Problem 1570: Dot Product of Two Sparse Vectors

Problem Information

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

Problem Description

Given two sparse vectors, compute their dot product.

Implement class

SparseVector

:

SparseVector(nums)

Initializes the object with the vector

nums

dotProduct(vec)

Compute the dot product between the instance of

SparseVector

and

vec

A

sparse vector

is a vector that has mostly zero values, you should store the sparse vector

efficiently

and compute the dot product between two

SparseVector

.

Follow up:

What if only one of the vectors is sparse?

Example 1:

Input:

nums1 = [1,0,0,2,3], nums2 = [0,3,0,4,0]

Output:

8

Explanation:

v1 = SparseVector(nums1) , v2 = SparseVector(nums2) v1.dotProduct(v2) = 1*0 + 0*3 + 0*0 + 2*4 + 3*0 = 8

Example 2:

Input:

nums1 = [0,1,0,0,0], nums2 = [0,0,0,0,2]

Output:

0

Explanation:

v1 = SparseVector(nums1) , v2 = SparseVector(nums2) v1.dotProduct(v2) = 0*0 + 1*0 + 0*0 + 0*0 + 0*2 = 0

Example 3:

Input:

nums1 = [0,1,0,0,2,0,0], nums2 = [1,0,0,0,3,0,4]

Output:

6

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10^5

0 <= nums1[i], nums2[i] <= 100

Code Snippets

C++:

```
class SparseVector {
public:

SparseVector(vector<int> &nums) {

}

// Return the dotProduct of two sparse vectors
int dotProduct(SparseVector& vec) {
}
};

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1(nums1);
// SparseVector v2(nums2);
// int ans = v1.dotProduct(v2);
```

Java:

```
class SparseVector {

SparseVector(int[] nums) {

}

// Return the dotProduct of two sparse vectors
public int dotProduct(SparseVector vec) {

}
}

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1 = new SparseVector(nums1);
// SparseVector v2 = new SparseVector(nums2);
// int ans = v1.dotProduct(v2);
```

Python3:

```
class SparseVector:
def __init__(self, nums: List[int]):


# Return the dotProduct of two sparse vectors
def dotProduct(self, vec: 'SparseVector') -> int:


# Your SparseVector object will be instantiated and called as such:
# v1 = SparseVector(nums1)
# v2 = SparseVector(nums2)
# ans = v1.dotProduct(v2)
```

Python:

```
class SparseVector:
def __init__(self, nums):
"""
:type nums: List[int]
"""



# Return the dotProduct of two sparse vectors
def dotProduct(self, vec):
"""
:type vec: 'SparseVector'
:rtype: int
"""



# Your SparseVector object will be instantiated and called as such:
# v1 = SparseVector(nums1)
# v2 = SparseVector(nums2)
# ans = v1.dotProduct(v2)
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {SparseVector}
*/
var SparseVector = function(nums) {

};

// Return the dotProduct of two sparse vectors
/**
* @param {SparseVector} vec
* @return {number}
*/
SparseVector.prototype.dotProduct = function(vec) {

};

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

```
// let v1 = new SparseVector(nums1);
// let v2 = new SparseVector(nums2);
// let ans = v1.dotProduct(v2);
```

TypeScript:

```
class SparseVector {
constructor(nums: number[]) {

}

// Return the dotProduct of two sparse vectors
dotProduct(vec: SparseVector): number {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = new SparseVector(nums1)
* var v2 = new SparseVector(nums1)
* var ans = v1.dotProduct(v2)
*/
```

C#:

```
public class SparseVector {

public SparseVector(int[] nums) {

}

// Return the dotProduct of two sparse vectors
public int DotProduct(SparseVector vec) {

}
}

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1 = new SparseVector(nums1);
// SparseVector v2 = new SparseVector(nums2);
// int ans = v1.DotProduct(v2);
```

C:

```
typedef struct {

} SparseVector;


SparseVector* sparseVectorCreate(int* nums, int numsSize) {

}

// Return the dotProduct of two sparse vectors
int sparseVectordotProduct(SparseVector* obj, SparseVector* vec) {

}

/**
* Your SparseVector struct will be instantiated and called as such:
* SparseVector* v1 = sparseVectorCreate(nums1, nums1Size);
* SparseVector* v2 = sparseVectorCreate(nums2, nums2Size);
* int ans = sparseVectordotProduct(v1, v2);
*/
```

Go:

```
type SparseVector struct {

}

func Constructor(nums []int) SparseVector {

}

// Return the dotProduct of two sparse vectors
func (this *SparseVector) dotProduct(vec SparseVector) int {

}

/**
```

```

* Your SparseVector object will be instantiated and called as such:
* v1 := Constructor(nums1);
* v2 := Constructor(nums2);
* ans := v1.dotProduct(v2);
*/
```

Kotlin:

```
class SparseVector(nums: IntArray) {
// Return the dotProduct of two sparse vectors
fun dotProduct(vec: SparseVector): Int {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = SparseVector(nums1)
* var v2 = SparseVector(nums2)
* val ans = v1.dotProduct(v2)
*/
```

Swift:

```

class SparseVector {

init(_ nums: [Int]) {

}

// Return the dotProduct of two sparse vectors
func dotProduct(_ vec: SparseVector) -> Int {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* let v1 = SparseVector(nums1)
* let v2 = SparseVector(nums2)
* let ans = v1.dotProduct(v2)
```

```
*/
```

Rust:

```
struct SparseVector {

}

/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SparseVector {
fn new(nums: Vec<i32>) -> Self {

}

// Return the dotProduct of two sparse vectors
fn dot_product(&self, vec: SparseVector) -> i32 {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* let v1 = SparseVector::new(nums1);
* let v2 = SparseVector::new(nums2);
* let ans = v1.dot_product(v2);
*/
```

Ruby:

```
class SparseVector

=begin
:type nums: Integer[]
=end
def initialize(nums)

end

# Return the dotProduct of two sparse vectors
```

```
=begin
:type vec: SparseVector
:rtype: Integer
=end
def dotProduct(vec)

end
end

# Your SparseVector object will be instantiated and called as such:
# v1 = SparseVector.new(nums1)
# v2 = SparseVector.new(nums2)
# ans = v1.dotProduct(v2)
```

PHP:

```
class SparseVector {
/**
* @param Integer[] $nums
*/
function __construct($nums) {

}

// Return the dotProduct of two sparse vectors
/**
* @param SparseVector $vec
* @return Integer
*/
function dotProduct($vec) {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* $v1 = new SparseVector($nums1);
* $v2 = new SparseVector($nums2);
* $ans = $v1->dotProduct($v2);
*/
```

Scala:

```
class SparseVector(nums: Array[Int]) {
// Return the dotProduct of two sparse vectors
def dotProduct(vec: SparseVector): Int = {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = new SparseVector(nums1)
* var v2 = new SparseVector(nums2)
* val ans = v1.dotProduct(v2)
*/
```

Solutions

C++ Solution:

```
/*
* Problem: Dot Product of Two Sparse Vectors
* 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 SparseVector {
public:

SparseVector(vector<int> &nums) {

}

// Return the dotProduct of two sparse vectors
int dotProduct(SparseVector& vec) {

}
};

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1(nums1);
// SparseVector v2(nums2);
// int ans = v1.dotProduct(v2);
```

Java Solution:

```
/**
* Problem: Dot Product of Two Sparse Vectors
* 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 SparseVector {

SparseVector(int[] nums) {

}

// Return the dotProduct of two sparse vectors
public int dotProduct(SparseVector vec) {

}
}

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1 = new SparseVector(nums1);
// SparseVector v2 = new SparseVector(nums2);
// int ans = v1.dotProduct(v2);
```

Python3 Solution:

```
"""
Problem: Dot Product of Two Sparse Vectors
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 SparseVector:
def __init__(self, nums: List[int]):



# Return the dotProduct of two sparse vectors
def dotProduct(self, vec: 'SparseVector') -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SparseVector:
def __init__(self, nums):
"""
:type nums: List[int]
"""



# Return the dotProduct of two sparse vectors
def dotProduct(self, vec):
"""
:type vec: 'SparseVector'
:rtype: int
"""



# Your SparseVector object will be instantiated and called as such:
# v1 = SparseVector(nums1)
# v2 = SparseVector(nums2)
# ans = v1.dotProduct(v2)
```

JavaScript Solution:

```
/**
* Problem: Dot Product of Two Sparse Vectors
* 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[]} nums
* @return {SparseVector}
*/
var SparseVector = function(nums) {

};

// Return the dotProduct of two sparse vectors
/**
* @param {SparseVector} vec
* @return {number}
*/
SparseVector.prototype.dotProduct = function(vec) {

};

// Your SparseVector object will be instantiated and called as such:
// let v1 = new SparseVector(nums1);
// let v2 = new SparseVector(nums2);
// let ans = v1.dotProduct(v2);
```

TypeScript Solution:

```
/**
* Problem: Dot Product of Two Sparse Vectors
* 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 SparseVector {
constructor(nums: number[]) {

}

// Return the dotProduct of two sparse vectors
dotProduct(vec: SparseVector): number {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = new SparseVector(nums1)
* var v2 = new SparseVector(nums1)
* var ans = v1.dotProduct(v2)
*/
```

C# Solution:

```
/*
* Problem: Dot Product of Two Sparse Vectors
* 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 SparseVector {

public SparseVector(int[] nums) {

}

// Return the dotProduct of two sparse vectors
public int DotProduct(SparseVector vec) {
}
}

// Your SparseVector object will be instantiated and called as such:
// SparseVector v1 = new SparseVector(nums1);
// SparseVector v2 = new SparseVector(nums2);
// int ans = v1.DotProduct(v2);
```

C Solution:

```
/*
* Problem: Dot Product of Two Sparse Vectors
* 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 {

} SparseVector;


SparseVector* sparseVectorCreate(int* nums, int numsSize) {

}

// Return the dotProduct of two sparse vectors
int sparseVectordotProduct(SparseVector* obj, SparseVector* vec) {

}

/**
* Your SparseVector struct will be instantiated and called as such:
* SparseVector* v1 = sparseVectorCreate(nums1, nums1Size);
* SparseVector* v2 = sparseVectorCreate(nums2, nums2Size);
* int ans = sparseVectordotProduct(v1, v2);
*/
```

Go Solution:

```
// Problem: Dot Product of Two Sparse Vectors
// 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 SparseVector struct {

}

func Constructor(nums []int) SparseVector {

}

// Return the dotProduct of two sparse vectors
func (this *SparseVector) dotProduct(vec SparseVector) int {

}

/**
* Your SparseVector object will be instantiated and called as such:
* v1 := Constructor(nums1);
* v2 := Constructor(nums2);
* ans := v1.dotProduct(v2);
*/
```

Kotlin Solution:

```
class SparseVector(nums: IntArray) {
// Return the dotProduct of two sparse vectors
fun dotProduct(vec: SparseVector): Int {

}
}
```

```
/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = SparseVector(nums1)
* var v2 = SparseVector(nums2)
* val ans = v1.dotProduct(v2)
*/
```

Swift Solution:

```
class SparseVector {

init(_ nums: [Int]) {

}

// Return the dotProduct of two sparse vectors
func dotProduct(_ vec: SparseVector) -> Int {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* let v1 = SparseVector(nums1)
* let v2 = SparseVector(nums2)
* let ans = v1.dotProduct(v2)
*/
```

Rust Solution:

```
// Problem: Dot Product of Two Sparse Vectors
// 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 SparseVector {
}

/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SparseVector {
fn new(nums: Vec<i32>) -> Self {

}

// Return the dotProduct of two sparse vectors
fn dot_product(&self, vec: SparseVector) -> i32 {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* let v1 = SparseVector::new(nums1);
* let v2 = SparseVector::new(nums2);
* let ans = v1.dot_product(v2);
*/
```

Ruby Solution:

```
class SparseVector

=begin
:type nums: Integer[]
=end
def initialize(nums)

end

# Return the dotProduct of two sparse vectors
=begin
:type vec: SparseVector
:rtype: Integer
=end
```

```
def dotProduct(vec)

end
end

# Your SparseVector object will be instantiated and called as such:
# v1 = SparseVector.new(nums1)
# v2 = SparseVector.new(nums2)
# ans = v1.dotProduct(v2)
```

PHP Solution:

```
class SparseVector {
/**
* @param Integer[] $nums
*/
function __construct($nums) {

}

// Return the dotProduct of two sparse vectors
/**
* @param SparseVector $vec
* @return Integer
*/
function dotProduct($vec) {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* $v1 = new SparseVector($nums1);
* $v2 = new SparseVector($nums2);
* $ans = $v1->dotProduct($v2);
*/
```

Scala Solution:

```
class SparseVector(nums: Array[Int]) {
// Return the dotProduct of two sparse vectors
def dotProduct(vec: SparseVector): Int = {

}
}

/**
* Your SparseVector object will be instantiated and called as such:
* var v1 = new SparseVector(nums1)
* var v2 = new SparseVector(nums2)
* val ans = v1.dotProduct(v2)
*/
```