860. Lemonade Change

Easy Greedy Array

Leetcode Link

Problem Description

In this problem, you are operating a lemonade stand where each glass of lemonade is sold for `$5`. You start without any change in hand. Customers come in a queue, and you should sell them lemonade in the order they come, with each customer buying exactly one glass. Customers pay with a `$5`, `$10`, or `$20` bill. Your task is to determine whether you can provide the correct change to each customer using the bills you have at your disposal. You need to give back the net difference such that each customer effectively pays exactly `$5` for their lemonade. For instance, if a customer pays with `$20`, and you have enough change, you will hand back `$15` in change. If it is possible to provide the correct change for all customers, the function should return `true`, otherwise it should return `false`.

Intuition

The main issue is to always have enough `$5` bills to make change, since they are the cornerstone of all transactions. Here's the intuitive approach:

1. Keep track of the number of `$5` and `$10` bills (we don't need to track `$20` bills as they don't help in making change).
2. Process each customer in the queue one by one and handle the payment.
   - If a customer pays with a `$5` bill, no change is needed, and you just increase your count of `$5` bills.
   - If a customer pays with a `$10` bill, you need to give back a `$5` as change, so you decrement the `$5` bill count and increment the `$10` bill count.
   - If a customer pays with a `$20` bill, you prefer to give back one `$10` and one `$5` as change because this uses up larger bills and keeps more `$5` bills for future transactions (which are more crucial). If you can't give change in this combination, you attempt to give three `$5` bills as change.
3. After handling a transaction, if the count of `$5` bills is negative, that means you didn't have enough change for a customer, and hence you should return `false`.
4. If you finish processing all customers and never ran out of `$5` bills, then you return `true`.

Solution Approach

The solution is straightforward and follows a greedy algorithmic approach, which operates on the simplest of principles: give out the minimal amount of change necessary.

In terms of data structures, we only need two variables to keep track of the number of `$5` and `$10` bills as these are the only denominations we can give out for change.

The algorithm goes as follows:

1. Initialize two variables: `five` and `ten` to track the count of `$5` and `$10` bills we have. We do not need to keep track of `$20` bills as they cannot be given as change.
2. Iterate through each bill in the `bills` list.
   - If the bill is `$5`, increment the `five` counter, since we now have an additional `$5` bill.
   - If the bill is `$10`, we need to give a `$5` bill as change, so we decrement the `five` counter and increment the `ten` counter.
   - If the bill is `$20`, firstly, we check if we have any `$10` bills:
     - If we have a `$10` bill, we use it along with a `$5` bill to make `$15` in change (decrement both `ten` and `five`).
     - If we don't have a `$10` bill, we need to give out three `$5` bills (decrement `five` by three).
   - After handling the bill, we check if the `five` counter has gone negative. If it has, it means we can't make change for the current transaction, and we return `false`.
3. After the loop, if we've successfully given change to every customer, our `five` counter should never be negative, and we can return `true`.

The algorithm effectively balances between preserving `$5` bills, which are necessary for giving change to customers who pay with `$10` and `$20`, and using up `$10` bills when possible to avoid depleting the `$5` bills too quickly. This greedy strategy works for any sequence of bills because making change with larger bills when possible is always at least as good as, and often better than, breaking down `$5` bills.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach. Suppose we have a queue of customers with the following bills: 5,10, 20,5, $10.

1. Initialize `five` = 0 and `ten` = 0.
2. Customer 1 pays with $5:
   - `five` is incremented by 1 (`five` = 1, `ten` = 0).
3. Customer 2 pays with $10:
   - We give one $5 bill as change: `five` is decremented by 1, `ten` is incremented by 1 (`five` = 0, `ten` = 1).
4. Customer 3 pays with $20:
   - To give change, we'd prefer a $10 and a 5$ bill. We have one $10 bill:
     - `ten` is decremented by 1 and `five` is decremented by 1 to provide $15 in change (`five` = -1, `ten` = 0).
     - Since we don't have any $5 bills left to give as change, we cannot fulfill this transaction - we return `false`.

We stop the process as we've been unable to provide change for the third customer. Thus, the function will return `false` in this example.

Python Solution

```python
class Solution:
    def lemonadeChange(self, bills: List[int]) -> bool:
        # Initialize counters for five and ten dollar bills
        five_dollar_count = ten_dollar_count = 0

        # Iterate over each bill received
        for bill in bills:
            if bill == 5:
                # If it's a $5 bill, simply increase the count of $5 bills
                five_dollar_count += 1
            elif bill == 10:
                # If it's a $10 bill, give one $5 bill as change
                ten_dollar_count += 1
                five_dollar_count -= 1
            else:
                # If it's a $20 bill, try to give one $10 and one $5 as change if possible
                # Otherwise, give three $5 bills as change
                if ten_dollar_count:
                    ten_dollar_count -= 1
                    five_dollar_count -= 1
                else:
                    five_dollar_count -= 3

            # If at any point the count of $5 bills drops below zero, it's impossible to give change
            if five_dollar_count < 0:
                return False

        # If we got to the end without running out of $5 bills, we can give change for all transactions
        return True

```

Java Solution

```java
class Solution {
    public boolean lemonadeChange(int[] bills) {
        // Initialize counters for five and ten dollar bills
        int fiveDollarBills = 0;
        int tenDollarBills = 0;

        // Iterate over each bill in the array
        for (int bill : bills) {
            // Use switch-case to handle different bill values
            switch (bill) {
                case 5:
                    // If it's a $5 bill, no change is needed, increase count of $5 bills
                    fiveDollarBills++;
                    break;
                case 10:
                    // For a $10 bill, we need to give one $5 bill as change
                    tenDollarBills++;        // Increase $10 bills
                    fiveDollarBills--;       // Reduce $5 bills as we give it as change
                    break;
                case 20:
                    // For a $20 bill, prefer to give one $10 and one $5 as change if possible
                    if (tenDollarBills > 0) {
                        tenDollarBills--;     // Use a $10 bill for change
                        fiveDollarBills--;    // Use a $5 bill for change
                    } else {
                        // If no $10 bills, we need to give three $5 bills as change
                        fiveDollarBills -= 3;
                    }
                    break;
            }
            // If at any point we do not have enough $5 bills to give as change, return false
            if (fiveDollarBills < 0) {
                return false;
            }
        }
        // If we can make change for all customers, return true
        return true;
    }
}

```

C++ Solution

```cpp
#include <vector> // Include the vector header for using the vector container.

class Solution {
public:
    // Method to determine if we can provide every customer with correct change.
    bool lemonadeChange(vector<int>& bills) {
        // Initialize counters for $5 and $10 bills.
        int fiveDollarBills = 0, tenDollarBills = 0;

        // Iterate over each bill in the vector 'bills'.
        for (int bill : bills) {
            if (bill == 5) {
                // If the bill is $5, no change is needed, simply increment $5 bill counter.
                ++fiveDollarBills;
            } else if (bill == 10) {
                // If the bill is $10, give one $5 bill as change and increment $10 bill counter.
                ++tenDollarBills;
                --fiveDollarBills; // Giving change of one $5 bill.
            } else {
                // For a $20 bill, prefer to give one $10 and one $5 bill as change if possible.
                if (tenDollarBills > 0) {
                    --tenDollarBills;
                    --fiveDollarBills; // Giving change of one $10 and one $5 bill.
                } else {
                    // If no $10 bills are available, give three $5 bills as change.
                    fiveDollarBills -= 3;
                }
            }

            // If at any point we do not have enough $5 bills to give change, return false.
            if (fiveDollarBills < 0) {
                return false;
            }
        }
        // If we were able to provide change for all customers, return true.
        return true;
    }
};

```

Typescript Solution

```typescript
function lemonadeChange(bills: number[]): boolean {
    // Initialize the count of $5 and $10 bills
    let fiveDollarBills = 0;
    let tenDollarBills = 0;

    // Iterate through each bill received
    for (let bill of bills) {
        switch (bill) {
            case 5: // When the bill is $5, no change is needed, simply increase the count of $5 bills
                fiveDollarBills++;
                break;
            case 10: // For a $10 bill, give back one $5 bill as change and increase $10 bill count
                fiveDollarBills--;   // Giving change
                tenDollarBills++;    // Receiving $10
                break;
            case 20: // For a $20 bill, try to give one $10 and one $5 as change if possible
                if (tenDollarBills > 0) {
                    tenDollarBills -= 1;    // Giving one $10 bill as change
                    bill -= 10;             // $10 change has been given, need $10 more
                }
                fiveDollarBills -= bill / 5 - 1; // Give the rest of the change in $5 bills
                break;
            // Note: Though the use of the 'bill' variable here is a bit unconventional,
            //       since we've subtracted $10 if we've used a $10 bill for change,
            //       dividing 'bill' by 5 now effectively gives us how many more $5 bills
            //       we need to give as change (either one $5 bill if we gave a $10, or
            //       three $5 bills if we didn't).
        }

        // If we don't have enough $5 bills to give change, return false
        if (fiveDollarBills < 0) {
            return false;
        }
    }

    // If we've iterated through all bills and always had enough to give change, return true
    return true;
}

```

Time and Space Complexity

Time Complexity:

The given algorithm iterates through the list of bills once. During each iteration, it performs a constant number of operations such as comparison, increment, and decrement on integer variables. These operations do not depend on the size of the input and hence take constant time.

Therefore, the time complexity of the algorithm is determined by the number of iterations, which is directly proportional to the length of the list of bills, `n`. Hence, the time complexity is `O(n)`, where `n` is the length of the bills list.

Space Complexity:

The algorithm uses a fixed number of integer variables (`five` and `ten`) to keep track of the count of $5 and 10$ bills. The space used does not grow with the size of the input list. These two integer variables use a constant amount of space.

Consequently, the space complexity of the algorithm is `O(1)`, which means it uses constant additional space regardless of the input size.