

Problem 1722: Minimize Hamming Distance After Swap Operations

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two integer arrays,

source

and

target

, both of length

n

. You are also given an array

allowedSwaps

where each

allowedSwaps[i] = [a

i

, b

i

]

indicates that you are allowed to swap the elements at index

a

i

and index

b

i

(0-indexed)

of array

source

. Note that you can swap elements at a specific pair of indices

multiple

times and in

any

order.

The

Hamming distance

of two arrays of the same length,

source

and

target

, is the number of positions where the elements are different. Formally, it is the number of indices

i

for

$0 \leq i \leq n-1$

where

$\text{source}[i] \neq \text{target}[i]$

(0-indexed)

.

Return

the

minimum Hamming distance

of

source

and

target

after performing

any

amount of swap operations on array

source

.

Example 1:

Input:

source = [1,2,3,4], target = [2,1,4,5], allowedSwaps = [[0,1],[2,3]]

Output:

1

Explanation:

source can be transformed the following way: - Swap indices 0 and 1: source = [

2

,

1

,3,4] - Swap indices 2 and 3: source = [2,1,

4

,

3

] The Hamming distance of source and target is 1 as they differ in 1 position: index 3.

Example 2:

Input:

```
source = [1,2,3,4], target = [1,3,2,4], allowedSwaps = []
```

Output:

2

Explanation:

There are no allowed swaps. The Hamming distance of source and target is 2 as they differ in 2 positions: index 1 and index 2.

Example 3:

Input:

```
source = [5,1,2,4,3], target = [1,5,4,2,3], allowedSwaps = [[0,4],[4,2],[1,3],[1,4]]
```

Output:

0

Constraints:

$n == \text{source.length} == \text{target.length}$

$1 \leq n \leq 10$

5

$1 \leq \text{source}[i], \text{target}[i] \leq 10$

5

$0 \leq \text{allowedSwaps.length} \leq 10$

5

$\text{allowedSwaps}[i].length == 2$

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

Code Snippets

C++:

```
class Solution {
public:
    int minimumHammingDistance(vector<int>& source, vector<int>& target,
    vector<vector<int>>& allowedSwaps) {

    }
};
```

Java:

```
class Solution {
    public int minimumHammingDistance(int[] source, int[] target, int[][][]
    allowedSwaps) {

    }
}
```

Python3:

```
class Solution:
    def minimumHammingDistance(self, source: List[int], target: List[int],
                               allowedSwaps: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
    def minimumHammingDistance(self, source, target, allowedSwaps):
        """
        :type source: List[int]
        :type target: List[int]
        :type allowedSwaps: List[List[int]]
        :rtype: int
        """

```

JavaScript:

```
/**
 * @param {number[]} source
 * @param {number[]} target
 * @param {number[][]} allowedSwaps
 * @return {number}
 */
var minimumHammingDistance = function(source, target, allowedSwaps) {
}
```

TypeScript:

```
function minimumHammingDistance(source: number[], target: number[],
                                allowedSwaps: number[][]): number {
}
```

C#:

```
public class Solution {
    public int MinimumHammingDistance(int[] source, int[] target, int[][] allowedSwaps) {
}
```

```
}
```

C:

```
int minimumHammingDistance(int* source, int sourceSize, int* target, int
targetSize, int** allowedSwaps, int allowedSwapsSize, int*
allowedSwapsColSize) {

}
```

Go:

```
func minimumHammingDistance(source []int, target []int, allowedSwaps [][]int)
int {

}
```

Kotlin:

```
class Solution {
    fun minimumHammingDistance(source: IntArray, target: IntArray, allowedSwaps:
    Array<IntArray>): Int {
        }
    }
```

Swift:

```
class Solution {
    func minimumHammingDistance(_ source: [Int], _ target: [Int], _ allowedSwaps:
    [[[Int]]]) -> Int {
        }
    }
```

Rust:

```
impl Solution {
    pub fn minimum_hamming_distance(source: Vec<i32>, target: Vec<i32>,
    allowed_swaps: Vec<Vec<i32>>) -> i32 {
        }
```

```
}
```

Ruby:

```
# @param {Integer[]} source
# @param {Integer[]} target
# @param {Integer[][]} allowed_swaps
# @return {Integer}
def minimum_hamming_distance(source, target, allowed_swaps)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $source
     * @param Integer[] $target
     * @param Integer[][] $allowedSwaps
     * @return Integer
     */
    function minimumHammingDistance($source, $target, $allowedSwaps) {

    }
}
```

Dart:

```
class Solution {
int minimumHammingDistance(List<int> source, List<int> target,
List<List<int>> allowedSwaps) {

}
```

Scala:

```
object Solution {
def minimumHammingDistance(source: Array[Int], target: Array[Int],
allowedSwaps: Array[Array[Int]]): Int = {
```

```
}
```

```
}
```

Elixir:

```
defmodule Solution do
  @spec minimum_hamming_distance(source :: [integer], target :: [integer],
  allowed_swaps :: [[integer]]) :: integer
  def minimum_hamming_distance(source, target, allowed_swaps) do
    end
  end
end
```

Erlang:

```
-spec minimum_hamming_distance(Source :: [integer()], Target :: [integer()],
  AllowedSwaps :: [[integer()]]) -> integer().
minimum_hamming_distance(Source, Target, AllowedSwaps) ->
  .
```

Racket:

```
(define/contract (minimum-hamming-distance source target allowedSwaps)
  (-> (listof exact-integer?) (listof exact-integer?) (listof (listof
  exact-integer?)) exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */
```

```

class Solution {
public:
    int minimumHammingDistance(vector<int>& source, vector<int>& target,
    vector<vector<int>>& allowedSwaps) {

    }
};

```

Java Solution:

```

/**
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public int minimumHammingDistance(int[] source, int[] target, int[][] allowedSwaps) {

}
}

```

Python3 Solution:

```

"""
Problem: Minimize Hamming Distance After Swap Operations
Difficulty: Medium
Tags: array, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

```

```
def minimumHammingDistance(self, source: List[int], target: List[int],
                           allowedSwaps: List[List[int]]) -> int:
    # TODO: Implement optimized solution
    pass
```

Python Solution:

```
class Solution(object):
    def minimumHammingDistance(self, source, target, allowedSwaps):
        """
        :type source: List[int]
        :type target: List[int]
        :type allowedSwaps: List[List[int]]
        :rtype: int
        """
```

JavaScript Solution:

```
/**
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} source
 * @param {number[]} target
 * @param {number[][]} allowedSwaps
 * @return {number}
 */
var minimumHammingDistance = function(source, target, allowedSwaps) {
```

```
};
```

TypeScript Solution:

```

/**
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function minimumHammingDistance(source: number[], target: number[],
allowedSwaps: number[][]): number {

};

```

C# Solution:

```

/*
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int MinimumHammingDistance(int[] source, int[] target, int[][] allowedSwaps) {

    }
}

```

C Solution:

```

/*
 * Problem: Minimize Hamming Distance After Swap Operations
 * Difficulty: Medium
 * Tags: array, graph, search
 *
 * Approach: Use two pointers or sliding window technique

```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
int minimumHammingDistance(int* source, int sourceSize, int* target, int
targetSize, int** allowedSwaps, int allowedSwapsSize, int*
allowedSwapsColSize) {

}

```

Go Solution:

```

// Problem: Minimize Hamming Distance After Swap Operations
// Difficulty: Medium
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minimumHammingDistance(source []int, target []int, allowedSwaps [][]int)
int {
}

}

```

Kotlin Solution:

```

class Solution {
    fun minimumHammingDistance(source: IntArray, target: IntArray, allowedSwaps:
    Array<IntArray>): Int {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func minimumHammingDistance(_ source: [Int], _ target: [Int], _ allowedSwaps:
    [[Int]]) -> Int {
        }
}

```

```
}
```

Rust Solution:

```
// Problem: Minimize Hamming Distance After Swap Operations
// Difficulty: Medium
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn minimum_hamming_distance(source: Vec<i32>, target: Vec<i32>,
        allowed_swaps: Vec<Vec<i32>>) -> i32 {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[]} source
# @param {Integer[]} target
# @param {Integer[][]} allowed_swaps
# @return {Integer}
def minimum_hamming_distance(source, target, allowed_swaps)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $source
     * @param Integer[] $target
     * @param Integer[][] $allowedSwaps
     * @return Integer
     */
    function minimumHammingDistance($source, $target, $allowedSwaps) {
```

```
}
```

```
}
```

Dart Solution:

```
class Solution {  
    int minimumHammingDistance(List<int> source, List<int> target,  
        List<List<int>> allowedSwaps) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def minimumHammingDistance(source: Array[Int], target: Array[Int],  
        allowedSwaps: Array[Array[Int]]): Int = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec minimum_hamming_distance(source :: [integer], target :: [integer],  
        allowed_swaps :: [[integer]]) :: integer  
    def minimum_hamming_distance(source, target, allowed_swaps) do  
  
    end  
end
```

Erlang Solution:

```
-spec minimum_hamming_distance(Source :: [integer()], Target :: [integer()],  
    AllowedSwaps :: [[integer()]]) -> integer().  
minimum_hamming_distance(Source, Target, AllowedSwaps) ->  
.
```

Racket Solution:

```
(define/contract (minimum-hamming-distance source target allowedSwaps)
  (-> (listof exact-integer?) (listof exact-integer?) (listof (listof
    exact-integer?)) exact-integer?)
  )
```