

Problem 781: Rabbits in Forest

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

There is a forest with an unknown number of rabbits. We asked n rabbits

"How many rabbits have the same color as you?"

and collected the answers in an integer array

answers

where

answers[i]

is the answer of the

i

th

rabbit.

Given the array

answers

, return

the minimum number of rabbits that could be in the forest

.

Example 1:

Input:

answers = [1,1,2]

Output:

5

Explanation:

The two rabbits that answered "1" could both be the same color, say red. The rabbit that answered "2" can't be red or the answers would be inconsistent. Say the rabbit that answered "2" was blue. Then there should be 2 other blue rabbits in the forest that didn't answer into the array. The smallest possible number of rabbits in the forest is therefore 5: 3 that answered plus 2 that didn't.

Example 2:

Input:

answers = [10,10,10]

Output:

11

Constraints:

$1 \leq \text{answers.length} \leq 1000$

$0 \leq \text{answers}[i] < 1000$

Code Snippets

C++:

```
class Solution {
public:
    int numRabbits(vector<int>& answers) {

    }
};
```

Java:

```
class Solution {
    public int numRabbits(int[] answers) {

    }
}
```

Python3:

```
class Solution:
    def numRabbits(self, answers: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numRabbits(answers: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int NumRabbits(int[] answers) {  
  
    }  
}
```

C:

```
int numRabbits(int* answers, int answersSize) {  
  
}
```

Go:

```
func numRabbits(answers []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun numRabbits(answers: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func numRabbits(_ answers: [Int]) -> Int {  
  
    }  
}
```

Rust:

```

impl Solution {
  pub fn num_rabbits(answers: Vec<i32>) -> i32 {

  }
}

```

Ruby:

```

# @param {Integer[]} answers
# @return {Integer}
def num_rabbits(answers)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[] $answers
     * @return Integer
     */
    function numRabbits($answers) {

    }

}

```

Dart:

```

class Solution {
  int numRabbits(List<int> answers) {

  }
}

```

Scala:

```

object Solution {
  def numRabbits(answers: Array[Int]): Int = {

  }
}

```

Elixir:

```
defmodule Solution do
  @spec num_rabbits(answers :: [integer]) :: integer
  def num_rabbits(answers) do

  end
end
```

Erlang:

```
-spec num_rabbits(Answers :: [integer()]) -> integer().
num_rabbits(Answers) ->
.
```

Racket:

```
(define/contract (num-rabbits answers)
  (-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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 numRabbits(vector<int>& answers) {

    }
};
```

Java Solution:

```
/**
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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 numRabbits(int[] answers) {

    }
}
```

Python3 Solution:

```
"""
Problem: Rabbits in Forest
Difficulty: Medium
Tags: array, greedy, math, 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 numRabbits(self, answers: List[int]) -> int:
        # TODO: Implement optimized solution
        pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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[]} answers
 * @return {number}
 */
var numRabbits = function(answers) {

};
```

TypeScript Solution:

```
/**
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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 numRabbits(answers: number[]): number {

};
```

C# Solution:

```

/*
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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 NumRabbits(int[] answers) {

    }
}

```

C Solution:

```

/*
 * Problem: Rabbits in Forest
 * Difficulty: Medium
 * Tags: array, greedy, math, 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 numRabbits(int* answers, int answersSize) {

}

```

Go Solution:

```

// Problem: Rabbits in Forest
// Difficulty: Medium
// Tags: array, greedy, math, 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 numRabbits(answers []int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun numRabbits(answers: IntArray): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func numRabbits(_ answers: [Int]) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Rabbits in Forest
// Difficulty: Medium
// Tags: array, greedy, math, 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 num_rabbits(answers: Vec<i32>) -> i32 {

    }
}

```

Ruby Solution:

```

# @param {Integer[]} answers
# @return {Integer}
def num_rabbits(answers)

```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $answers  
     * @return Integer  
     */  
    function numRabbits($answers) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    int numRabbits(List<int> answers) {  
  
    }  
}
```

Scala Solution:

```
object Solution {  
    def numRabbits(answers: Array[Int]): Int = {  
  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
    @spec num_rabbits(answers :: [integer]) :: integer  
    def num_rabbits(answers) do  
  
    end  
end
```

Erlang Solution:

```
-spec num_rabbits(Answers :: [integer()]) -> integer().  
num_rabbits(Answers) ->  
.
```

Racket Solution:

```
(define/contract (num-rabbits answers)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```