Problem 860: Lemonade Change

Problem Information

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

Problem Description

At a lemonade stand, each lemonade costs

$5

. Customers are standing in a queue to buy from you and order one at a time (in the order specified by bills). Each customer will only buy one lemonade and pay with either a

$5

,

$10

, or

$20

bill. You must provide the correct change to each customer so that the net transaction is that the customer pays

$5

.

Note that you do not have any change in hand at first.

Given an integer array

bills

where

bills[i]

is the bill the

i

th

customer pays, return

true

if you can provide every customer with the correct change, or

false

otherwise

.

Example 1:

Input:

bills = [5,5,5,10,20]

Output:

true

Explanation:

From the first 3 customers, we collect three $5 bills in order. From the fourth customer, we collect a $10 bill and give back a $5. From the fifth customer, we give a $10 bill and a $5 bill. Since all customers got correct change, we output true.

Example 2:

Input:

bills = [5,5,10,10,20]

Output:

false

Explanation:

From the first two customers in order, we collect two $5 bills. For the next two customers in order, we collect a $10 bill and give back a $5 bill. For the last customer, we can not give the change of $15 back because we only have two $10 bills. Since not every customer received the correct change, the answer is false.

Constraints:

1 <= bills.length <= 10

5

bills[i]

is either

5

,

10

, or

20

.

Code Snippets

C++:

```
class Solution {
public:
bool lemonadeChange(vector<int>& bills) {

}
};
```

Java:

```
class Solution {
public boolean lemonadeChange(int[] bills) {

}
}
```

Python3:

```
class Solution:
def lemonadeChange(self, bills: List[int]) -> bool:
```

Python:

```
class Solution(object):
def lemonadeChange(self, bills):
"""
:type bills: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} bills
```

```
* @return {boolean}
*/
var lemonadeChange = function(bills) {

};
```

TypeScript:

```
function lemonadeChange(bills: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool LemonadeChange(int[] bills) {

}
}
```

C:

```
bool lemonadeChange(int* bills, int billsSize) {

}
```

Go:

```
func lemonadeChange(bills []int) bool {

}
```

Kotlin:

```
class Solution {
fun lemonadeChange(bills: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func lemonadeChange(_ bills: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn lemonade_change(bills: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} bills
# @return {Boolean}
def lemonade_change(bills)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $bills
* @return Boolean
*/
function lemonadeChange($bills) {

}
}
```

Dart:

```
class Solution {
bool lemonadeChange(List<int> bills) {

}
}
```

Scala:

```
object Solution {
def lemonadeChange(bills: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec lemonade_change(bills :: [integer]) :: boolean
def lemonade_change(bills) do

end
end
```

Erlang:

```
-spec lemonade_change(Bills :: [integer()]) -> boolean().
lemonade_change(Bills) ->
.
```

Racket:

```
(define/contract (lemonade-change bills)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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:
bool lemonadeChange(vector<int>& bills) {

}
};
```

Java Solution:

```
/**
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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 boolean lemonadeChange(int[] bills) {

}
}
```

Python3 Solution:

```
"""
Problem: Lemonade Change
Difficulty: Easy
Tags: array, greedy, queue

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 lemonadeChange(self, bills: List[int]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def lemonadeChange(self, bills):
"""
:type bills: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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[]} bills
* @return {boolean}
*/
var lemonadeChange = function(bills) {

};
```

TypeScript Solution:

```
/**
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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 lemonadeChange(bills: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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 bool LemonadeChange(int[] bills) {

}
}
```

C Solution:

```
/*
* Problem: Lemonade Change
* Difficulty: Easy
* Tags: array, greedy, queue
*
* 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
*/

bool lemonadeChange(int* bills, int billsSize) {

}
```

Go Solution:

```
// Problem: Lemonade Change
// Difficulty: Easy
// Tags: array, greedy, queue
//
// 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 lemonadeChange(bills []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun lemonadeChange(bills: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func lemonadeChange(_ bills: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Lemonade Change
// Difficulty: Easy
// Tags: array, greedy, queue
//
// 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 lemonade_change(bills: Vec<i32>) -> bool {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} bills
# @return {Boolean}
def lemonade_change(bills)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $bills
* @return Boolean
*/
function lemonadeChange($bills) {

}
}
```

Dart Solution:

```
class Solution {
bool lemonadeChange(List<int> bills) {

}
}
```

Scala Solution:

```
object Solution {
def lemonadeChange(bills: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec lemonade_change(bills :: [integer]) :: boolean
def lemonade_change(bills) do

end
end
```

Erlang Solution:

```
-spec lemonade_change(Bills :: [integer()]) -> boolean().
lemonade_change(Bills) ->
.
```

Racket Solution:

```
(define/contract (lemonade-change bills)
(-> (listof exact-integer?) boolean?)
)
```