Problem 2260: Minimum Consecutive Cards to Pick Up

Problem Information

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

Problem Description

You are given an integer array

cards

where

cards[i]

represents the

value

of the

i

th

card. A pair of cards are

matching

if the cards have the

same

value.

Return

the

minimum

number of

consecutive

cards you have to pick up to have a pair of

matching

cards among the picked cards.

If it is impossible to have matching cards, return

-1

.

Example 1:

Input:

cards = [3,4,2,3,4,7]

Output:

4

Explanation:

We can pick up the cards [3,4,2,3] which contain a matching pair of cards with value 3. Note that picking up the cards [4,2,3,4] is also optimal.

Example 2:

Input:

cards = [1,0,5,3]

Output:

-1

Explanation:

There is no way to pick up a set of consecutive cards that contain a pair of matching cards.

Constraints:

1 <= cards.length <= 10

5

0 <= cards[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int minimumCardPickup(vector<int>& cards) {

}
};
```

Java:

```
class Solution {
public int minimumCardPickup(int[] cards) {
```

```
}
}
```

Python3:

```
class Solution:
def minimumCardPickup(self, cards: List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumCardPickup(self, cards):
"""
:type cards: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minimumCardPickup(cards: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumCardPickup(int[] cards) {

}
}
```

C:

```
int minimumCardPickup(int* cards, int cardsSize) {

}
```

Go:

```
func minimumCardPickup(cards []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumCardPickup(cards: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumCardPickup(_ cards: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_card_pickup(cards: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} cards
# @return {Integer}
def minimum_card_pickup(cards)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $cards
* @return Integer
*/
function minimumCardPickup($cards) {

}
}
```

Dart:

```
class Solution {
int minimumCardPickup(List<int> cards) {

}
}
```

Scala:

```
object Solution {
def minimumCardPickup(cards: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_card_pickup(cards :: [integer]) :: integer
def minimum_card_pickup(cards) do

end
end
```

Erlang:

```
-spec minimum_card_pickup(Cards :: [integer()]) -> integer().
minimum_card_pickup(Cards) ->
.
```

Racket:

```
(define/contract (minimum-card-pickup cards)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Consecutive Cards to Pick Up
* 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 Solution {
public:
int minimumCardPickup(vector<int>& cards) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Consecutive Cards to Pick Up
* 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 Solution {
public int minimumCardPickup(int[] cards) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Consecutive Cards to Pick Up
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 Solution:
def minimumCardPickup(self, cards: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCardPickup(self, cards):
"""
:type cards: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Consecutive Cards to Pick Up
* 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[]} cards
* @return {number}
*/
var minimumCardPickup = function(cards) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Consecutive Cards to Pick Up
* 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
*/

function minimumCardPickup(cards: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Consecutive Cards to Pick Up
* 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 Solution {
public int MinimumCardPickup(int[] cards) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Consecutive Cards to Pick Up
* 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
*/

int minimumCardPickup(int* cards, int cardsSize) {

}
```

Go Solution:

```
// Problem: Minimum Consecutive Cards to Pick Up
// 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

func minimumCardPickup(cards []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCardPickup(cards: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumCardPickup(_ cards: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Consecutive Cards to Pick Up
// 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

impl Solution {
pub fn minimum_card_pickup(cards: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} cards
# @return {Integer}
def minimum_card_pickup(cards)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $cards
* @return Integer
*/
function minimumCardPickup($cards) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCardPickup(List<int> cards) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCardPickup(cards: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_card_pickup(cards :: [integer]) :: integer
def minimum_card_pickup(cards) do

end
end
```

Erlang Solution:

```
-spec minimum_card_pickup(Cards :: [integer()]) -> integer().
minimum_card_pickup(Cards) ->
.
```

Racket Solution:

```
(define/contract (minimum-card-pickup cards)
(-> (listof exact-integer?) exact-integer?)
)
```