Problem 1865: Finding Pairs With a Certain Sum

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

. You are tasked to implement a data structure that supports queries of two types:

Add

a positive integer to an element of a given index in the array

nums2

.

Count

the number of pairs

(i, j)

such that

nums1[i] + nums2[j]

equals a given value (

0 <= i < nums1.length

and

0 <= j < nums2.length

).

Implement the

FindSumPairs

class:

FindSumPairs(int[] nums1, int[] nums2)

Initializes the

FindSumPairs

object with two integer arrays

nums1

and

nums2

.

void add(int index, int val)

Adds

val

to

nums2[index]

, i.e., apply

nums2[index] += val

.

int count(int tot)

Returns the number of pairs

(i, j)

such that

nums1[i] + nums2[j] == tot

.

Example 1:

Input

["FindSumPairs", "count", "add", "count", "count", "add", "add", "count"] [[[1, 1, 2, 2, 2, 3], [1, 4, 5, 2, 5, 4]], [7], [3, 2], [8], [4], [0, 1], [1, 1], [7]]

Output

[null, 8, null, 2, 1, null, null, 11]

Explanation

FindSumPairs findSumPairs = new FindSumPairs([1, 1, 2, 2, 2, 3], [1, 4, 5, 2, 5, 4]); findSumPairs.count(7); // return 8; pairs (2,2), (3,2), (4,2), (2,4), (3,4), (4,4) make 2 + 5 and pairs (5,1), (5,5) make 3 + 4 findSumPairs.add(3, 2); // now nums2 = [1,4,5,

4

,5,4

] findSumPairs.count(8); // return 2; pairs (5,2), (5,4) make 3 + 5 findSumPairs.count(4); // return 1; pair (5,0) makes 3 + 1 findSumPairs.add(0, 1); // now nums2 = [

2

,4,5,4

,5,4

] findSumPairs.add(1, 1); // now nums2 = [

2

,

5

,5,4

,5,4

] findSumPairs.count(7); // return 11; pairs (2,1), (2,2), (2,4), (3,1), (3,2), (3,4), (4,1), (4,2), (4,4) make 2 + 5 and pairs (5,3), (5,5) make 3 + 4

Constraints:

1 <= nums1.length <= 1000

1 <= nums2.length <= 10

5

1 <= nums1[i] <= 10

9

1 <= nums2[i] <= 10

5

0 <= index < nums2.length

1 <= val <= 10

5

1 <= tot <= 10

9

At most

1000

calls are made to

add

and

count

each

.

Code Snippets

C++:

```
class FindSumPairs {
public:
FindSumPairs(vector<int>& nums1, vector<int>& nums2) {
```

```
}

void add(int index, int val) {

}

int count(int tot) {

}
};

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs* obj = new FindSumPairs(nums1, nums2);
* obj->add(index,val);
* int param_2 = obj->count(tot);
*/
```

Java:

```
class FindSumPairs {

public FindSumPairs(int[] nums1, int[] nums2) {

}

public void add(int index, int val) {

}

public int count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs obj = new FindSumPairs(nums1, nums2);
* obj.add(index,val);
* int param_2 = obj.count(tot);
*/
```

Python3:

```
class FindSumPairs:

def __init__(self, nums1: List[int], nums2: List[int]):


def add(self, index: int, val: int) -> None:


def count(self, tot: int) -> int:




# Your FindSumPairs object will be instantiated and called as such:
# obj = FindSumPairs(nums1, nums2)
# obj.add(index,val)
# param_2 = obj.count(tot)
```

Python:

```
class FindSumPairs(object):

def __init__(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
"""


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


def count(self, tot):
"""
:type tot: int
:rtype: int
```

```
"""



# Your FindSumPairs object will be instantiated and called as such:
# obj = FindSumPairs(nums1, nums2)
# obj.add(index,val)
# param_2 = obj.count(tot)
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
*/
var FindSumPairs = function(nums1, nums2) {

};

/**
* @param {number} index
* @param {number} val
* @return {void}
*/
FindSumPairs.prototype.add = function(index, val) {

};

/**
* @param {number} tot
* @return {number}
*/
FindSumPairs.prototype.count = function(tot) {

};

/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = new FindSumPairs(nums1, nums2)
* obj.add(index,val)
* var param_2 = obj.count(tot)
*/
```

TypeScript:

```
class FindSumPairs {
constructor(nums1: number[], nums2: number[]) {

}

add(index: number, val: number): void {

}

count(tot: number): number {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = new FindSumPairs(nums1, nums2)
* obj.add(index,val)
* var param_2 = obj.count(tot)
*/
```

C#:

```
public class FindSumPairs {

public FindSumPairs(int[] nums1, int[] nums2) {

}

public void Add(int index, int val) {

}

public int Count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
```

```
* FindSumPairs obj = new FindSumPairs(nums1, nums2);
* obj.Add(index,val);
* int param_2 = obj.Count(tot);
*/
```

C:

```
typedef struct {

} FindSumPairs;


FindSumPairs* findSumPairsCreate(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}

void findSumPairsAdd(FindSumPairs* obj, int index, int val) {

}

int findSumPairsCount(FindSumPairs* obj, int tot) {

}

void findSumPairsFree(FindSumPairs* obj) {

}

/**
* Your FindSumPairs struct will be instantiated and called as such:
* FindSumPairs* obj = findSumPairsCreate(nums1, nums1Size, nums2, nums2Size);
* findSumPairsAdd(obj, index, val);

* int param_2 = findSumPairsCount(obj, tot);

* findSumPairsFree(obj);
*/
```

Go:

```
type FindSumPairs struct {

}


func Constructor(nums1 []int, nums2 []int) FindSumPairs {

}


func (this *FindSumPairs) Add(index int, val int) {

}


func (this *FindSumPairs) Count(tot int) int {

}


/**
* Your FindSumPairs object will be instantiated and called as such:
* obj := Constructor(nums1, nums2);
* obj.Add(index,val);
* param_2 := obj.Count(tot);
*/
```

Kotlin:

```
class FindSumPairs(nums1: IntArray, nums2: IntArray) {

fun add(index: Int, `val`: Int) {

}

fun count(tot: Int): Int {

}

}
```

```
/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = FindSumPairs(nums1, nums2)
* obj.add(index,`val`)
* var param_2 = obj.count(tot)
*/
```

Swift:

```
class FindSumPairs {

init(_ nums1: [Int], _ nums2: [Int]) {

}

func add(_ index: Int, _ val: Int) {

}

func count(_ tot: Int) -> Int {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* let obj = FindSumPairs(nums1, nums2)
* obj.add(index, val)
* let ret_2: Int = obj.count(tot)
*/
```

Rust:

```
struct FindSumPairs {

}


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

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

fn new(nums1: Vec<i32>, nums2: Vec<i32>) -> Self {

}

fn add(&self, index: i32, val: i32) {

}

fn count(&self, tot: i32) -> i32 {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* let obj = FindSumPairs::new(nums1, nums2);
* obj.add(index, val);
* let ret_2: i32 = obj.count(tot);
*/
```

Ruby:

```
class FindSumPairs

=begin
:type nums1: Integer[]
:type nums2: Integer[]
=end
def initialize(nums1, nums2)

end


=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
```

```
def add(index, val)

end


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

end



end

# Your FindSumPairs object will be instantiated and called as such:
# obj = FindSumPairs.new(nums1, nums2)
# obj.add(index, val)
# param_2 = obj.count(tot)
```

PHP:

```
class FindSumPairs {
/**
* @param Integer[] $nums1
* @param Integer[] $nums2
*/
function __construct($nums1, $nums2) {

}

/**
* @param Integer $index
* @param Integer $val
* @return NULL
*/
function add($index, $val) {

}

/**
```

```
* @param Integer $tot
* @return Integer
*/
function count($tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* $obj = FindSumPairs($nums1, $nums2);
* $obj->add($index, $val);
* $ret_2 = $obj->count($tot);
*/
```

Dart:

```
class FindSumPairs {

FindSumPairs(List<int> nums1, List<int> nums2) {

}

void add(int index, int val) {

}

int count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs obj = FindSumPairs(nums1, nums2);
* obj.add(index,val);
* int param2 = obj.count(tot);
*/
```

Scala:

```
class FindSumPairs(_nums1: Array[Int], _nums2: Array[Int]) {

def add(index: Int, `val`: Int): Unit = {

}

def count(tot: Int): Int = {

}

}

/**
* Your FindSumPairs object will be instantiated and called as such:
* val obj = new FindSumPairs(nums1, nums2)
* obj.add(index,`val`)
* val param_2 = obj.count(tot)
*/
```

Elixir:

```
defmodule FindSumPairs do
@spec init_(nums1 :: [integer], nums2 :: [integer]) :: any
def init_(nums1, nums2) do

end

@spec add(index :: integer, val :: integer) :: any
def add(index, val) do

end

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

end
end

# Your functions will be called as such:
# FindSumPairs.init_(nums1, nums2)
# FindSumPairs.add(index, val)
# param_2 = FindSumPairs.count(tot)
```

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

Erlang:

```
-spec find_sum_pairs_init_(Nums1 :: [integer()], Nums2 :: [integer()]) ->
any().
find_sum_pairs_init_(Nums1, Nums2) ->
.

-spec find_sum_pairs_add(Index :: integer(), Val :: integer()) -> any().
find_sum_pairs_add(Index, Val) ->
.

-spec find_sum_pairs_count(Tot :: integer()) -> integer().
find_sum_pairs_count(Tot) ->
.


%% Your functions will be called as such:
%% find_sum_pairs_init_(Nums1, Nums2),
%% find_sum_pairs_add(Index, Val),
%% Param_2 = find_sum_pairs_count(Tot),

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

Racket:

```
(define find-sum-pairs%
(class object%
(super-new)

; nums1 : (listof exact-integer?)
; nums2 : (listof exact-integer?)
(init-field
nums1
nums2)

; add : exact-integer? exact-integer? -> void?
(define/public (add index val)
)
```

```
; count : exact-integer? -> exact-integer?
(define/public (count tot)
)))

;; Your find-sum-pairs% object will be instantiated and called as such:
;; (define obj (new find-sum-pairs% [nums1 nums1] [nums2 nums2]))
;; (send obj add index val)
;; (define param_2 (send obj count tot))
```

Solutions

C++ Solution:

```
/*
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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 FindSumPairs {
public:
FindSumPairs(vector<int>& nums1, vector<int>& nums2) {

}

void add(int index, int val) {

}

int count(int tot) {

}
};

/**
* Your FindSumPairs object will be instantiated and called as such:
```

```
* FindSumPairs* obj = new FindSumPairs(nums1, nums2);
* obj->add(index,val);
* int param_2 = obj->count(tot);
*/
```

Java Solution:

```
/**
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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 FindSumPairs {

public FindSumPairs(int[] nums1, int[] nums2) {

}

public void add(int index, int val) {

}

public int count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs obj = new FindSumPairs(nums1, nums2);
* obj.add(index,val);
* int param_2 = obj.count(tot);
*/
```

Python3 Solution:

```
"""
Problem: Finding Pairs With a Certain Sum
Difficulty: Medium
Tags: array, 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 FindSumPairs:

def __init__(self, nums1: List[int], nums2: List[int]):



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

Python Solution:

```
class FindSumPairs(object):

def __init__(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
"""



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



def count(self, tot):
"""
:type tot: int
:rtype: int
```

```
"""



# Your FindSumPairs object will be instantiated and called as such:
# obj = FindSumPairs(nums1, nums2)
# obj.add(index,val)
# param_2 = obj.count(tot)
```

JavaScript Solution:

```
/**
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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[]} nums1
* @param {number[]} nums2
*/
var FindSumPairs = function(nums1, nums2) {

};

/**
* @param {number} index
* @param {number} val
* @return {void}
*/
FindSumPairs.prototype.add = function(index, val) {

};

/**
* @param {number} tot
* @return {number}
```

```
*/
FindSumPairs.prototype.count = function(tot) {

};

/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = new FindSumPairs(nums1, nums2)
* obj.add(index,val)
* var param_2 = obj.count(tot)
*/
```

TypeScript Solution:

```
/**
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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 FindSumPairs {
constructor(nums1: number[], nums2: number[]) {

}

add(index: number, val: number): void {

}

count(tot: number): number {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = new FindSumPairs(nums1, nums2)
```

```
* obj.add(index,val)
* var param_2 = obj.count(tot)
*/
```

C# Solution:

```
/*
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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 FindSumPairs {

public FindSumPairs(int[] nums1, int[] nums2) {

}

public void Add(int index, int val) {

}

public int Count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs obj = new FindSumPairs(nums1, nums2);
* obj.Add(index,val);
* int param_2 = obj.Count(tot);
*/
```

C Solution:

```c
/*
* Problem: Finding Pairs With a Certain Sum
* Difficulty: Medium
* Tags: array, 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 {

} FindSumPairs;


FindSumPairs* findSumPairsCreate(int* nums1, int nums1Size, int* nums2, int
nums2Size) {

}

void findSumPairsAdd(FindSumPairs* obj, int index, int val) {

}

int findSumPairsCount(FindSumPairs* obj, int tot) {

}

void findSumPairsFree(FindSumPairs* obj) {

}

/**
* Your FindSumPairs struct will be instantiated and called as such:
* FindSumPairs* obj = findSumPairsCreate(nums1, nums1Size, nums2, nums2Size);
* findSumPairsAdd(obj, index, val);

* int param_2 = findSumPairsCount(obj, tot);

* findSumPairsFree(obj);
```

```
*/
```

Go Solution:

```
// Problem: Finding Pairs With a Certain Sum
// Difficulty: Medium
// Tags: array, 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 FindSumPairs struct {

}


func Constructor(nums1 []int, nums2 []int) FindSumPairs {

}


func (this *FindSumPairs) Add(index int, val int) {

}


func (this *FindSumPairs) Count(tot int) int {

}


/**
* Your FindSumPairs object will be instantiated and called as such:
* obj := Constructor(nums1, nums2);
* obj.Add(index,val);
* param_2 := obj.Count(tot);
*/
```

Kotlin Solution:

```
class FindSumPairs(nums1: IntArray, nums2: IntArray) {

fun add(index: Int, `val`: Int) {

}

fun count(tot: Int): Int {

}

}

/**
* Your FindSumPairs object will be instantiated and called as such:
* var obj = FindSumPairs(nums1, nums2)
* obj.add(index,`val`)
* var param_2 = obj.count(tot)
*/
```

Swift Solution:

```
class FindSumPairs {

init(_ nums1: [Int], _ nums2: [Int]) {

}

func add(_ index: Int, _ val: Int) {

}

func count(_ tot: Int) -> Int {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* let obj = FindSumPairs(nums1, nums2)
* obj.add(index, val)
* let ret_2: Int = obj.count(tot)
```

```
*/
```

Rust Solution:

```
// Problem: Finding Pairs With a Certain Sum
// Difficulty: Medium
// Tags: array, 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 FindSumPairs {

}



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

fn new(nums1: Vec<i32>, nums2: Vec<i32>) -> Self {

}

fn add(&self, index: i32, val: i32) {

}

fn count(&self, tot: i32) -> i32 {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* let obj = FindSumPairs::new(nums1, nums2);
* obj.add(index, val);
* let ret_2: i32 = obj.count(tot);
```

```
*/
```

Ruby Solution:

```
class FindSumPairs

=begin
:type nums1: Integer[]
:type nums2: Integer[]
=end
def initialize(nums1, nums2)

end


=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
def add(index, val)

end


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

end


end

# Your FindSumPairs object will be instantiated and called as such:
# obj = FindSumPairs.new(nums1, nums2)
# obj.add(index, val)
# param_2 = obj.count(tot)
```

PHP Solution:

```
class FindSumPairs {
/**
* @param Integer[] $nums1
* @param Integer[] $nums2
*/
function __construct($nums1, $nums2) {

}

/**
* @param Integer $index
* @param Integer $val
* @return NULL
*/
function add($index, $val) {

}

/**
* @param Integer $tot
* @return Integer
*/
function count($tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* $obj = FindSumPairs($nums1, $nums2);
* $obj->add($index, $val);
* $ret_2 = $obj->count($tot);
*/
```

Dart Solution:

```
class FindSumPairs {

FindSumPairs(List<int> nums1, List<int> nums2) {

}
```

```
void add(int index, int val) {

}

int count(int tot) {

}
}

/**
* Your FindSumPairs object will be instantiated and called as such:
* FindSumPairs obj = FindSumPairs(nums1, nums2);
* obj.add(index,val);
* int param2 = obj.count(tot);
*/
```

Scala Solution:

```
class FindSumPairs(_nums1: Array[Int], _nums2: Array[Int]) {

def add(index: Int, `val`: Int): Unit = {

}

def count(tot: Int): Int = {

}

}

/**
* Your FindSumPairs object will be instantiated and called as such:
* val obj = new FindSumPairs(nums1, nums2)
* obj.add(index,`val`)
* val param_2 = obj.count(tot)
*/
```

Elixir Solution:

```
defmodule FindSumPairs do
@spec init_(nums1 :: [integer], nums2 :: [integer]) :: any
def init_(nums1, nums2) do

end

@spec add(index :: integer, val :: integer) :: any
def add(index, val) do

end

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

end
end

# Your functions will be called as such:
# FindSumPairs.init_(nums1, nums2)
# FindSumPairs.add(index, val)
# param_2 = FindSumPairs.count(tot)

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

Erlang Solution:

```
-spec find_sum_pairs_init_(Nums1 :: [integer()], Nums2 :: [integer()]) ->
any().
find_sum_pairs_init_(Nums1, Nums2) ->
.

-spec find_sum_pairs_add(Index :: integer(), Val :: integer()) -> any().
find_sum_pairs_add(Index, Val) ->
.

-spec find_sum_pairs_count(Tot :: integer()) -> integer().
find_sum_pairs_count(Tot) ->
.


%% Your functions will be called as such:
```

```
%% find_sum_pairs_init_(Nums1, Nums2),
%% find_sum_pairs_add(Index, Val),
%% Param_2 = find_sum_pairs_count(Tot),

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

Racket Solution:

```
(define find-sum-pairs%
(class object%
(super-new)

; nums1 : (listof exact-integer?)
; nums2 : (listof exact-integer?)
(init-field
nums1
nums2)

; add : exact-integer? exact-integer? -> void?
(define/public (add index val)
)
; count : exact-integer? -> exact-integer?
(define/public (count tot)
)))

;; Your find-sum-pairs% object will be instantiated and called as such:
;; (define obj (new find-sum-pairs% [nums1 nums1] [nums2 nums2]))
;; (send obj add index val)
;; (define param_2 (send obj count tot))
```