Problem 3664: Two-Letter Card Game

Problem Information

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

Problem Description

You are given a deck of cards represented by a string array

cards

, and each card displays two lowercase letters.

You are also given a letter

x

. You play a game with the following rules:

Start with 0 points.

On each turn, you must find two

compatible

cards from the deck that both contain the letter

x

in any position.

Remove the pair of cards and earn

1 point

.

The game ends when you can no longer find a pair of compatible cards.

Return the

maximum

number of points you can gain with optimal play.

Two cards are

compatible

if the strings differ in

exactly

1 position.

Example 1:

Input:

cards = ["aa","ab","ba","ac"], x = "a"

Output:

2

Explanation:

On the first turn, select and remove cards

"ab"

and

"ac"

, which are compatible because they differ at only index 1.

On the second turn, select and remove cards

"aa"

and

"ba"

, which are compatible because they differ at only index 0.

Because there are no more compatible pairs, the total score is 2.

Example 2:

Input:

cards = ["aa","ab","ba"], x = "a"

Output:

1

Explanation:

On the first turn, select and remove cards

"aa"

and

"ba"

.

Because there are no more compatible pairs, the total score is 1.

Example 3:

Input:

cards = ["aa","ab","ba","ac"], x = "b"

Output:

0

Explanation:

The only cards that contain the character

'b'

are

"ab"

and

"ba"

. However, they differ in both indices, so they are not compatible. Thus, the output is 0.

Constraints:

2 <= cards.length <= 10

5

cards[i].length == 2

Each

cards[i]

is composed of only lowercase English letters between

'a'

and

'j'

.

x

is a lowercase English letter between

'a'

and

'j'

.

Code Snippets

C++:

```
class Solution {
public:
int score(vector<string>& cards, char x) {

}
};
```

Java:

```
class Solution {
public int score(String[] cards, char x) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} cards
* @param {character} x
* @return {number}
*/
var score = function(cards, x) {

};
```

TypeScript:

```
function score(cards: string[], x: string): number {

};
```

C#:

```
public class Solution {
public int Score(string[] cards, char x) {

}
}
```

C:

```
int score(char** cards, int cardsSize, char x) {

}
```

Go:

```
func score(cards []string, x byte) int {

}
```

Kotlin:

```
class Solution {
fun score(cards: Array<String>, x: Char): Int {

}
}
```

Swift:

```
class Solution {
func score(_ cards: [String], _ x: Character) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn score(cards: Vec<String>, x: char) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} cards
# @param {Character} x
# @return {Integer}
def score(cards, x)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String[] $cards
* @param String $x
* @return Integer
*/
function score($cards, $x) {

}
}
```

Dart:

```
class Solution {
int score(List<String> cards, String x) {

}
}
```

Scala:

```
object Solution {
def score(cards: Array[String], x: Char): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec score(cards :: [String.t], x :: char) :: integer
def score(cards, x) do

end
end
```

Erlang:

```
-spec score(Cards :: [unicode:unicode_binary()], X :: char()) -> integer().
score(Cards, X) ->
.
```

Racket:

```
(define/contract (score cards x)
(-> (listof string?) char? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 score(vector<string>& cards, char x) {

}
};
```

Java Solution:

```
/**
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 score(String[] cards, char x) {

}
}
```

Python3 Solution:

```
"""
Problem: Two-Letter Card Game
Difficulty: Medium
Tags: array, string, 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 score(self, cards: List[str], x: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 {string[]} cards
* @param {character} x
* @return {number}
*/
var score = function(cards, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 score(cards: string[], x: string): number {

};
```

C# Solution:

```
/*
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 Score(string[] cards, char x) {

}
}
```

C Solution:

```
/*
* Problem: Two-Letter Card Game
* Difficulty: Medium
* Tags: array, string, 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 score(char** cards, int cardsSize, char x) {

}
```

Go Solution:

```
// Problem: Two-Letter Card Game
// Difficulty: Medium
// Tags: array, string, 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 score(cards []string, x byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun score(cards: Array<String>, x: Char): Int {

}
}
```

Swift Solution:

```
class Solution {
func score(_ cards: [String], _ x: Character) -> Int {

}
}
```

Rust Solution:

```
// Problem: Two-Letter Card Game
// Difficulty: Medium
// Tags: array, string, 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 score(cards: Vec<String>, x: char) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} cards
# @param {Character} x
# @return {Integer}
def score(cards, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $cards
* @param String $x
* @return Integer
*/
function score($cards, $x) {

}
}
```

Dart Solution:

```
class Solution {
int score(List<String> cards, String x) {

}
}
```

Scala Solution:

```
object Solution {
def score(cards: Array[String], x: Char): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec score(cards :: [String.t], x :: char) :: integer
def score(cards, x) do

end
end
```

Erlang Solution:

```
-spec score(Cards :: [unicode:unicode_binary()], X :: char()) -> integer().
score(Cards, X) ->
.
```

Racket Solution:

```
(define/contract (score cards x)
(-> (listof string?) char? exact-integer?)
)
```