Problem 1007: Minimum Domino Rotations For Equal Row

Problem Information

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

Problem Description

In a row of dominoes,

tops[i]

and

bottoms[i]

represent the top and bottom halves of the

i

th

domino. (A domino is a tile with two numbers from 1 to 6 - one on each half of the tile.)

We may rotate the

i

th

domino, so that

tops[i]

and

bottoms[i]

swap values.

Return the minimum number of rotations so that all the values in

tops

are the same, or all the values in

bottoms

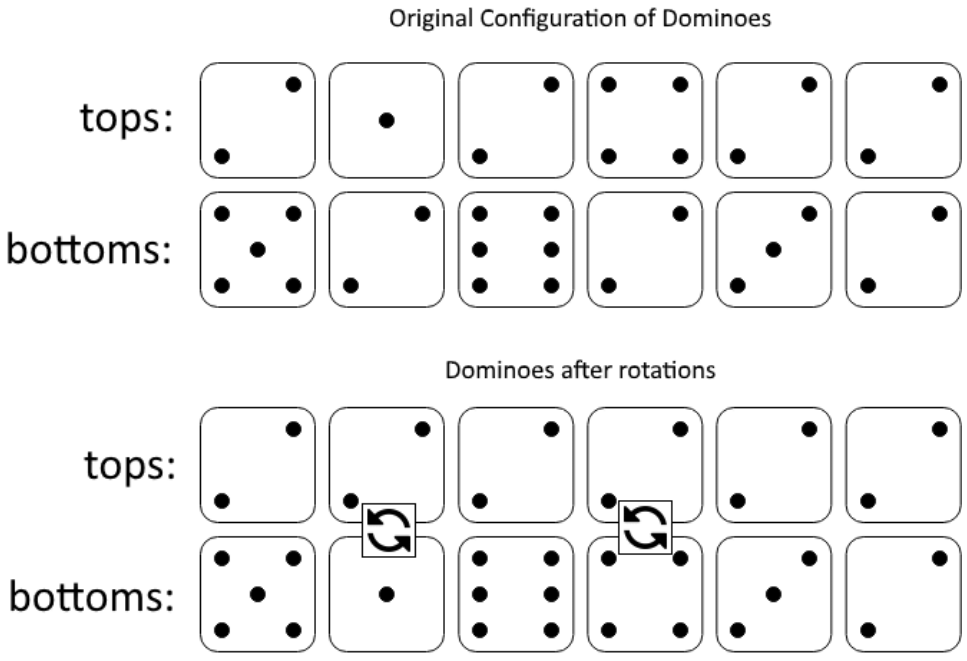
are the same.

If it cannot be done, return

-1

.

Example 1:



Input:

tops = [2,1,2,4,2,2], bottoms = [5,2,6,2,3,2]

Output:

2

Explanation:

The first figure represents the dominoes as given by tops and bottoms: before we do any rotations. If we rotate the second and fourth dominoes, we can make every value in the top row equal to 2, as indicated by the second figure.

Example 2:

Input:

tops = [3,5,1,2,3], bottoms = [3,6,3,3,4]

Output:

-1

Explanation:

In this case, it is not possible to rotate the dominoes to make one row of values equal.

Constraints:

2 <= tops.length <= 2 * 10

4

bottoms.length == tops.length

1 <= tops[i], bottoms[i] <= 6

Code Snippets

C++:

```
class Solution {
public:
int minDominoRotations(vector<int>& tops, vector<int>& bottoms) {

}
};
```

Java:

```
class Solution {
public int minDominoRotations(int[] tops, int[] bottoms) {

}
}
```

Python3:

```
class Solution:
def minDominoRotations(self, tops: List[int], bottoms: List[int]) -> int:
```

Python:

```
class Solution(object):
def minDominoRotations(self, tops, bottoms):
"""
:type tops: List[int]
:type bottoms: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} tops
* @param {number[]} bottoms
* @return {number}
*/
var minDominoRotations = function(tops, bottoms) {
```

```
};
```

TypeScript:

```
function minDominoRotations(tops: number[], bottoms: number[]): number {

};
```

C#:

```
public class Solution {
public int MinDominoRotations(int[] tops, int[] bottoms) {

}
}
```

C:

```
int minDominoRotations(int* tops, int topsSize, int* bottoms, int
bottomsSize) {

}
```

Go:

```
func minDominoRotations(tops []int, bottoms []int) int {

}
```

Kotlin:

```
class Solution {
fun minDominoRotations(tops: IntArray, bottoms: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minDominoRotations(_ tops: [Int], _ bottoms: [Int]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn min_domino_rotations(tops: Vec<i32>, bottoms: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tops
# @param {Integer[]} bottoms
# @return {Integer}
def min_domino_rotations(tops, bottoms)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $tops
* @param Integer[] $bottoms
* @return Integer
*/
function minDominoRotations($tops, $bottoms) {

}
}
```

Dart:

```
class Solution {
int minDominoRotations(List<int> tops, List<int> bottoms) {

}
}
```

Scala:

```
object Solution {
def minDominoRotations(tops: Array[Int], bottoms: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_domino_rotations(tops :: [integer], bottoms :: [integer]) ::
integer
def min_domino_rotations(tops, bottoms) do

end
end
```

Erlang:

```
-spec min_domino_rotations(Tops :: [integer()], Bottoms :: [integer()]) ->
integer().
min_domino_rotations(Tops, Bottoms) ->
.
```

Racket:

```
(define/contract (min-domino-rotations tops bottoms)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
* Tags: array, greedy
*
* 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 minDominoRotations(vector<int>& tops, vector<int>& bottoms) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
* Tags: array, greedy
*
* 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 minDominoRotations(int[] tops, int[] bottoms) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Domino Rotations For Equal Row
Difficulty: Medium
Tags: array, greedy

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 minDominoRotations(self, tops: List[int], bottoms: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minDominoRotations(self, tops, bottoms):
"""
:type tops: List[int]
:type bottoms: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
* Tags: array, greedy
*
* 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[]} tops
* @param {number[]} bottoms
* @return {number}
*/
var minDominoRotations = function(tops, bottoms) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
```

```
* Tags: array, greedy
*
* 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 minDominoRotations(tops: number[], bottoms: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
* Tags: array, greedy
*
* 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 MinDominoRotations(int[] tops, int[] bottoms) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Domino Rotations For Equal Row
* Difficulty: Medium
* Tags: array, greedy
*
* 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 minDominoRotations(int* tops, int topsSize, int* bottoms, int
bottomsSize) {

}
```

Go Solution:

```
// Problem: Minimum Domino Rotations For Equal Row
// Difficulty: Medium
// Tags: array, greedy
//
// 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 minDominoRotations(tops []int, bottoms []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDominoRotations(tops: IntArray, bottoms: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minDominoRotations(_ tops: [Int], _ bottoms: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Domino Rotations For Equal Row
// Difficulty: Medium
// Tags: array, greedy
//
```

```
// 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 min_domino_rotations(tops: Vec<i32>, bottoms: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} tops
# @param {Integer[]} bottoms
# @return {Integer}
def min_domino_rotations(tops, bottoms)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tops
* @param Integer[] $bottoms
* @return Integer
*/
function minDominoRotations($tops, $bottoms) {

}
}
```

Dart Solution:

```
class Solution {
int minDominoRotations(List<int> tops, List<int> bottoms) {

}
}
```

Scala Solution:

```
object Solution {
def minDominoRotations(tops: Array[Int], bottoms: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_domino_rotations(tops :: [integer], bottoms :: [integer]) ::
integer
def min_domino_rotations(tops, bottoms) do

end
end
```

Erlang Solution:

```
-spec min_domino_rotations(Tops :: [integer()], Bottoms :: [integer()]) ->
integer().
min_domino_rotations(Tops, Bottoms) ->
.
```

Racket Solution:

```
(define/contract (min-domino-rotations tops bottoms)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```