset = "00010". bs.fix(1); // the value at idx = 1 is updated to 1, so bitset = "01010". bs.flip(); // the value of each bit is flipped, so bitset = "10101". bs.all(); // return False, as not all values of the bitset are 1. bs.unfix(0); // the value at idx = 0 is updated to 0, so bitset = "00101". bs.flip(); // the value of each bit is flipped, so bitset = "11010". bs.one(); // return True, as there is at least 1 index with value 1. bs.unfix(0); // the value at idx = 0 is updated to 0, so bitset = "01010". bs.count(); // return 2, as there are 2 bits with value 1. bs.toString(); // return "01010", which is the composition of bitset.

Constraints:

1 <= size <= 10

5

0 <= idx <= size - 1

At most

10

5

calls will be made

in total

to

fix

,

unfix

,

flip

,

all

,

one

,

count

, and

toString

.

At least one call will be made to

all

,

one

,

count

, or

toString

.

At most

5

calls will be made to

toString

.

Code Snippets

C++:

```
class Bitset {
public:
Bitset(int size) {

}

void fix(int idx) {

}

void unfix(int idx) {

}

void flip() {

}

bool all() {

}

bool one() {
```

```
}

int count() {

}

string toString() {

}
};

/**
* Your Bitset object will be instantiated and called as such:
* Bitset* obj = new Bitset(size);
* obj->fix(idx);
* obj->unfix(idx);
* obj->flip();
* bool param_4 = obj->all();
* bool param_5 = obj->one();
* int param_6 = obj->count();
* string param_7 = obj->toString();
*/
```

Java:

```
class Bitset {

public Bitset(int size) {

}

public void fix(int idx) {

}

public void unfix(int idx) {

}

public void flip() {

}
```

```
public boolean all() {

}

public boolean one() {

}

public int count() {

}

public String toString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = new Bitset(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* boolean param_4 = obj.all();
* boolean param_5 = obj.one();
* int param_6 = obj.count();
* String param_7 = obj.toString();
*/
```

Python3:

```
class Bitset:

def __init__(self, size: int):


def fix(self, idx: int) -> None:


def unfix(self, idx: int) -> None:
```

```
def flip(self) -> None:


def all(self) -> bool:


def one(self) -> bool:


def count(self) -> int:


def toString(self) -> str:



# Your Bitset object will be instantiated and called as such:
# obj = Bitset(size)
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.toString()
```

Python:

```
class Bitset(object):

def __init__(self, size):
"""
:type size: int
"""



def fix(self, idx):
"""
:type idx: int
:rtype: None
"""
```

```
def unfix(self, idx):
"""
:type idx: int
:rtype: None
"""


def flip(self):
"""
:rtype: None
"""


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


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


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


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



# Your Bitset object will be instantiated and called as such:
# obj = Bitset(size)
```

```
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.toString()
```

JavaScript:

```
/**
* @param {number} size
*/
var Bitset = function(size) {

};

/**
* @param {number} idx
* @return {void}
*/
Bitset.prototype.fix = function(idx) {

};

/**
* @param {number} idx
* @return {void}
*/
Bitset.prototype.unfix = function(idx) {

};

/**
* @return {void}
*/
Bitset.prototype.flip = function() {

};

/**
* @return {boolean}
```

```
*/
Bitset.prototype.all = function() {

};

/**
* @return {boolean}
*/
Bitset.prototype.one = function() {

};

/**
* @return {number}
*/
Bitset.prototype.count = function() {

};

/**
* @return {string}
*/
Bitset.prototype.toString = function() {

};

/**
* Your Bitset object will be instantiated and called as such:
* var obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
*/
```

TypeScript:

```
class Bitset {
constructor(size: number) {
```

```
}

fix(idx: number): void {

}

unfix(idx: number): void {

}

flip(): void {

}

all(): boolean {

}

one(): boolean {

}

count(): number {

}

toString(): string {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* var obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
```

```
*/
```

C#:

```
public class Bitset {

public Bitset(int size) {

}

public void Fix(int idx) {

}

public void Unfix(int idx) {

}

public void Flip() {

}

public bool All() {

}

public bool One() {

}

public int Count() {

}

public string ToString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = new Bitset(size);
```

```
* obj.Fix(idx);
* obj.Unfix(idx);
* obj.Flip();
* bool param_4 = obj.All();
* bool param_5 = obj.One();
* int param_6 = obj.Count();
* string param_7 = obj.ToString();
*/
```

C:

```
typedef struct {

} Bitset;


Bitset* bitsetCreate(int size) {

}

void bitsetFix(Bitset* obj, int idx) {

}

void bitsetUnfix(Bitset* obj, int idx) {

}

void bitsetFlip(Bitset* obj) {

}

bool bitsetAll(Bitset* obj) {

}

bool bitsetOne(Bitset* obj) {

}
```

```
int bitsetCount(Bitset* obj) {

}

char* bitsetToString(Bitset* obj) {

}

void bitsetFree(Bitset* obj) {

}

/**
* Your Bitset struct will be instantiated and called as such:
* Bitset* obj = bitsetCreate(size);
* bitsetFix(obj, idx);

* bitsetUnfix(obj, idx);

* bitsetFlip(obj);

* bool param_4 = bitsetAll(obj);

* bool param_5 = bitsetOne(obj);

* int param_6 = bitsetCount(obj);

* char* param_7 = bitsetToString(obj);

* bitsetFree(obj);
*/
```

Go:

```
type Bitset struct {

}


func Constructor(size int) Bitset {
```

```
}


func (this *Bitset) Fix(idx int) {

}


func (this *Bitset) Unfix(idx int) {

}


func (this *Bitset) Flip() {

}


func (this *Bitset) All() bool {

}


func (this *Bitset) One() bool {

}


func (this *Bitset) Count() int {

}


func (this *Bitset) ToString() string {

}


/**
* Your Bitset object will be instantiated and called as such:
* obj := Constructor(size);
* obj.Fix(idx);
```

```
* obj.Unfix(idx);
* obj.Flip();
* param_4 := obj.All();
* param_5 := obj.One();
* param_6 := obj.Count();
* param_7 := obj.ToString();
*/
```

Kotlin:

```
class Bitset(size: Int) {

fun fix(idx: Int) {

}

fun unfix(idx: Int) {

}

fun flip() {

}

fun all(): Boolean {

}

fun one(): Boolean {

}

fun count(): Int {

}

fun toString(): String {

}

}
```

```
/**
* Your Bitset object will be instantiated and called as such:
* var obj = Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
*/
```

Swift:

```
class Bitset {

init(_ size: Int) {

}

func fix(_ idx: Int) {

}

func unfix(_ idx: Int) {

}

func flip() {

}

func all() -> Bool {

}

func one() -> Bool {

}

func count() -> Int {
```

```
}

func toString() -> String {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* let obj = Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* let ret_4: Bool = obj.all()
* let ret_5: Bool = obj.one()
* let ret_6: Int = obj.count()
* let ret_7: String = obj.toString()
*/
```

Rust:

```
struct Bitset {

}


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

fn new(size: i32) -> Self {

}

fn fix(&self, idx: i32) {

}

fn unfix(&self, idx: i32) {
```

```
}

fn flip(&self) {

}

fn all(&self) -> bool {

}

fn one(&self) -> bool {

}

fn count(&self) -> i32 {

}

fn to_string(&self) -> String {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* let obj = Bitset::new(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* let ret_4: bool = obj.all();
* let ret_5: bool = obj.one();
* let ret_6: i32 = obj.count();
* let ret_7: String = obj.to_string();
*/
```

Ruby:

```
class Bitset

=begin
:type size: Integer
```

```
=end
def initialize(size)

end


=begin
:type idx: Integer
:rtype: Void
=end
def fix(idx)

end


=begin
:type idx: Integer
:rtype: Void
=end
def unfix(idx)

end


=begin
:rtype: Void
=end
def flip()

end


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

end


=begin
:rtype: Boolean
```

```
=end
def one()

end


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

end


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

end


end

# Your Bitset object will be instantiated and called as such:
# obj = Bitset.new(size)
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.to_string()
```

PHP:

```
class Bitset {
/**
* @param Integer $size
*/
function __construct($size) {
```

```
}

/**
* @param Integer $idx
* @return NULL
*/
function fix($idx) {

}

/**
* @param Integer $idx
* @return NULL
*/
function unfix($idx) {

}

/**
* @return NULL
*/
function flip() {

}

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

}

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

}

/**
* @return Integer
*/
```

```
function count() {

}

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

}
}

/**
* Your Bitset object will be instantiated and called as such:
* $obj = Bitset($size);
* $obj->fix($idx);
* $obj->unfix($idx);
* $obj->flip();
* $ret_4 = $obj->all();
* $ret_5 = $obj->one();
* $ret_6 = $obj->count();
* $ret_7 = $obj->toString();
*/
```

Dart:

```
class Bitset {

Bitset(int size) {

}

void fix(int idx) {

}

void unfix(int idx) {

}

void flip() {
```

```
}

bool all() {

}

bool one() {

}

int count() {

}

String toString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = Bitset(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* bool param4 = obj.all();
* bool param5 = obj.one();
* int param6 = obj.count();
* String param7 = obj.toString();
*/
```

Scala:

```
class Bitset(_size: Int) {

def fix(idx: Int): Unit = {

}

def unfix(idx: Int): Unit = {

}
```

```
def flip(): Unit = {

}

def all(): Boolean = {

}

def one(): Boolean = {

}

def count(): Int = {

}

def toString(): String = {

}

}

/**
* Your Bitset object will be instantiated and called as such:
* val obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* val param_4 = obj.all()
* val param_5 = obj.one()
* val param_6 = obj.count()
* val param_7 = obj.toString()
*/
```

Elixir:

```
defmodule Bitset do
@spec init_(size :: integer) :: any
def init_(size) do

end
```

```
@spec fix(idx :: integer) :: any
def fix(idx) do

end

@spec unfix(idx :: integer) :: any
def unfix(idx) do

end

@spec flip() :: any
def flip() do

end

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

end

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

end

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

end

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

end
end

# Your functions will be called as such:
# Bitset.init_(size)
# Bitset.fix(idx)
# Bitset.unfix(idx)
# Bitset.flip()
```

```
# param_4 = Bitset.all()
# param_5 = Bitset.one()
# param_6 = Bitset.count()
# param_7 = Bitset.to_string()

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

Erlang:

```
-spec bitset_init_(Size :: integer()) -> any().
bitset_init_(Size) ->
.

-spec bitset_fix(Idx :: integer()) -> any().
bitset_fix(Idx) ->
.

-spec bitset_unfix(Idx :: integer()) -> any().
bitset_unfix(Idx) ->
.

-spec bitset_flip() -> any().
bitset_flip() ->
.

-spec bitset_all() -> boolean().
bitset_all() ->
.

-spec bitset_one() -> boolean().
bitset_one() ->
.

-spec bitset_count() -> integer().
bitset_count() ->
.

-spec bitset_to_string() -> unicode:unicode_binary().
bitset_to_string() ->
.
```

```
%% Your functions will be called as such:
%% bitset_init_(Size),
%% bitset_fix(Idx),
%% bitset_unfix(Idx),
%% bitset_flip(),
%% Param_4 = bitset_all(),
%% Param_5 = bitset_one(),
%% Param_6 = bitset_count(),
%% Param_7 = bitset_to_string(),

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

Racket:

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

; size : exact-integer?
(init-field
size)

; fix : exact-integer? -> void?
(define/public (fix idx)
)
; unfix : exact-integer? -> void?
(define/public (unfix idx)
)
; flip : -> void?
(define/public (flip)
)
; all : -> boolean?
(define/public (all)
)
; one : -> boolean?
(define/public (one)
)
; count : -> exact-integer?
(define/public (count)
)
```

```
; to-string : -> string?
(define/public (to-string)
)))

;; Your bitset% object will be instantiated and called as such:
;; (define obj (new bitset% [size size]))
;; (send obj fix idx)
;; (send obj unfix idx)
;; (send obj flip)
;; (define param_4 (send obj all))
;; (define param_5 (send obj one))
;; (define param_6 (send obj count))
;; (define param_7 (send obj to-string))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Bitset {
public:
Bitset(int size) {

}

void fix(int idx) {

}

void unfix(int idx) {
```

```
}

void flip() {

}

bool all() {

}

bool one() {

}

int count() {

}

string toString() {

}
};

/**
* Your Bitset object will be instantiated and called as such:
* Bitset* obj = new Bitset(size);
* obj->fix(idx);
* obj->unfix(idx);
* obj->flip();
* bool param_4 = obj->all();
* bool param_5 = obj->one();
* int param_6 = obj->count();
* string param_7 = obj->toString();
*/
```

Java Solution:

```
/**
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Bitset {

public Bitset(int size) {

}

public void fix(int idx) {

}

public void unfix(int idx) {

}

public void flip() {

}

public boolean all() {

}

public boolean one() {

}

public int count() {

}

public String toString() {

}
}

/**
```

```
* Your Bitset object will be instantiated and called as such:
* Bitset obj = new Bitset(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* boolean param_4 = obj.all();
* boolean param_5 = obj.one();
* int param_6 = obj.count();
* String param_7 = obj.toString();
*/
```

Python3 Solution:

```
"""
Problem: Design Bitset
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Bitset:

def __init__(self, size: int):


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

Python Solution:

```
class Bitset(object):

def __init__(self, size):
"""
:type size: int
"""
```

```
def fix(self, idx):
"""
:type idx: int
:rtype: None
"""


def unfix(self, idx):
"""
:type idx: int
:rtype: None
"""


def flip(self):
"""
:rtype: None
"""


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


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


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


def toString(self):
"""
:rtype: str
```

```
"""



# Your Bitset object will be instantiated and called as such:
# obj = Bitset(size)
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.toString()
```

JavaScript Solution:

```
/**
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} size
*/
var Bitset = function(size) {

};

/**
* @param {number} idx
* @return {void}
*/
Bitset.prototype.fix = function(idx) {

};
```

```
/**
* @param {number} idx
* @return {void}
*/
Bitset.prototype.unfix = function(idx) {

};

/**
* @return {void}
*/
Bitset.prototype.flip = function() {

};

/**
* @return {boolean}
*/
Bitset.prototype.all = function() {

};

/**
* @return {boolean}
*/
Bitset.prototype.one = function() {

};

/**
* @return {number}
*/
Bitset.prototype.count = function() {

};

/**
* @return {string}
*/
Bitset.prototype.toString = function() {

};
```

```
/**
* Your Bitset object will be instantiated and called as such:
* var obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
*/
```

TypeScript Solution:

```
/**
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Bitset {
constructor(size: number) {

}

fix(idx: number): void {

}

unfix(idx: number): void {

}

flip(): void {

}
```

```
all(): boolean {

}

one(): boolean {

}

count(): number {

}

toString(): string {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* var obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
*/
```

C# Solution:

```
/*
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
public class Bitset {

public Bitset(int size) {

}

public void Fix(int idx) {

}

public void Unfix(int idx) {

}

public void Flip() {

}

public bool All() {

}

public bool One() {

}

public int Count() {

}

public string ToString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = new Bitset(size);
* obj.Fix(idx);
* obj.Unfix(idx);
* obj.Flip();
```

```
* bool param_4 = obj.All();
* bool param_5 = obj.One();
* int param_6 = obj.Count();
* string param_7 = obj.ToString();
*/
```

C Solution:

```
/*
* Problem: Design Bitset
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} Bitset;


Bitset* bitsetCreate(int size) {

}

void bitsetFix(Bitset* obj, int idx) {

}

void bitsetUnfix(Bitset* obj, int idx) {

}

void bitsetFlip(Bitset* obj) {

}
```

```
bool bitsetAll(Bitset* obj) {

}

bool bitsetOne(Bitset* obj) {

}

int bitsetCount(Bitset* obj) {

}

char* bitsetToString(Bitset* obj) {

}

void bitsetFree(Bitset* obj) {

}

/**
* Your Bitset struct will be instantiated and called as such:
* Bitset* obj = bitsetCreate(size);
* bitsetFix(obj, idx);

* bitsetUnfix(obj, idx);

* bitsetFlip(obj);

* bool param_4 = bitsetAll(obj);

* bool param_5 = bitsetOne(obj);

* int param_6 = bitsetCount(obj);

* char* param_7 = bitsetToString(obj);

* bitsetFree(obj);
*/
```

Go Solution:

```
// Problem: Design Bitset
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type Bitset struct {

}


func Constructor(size int) Bitset {

}


func (this *Bitset) Fix(idx int) {

}


func (this *Bitset) Unfix(idx int) {

}


func (this *Bitset) Flip() {

}


func (this *Bitset) All() bool {

}


func (this *Bitset) One() bool {

}
```

```
func (this *Bitset) Count() int {

}


func (this *Bitset) ToString() string {

}


/**
* Your Bitset object will be instantiated and called as such:
* obj := Constructor(size);
* obj.Fix(idx);
* obj.Unfix(idx);
* obj.Flip();
* param_4 := obj.All();
* param_5 := obj.One();
* param_6 := obj.Count();
* param_7 := obj.ToString();
*/
```

Kotlin Solution:

```
class Bitset(size: Int) {

fun fix(idx: Int) {

}

fun unfix(idx: Int) {

}

fun flip() {

}

fun all(): Boolean {
```

```
}

fun one(): Boolean {

}

fun count(): Int {

}

fun toString(): String {

}

}

/**
* Your Bitset object will be instantiated and called as such:
* var obj = Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* var param_4 = obj.all()
* var param_5 = obj.one()
* var param_6 = obj.count()
* var param_7 = obj.toString()
*/
```

Swift Solution:

```
class Bitset {

init(_ size: Int) {

}

func fix(_ idx: Int) {

}
```

```
func unfix(_ idx: Int) {

}

func flip() {

}

func all() -> Bool {

}

func one() -> Bool {

}

func count() -> Int {

}

func toString() -> String {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* let obj = Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* let ret_4: Bool = obj.all()
* let ret_5: Bool = obj.one()
* let ret_6: Int = obj.count()
* let ret_7: String = obj.toString()
*/
```

Rust Solution:

```
// Problem: Design Bitset
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct Bitset {

}



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

fn new(size: i32) -> Self {

}

fn fix(&self, idx: i32) {

}

fn unfix(&self, idx: i32) {

}

fn flip(&self) {

}

fn all(&self) -> bool {

}

fn one(&self) -> bool {

}
```

```
fn count(&self) -> i32 {

}

fn to_string(&self) -> String {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* let obj = Bitset::new(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* let ret_4: bool = obj.all();
* let ret_5: bool = obj.one();
* let ret_6: i32 = obj.count();
* let ret_7: String = obj.to_string();
*/
```

Ruby Solution:

```
class Bitset

=begin
:type size: Integer
=end
def initialize(size)

end


=begin
:type idx: Integer
:rtype: Void
=end
def fix(idx)

end
```

```
=begin
:type idx: Integer
:rtype: Void
=end
def unfix(idx)

end


=begin
:rtype: Void
=end
def flip()

end


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

end


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

end


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

end
```

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

end



end

# Your Bitset object will be instantiated and called as such:
# obj = Bitset.new(size)
# obj.fix(idx)
# obj.unfix(idx)
# obj.flip()
# param_4 = obj.all()
# param_5 = obj.one()
# param_6 = obj.count()
# param_7 = obj.to_string()
```

PHP Solution:

```
class Bitset {
/**
* @param Integer $size
*/
function __construct($size) {

}

/**
* @param Integer $idx
* @return NULL
*/
function fix($idx) {

}

/**
* @param Integer $idx
* @return NULL
```

```
*/
function unfix($idx) {

}

/**
* @return NULL
*/
function flip() {

}

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

}

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

}

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

}

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

}
}

/**
```

```
* Your Bitset object will be instantiated and called as such:
* $obj = Bitset($size);
* $obj->fix($idx);
* $obj->unfix($idx);
* $obj->flip();
* $ret_4 = $obj->all();
* $ret_5 = $obj->one();
* $ret_6 = $obj->count();
* $ret_7 = $obj->toString();
*/
```

Dart Solution:

```
class Bitset {

Bitset(int size) {

}

void fix(int idx) {

}

void unfix(int idx) {

}

void flip() {

}

bool all() {

}

bool one() {

}

int count() {
```

```
}

String toString() {

}
}

/**
* Your Bitset object will be instantiated and called as such:
* Bitset obj = Bitset(size);
* obj.fix(idx);
* obj.unfix(idx);
* obj.flip();
* bool param4 = obj.all();
* bool param5 = obj.one();
* int param6 = obj.count();
* String param7 = obj.toString();
*/
```

Scala Solution:

```
class Bitset(_size: Int) {

def fix(idx: Int): Unit = {

}

def unfix(idx: Int): Unit = {

}

def flip(): Unit = {

}

def all(): Boolean = {

}

def one(): Boolean = {
```

```
}

def count(): Int = {

}

def toString(): String = {

}

}

/**
* Your Bitset object will be instantiated and called as such:
* val obj = new Bitset(size)
* obj.fix(idx)
* obj.unfix(idx)
* obj.flip()
* val param_4 = obj.all()
* val param_5 = obj.one()
* val param_6 = obj.count()
* val param_7 = obj.toString()
*/
```

Elixir Solution:

```
defmodule Bitset do
@spec init_(size :: integer) :: any
def init_(size) do

end

@spec fix(idx :: integer) :: any
def fix(idx) do

end

@spec unfix(idx :: integer) :: any
def unfix(idx) do

end
```

```
@spec flip() :: any
def flip() do

end

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

end

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

end

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

end

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

end
end

# Your functions will be called as such:
# Bitset.init_(size)
# Bitset.fix(idx)
# Bitset.unfix(idx)
# Bitset.flip()
# param_4 = Bitset.all()
# param_5 = Bitset.one()
# param_6 = Bitset.count()
# param_7 = Bitset.to_string()

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

Erlang Solution:

```
-spec bitset_init_(Size :: integer()) -> any().
bitset_init_(Size) ->
.

-spec bitset_fix(Idx :: integer()) -> any().
bitset_fix(Idx) ->
.

-spec bitset_unfix(Idx :: integer()) -> any().
bitset_unfix(Idx) ->
.

-spec bitset_flip() -> any().
bitset_flip() ->
.

-spec bitset_all() -> boolean().
bitset_all() ->
.

-spec bitset_one() -> boolean().
bitset_one() ->
.

-spec bitset_count() -> integer().
bitset_count() ->
.

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


%% Your functions will be called as such:
%% bitset_init_(Size),
%% bitset_fix(Idx),
%% bitset_unfix(Idx),
%% bitset_flip(),
%% Param_4 = bitset_all(),
%% Param_5 = bitset_one(),
%% Param_6 = bitset_count(),
%% Param_7 = bitset_to_string(),
```

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

Racket Solution:

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

; size : exact-integer?
(init-field
size)

; fix : exact-integer? -> void?
(define/public (fix idx)
)
; unfix : exact-integer? -> void?
(define/public (unfix idx)
)
; flip : -> void?
(define/public (flip)
)
; all : -> boolean?
(define/public (all)
)
; one : -> boolean?
(define/public (one)
)
; count : -> exact-integer?
(define/public (count)
)
; to-string : -> string?
(define/public (to-string)
)))

;; Your bitset% object will be instantiated and called as such:
;; (define obj (new bitset% [size size]))
;; (send obj fix idx)
;; (send obj unfix idx)
;; (send obj flip)
;; (define param_4 (send obj all))
```

```
;; (define param_5 (send obj one))
;; (define param_6 (send obj count))
;; (define param_7 (send obj to-string))
```