1357. Apply Discount Every n Orders

Medium Design Array Hash Table

Leetcode Link

Problem Description

In this problem, we are asked to simulate the checkout process of a supermarket. There are two arrays, `products` and `prices`, representing the IDs and prices of all the products available in the supermarket, respectively. The checkout process involves a `Cashier` class that needs to be implemented with specific functionality. The class should handle customer bills where each bill is represented by two arrays, `product` and `amount`. The total bill or subtotal is calculated by multiplying the `amount` of each product by its `price`. Occasionally, a discount is applied to the subtotal if the customer is every `n`th customer. If a discount is applied, it is a percentage discount based on the `discount` value provided. The customer then only pays a fraction of the subtotal after the discount is taken into account. The `Cashier` class should be able to return the final amount the customer needs to pay, including any discounts.

Intuition

To solve the problem, the `Cashier` class is designed with an initialization method and a method to calculate the bill with a potential discount. During initialization, the `Cashier` object stores `n`, `discount`, and a dictionary mapping each product ID to its price. This setup is to efficiently look up the prices during bill computation.

The `getBill` method serves to compute the bill for each customer. A customer count `i` is maintained to track when to provide a discount (every `n`th customer). When calculating the subtotal, we look up the prices of each product in the provided dictionary. The bill is calculated by summing up the product of the amount and price of each item bought by the customer. If the customer is eligible for a discount (checked by seeing if `i` is divisible by `n`), the discount is applied to each item's total before adding it to the final bill. In the end, the method returns the final amount due, including any discount applied.

Solution Approach

The solution uses a simple object-oriented programming approach with a class `Cashier` that contains two methods: the constructor `__init__` and `getBill`.

Constructor - `__init__`:

This method is used to initialize the cashier object. It takes the following parameters:

- `n`: the frequency of the discount, indicating every `n`th customer receives a discount.
- `discount`: the percentage of the discount applied to every `n`th customer's bill.
- `products`: a list of integers representing the IDs of products.
- `prices`: a list of integers representing the prices corresponding to the products by their index.

The constructor achieves the following:

- It sets the instance variable `self.i` to `0`, to keep track of the number of customers processed.
- It stores `n` and `discount` in instance variables `self.n` and `self.discount` for use in the `getBill` method.
- It creates a dictionary `self.d` that maps each product ID to its price for quick price lookup. This is accomplished by using the `zip` function to combine `products` and `prices` and converting the zipped object into a dictionary.

Method - `getBill`:

The `getBill` is responsible for calculating the total bill for a customer. It takes two arguments:

- `product`: a list of integers representing the IDs of products in the customer's bill.
- `amount`: a list of integers representing the quantities of the respective products in the customer's bill.

The method performs the following operations:

1. Increment the customer counter `self.i`.
2. Check if the current customer should get a discount by checking if the customer count `self.i` modulo `n` (`self.i % self.n`) is `0`.
3. Initialize a variable `ans` to accumulate the bill total.
4. Iterate over the `product` and `amount` lists in parallel using `zip`.
5. For each item in the bill:
   - Get the product price from the dictionary `self.d`.
   - Calculate the price for the `amount` of the product.
   - If the discount applies, reduce the price accordingly by multiplying by `(100 - discount) / 100`.
6. Add the calculated item price to the total `ans`.
7. Return the total `ans`, which represents the final bill amount.

By maintaining a customer count and using a dictionary for price lookup, the algorithm achieves efficient processing of each bill. The discount application is straightforward and only done every `n`th bill to avoid unnecessary checks and computations.

Example Walkthrough

Let's illustrate the solution approach with an example. We're given the following input:

- `n = 3` (every 3rd customer gets a discount)
- `discount = 10` (10% discount for the eligible customer)
- `products = [101, 102, 103, 104]` (product IDs)
- `prices = [10, 15, 20, 25]` (corresponding prices for product IDs)

First, we initialize the Cashier:

```
1 cashier = Cashier(n=3, discount=10, products=[101, 102, 103, 104], prices=[10, 15, 20, 25])
```

The `Cashier` object will track the number of customers in `self.i`, which starts at 0. It will also store our discount information and create a dictionary for product prices as `self.d = {101: 10, 102: 15, 103: 20, 104: 25}`.

Now let's simulate three customers checking out with their bills:

Customer 1 Buys:

- `product = [101, 102]`
- `amount = [2, 1]`

```
1 print(cashier.getBill(product=[101, 102], amount=[2, 1]))
```

For customer 1, `self.i` increments to 1. Since `1 % 3` is not `0`, no discount is applied. The total is computed as `2 * 10 (for product 101) + 1 * 15 (for product 102)`, totaling `35`.

Customer 2 Buys:

- `product = [103]`
- `amount = [3]`

```
1 print(cashier.getBill(product=[103], amount=[3]))
```

For customer 2, `self.i` increments to 2. Since `2 % 3` is not `0`, no discount is applied. The total is `3 * 20 (for product 103)`, totaling `60`.

Customer 3 Buys (eligible for discount):

- `product = [101, 104]`
- `amount = [1, 2]`

```
1 print(cashier.getBill(product=[101, 104], amount=[1, 2]))
```

For customer 3, `self.i` increments to 3. Since `3 % 3` is `0`, this customer gets a discount. The total before discount is `1 * 10 (for product 101) + 2 * 25 (for product 104)`, which is `60`. A 10% discount is applied, making the total `60 - (0.10 * 60)` = `54`.

So, for these three customers, the `getBill` method would return `35`, `60`, and `54`, respectively. This demonstrates how the `Cashier` class processes customers' bills and applies discounts every `n`th customer.

Python Solution

```python
from typing import List

class Cashier:
    def __init__(self, n: int, discount: int, products: List[int], prices: List[int]):
        # Initialize the cashier to keep track of the number of customers
        self.customer_count = 0
        # Store the frequency of the discount application
        self.discount_frequency = n
        # Store the discount percentage
        self.discount_percentage = discount
        # Create a dictionary mapping products to their respective prices
        self.product_prices = {product: price for product, price in zip(products, prices)}

    def getBill(self, product: List[int], amount: List[int]) -> float:
        # Increment the count of customers
        self.customer_count += 1
        # Check if the current customer is eligible for a discount
        is_discounted = self.customer_count % self.discount_frequency == 0
        total_bill = 0.0  # Initialize total bill to 0
        for product_id, amount in zip(product, amount):
            # Calculate the total price for the product
            total_price = self.product_prices[product_id] * amount

            # If the customer is eligible for discount, apply it to the product price
            if is_discounted:
                total_price -= (self.discount_percentage * total_price) / 100

            # Add the total price for this product to the overall bill
            total_bill += total_price

        # Return the final bill amount
        return total_bill

# Example on how to instantiate and call the Cashier class and getBill method:
# cashier = Cashier(n, discount, products, prices)
# bill = cashier.getBill(product, amount)
```

Java Solution

```java
import java.util.HashMap;
import java.util.Map;

class Cashier {
    private int customerCount;      // Counts the number of customers served
    private int discountFrequency;  // The frequency of which a discount is given
    private int discountPercentage; // The discount percentage
    private Map<Integer, Integer> productPrices = new HashMap<>(); // Stores the product ID and price

    /*
     * Constructor for the Cashier class.
     * @param n The number of customers after which the discount is applied.
     * @param discount The percentage of discount offered.
     * @param products A list of product IDs.
     * @param prices The corresponding prices of the products.
     */
    public Cashier(int n, int discount, int[] products, int[] prices) {
        this.n = n;
        this.discountPercentage = discount;
        // Initialize the productPrices map with the product IDs and their prices.
        for (int i = 0; i < products.length; ++i) {
            productPrices.put(products[i], prices[i]);
        }
    }

    /*
     * Calculates the bill for the given customer.
     * @param product An array of product IDs that the customer is buying.
     * @param amount The corresponding quantities of each product.
     * @return The total bill for the customer considering possible discounts.
     */
    public double getBill(int[] product, int[] amount) {
        customerCount++; // Increment the number of customers served
        // Check if the current customer should receive a discount
        int discountApplicable = customerCount % n == 0 ? discountPercentage : 0;
        double total = 0.0; // Total bill initialized to 0

        // Calculate the bill
        for (int i = 0; i < product.length; i++) {
            int productId = product[i];
            int productQuantity = amount[i];
            double productPrice = productPrices.get(productId);
            double cost = productPrice * productQuantity;
            double discountedCost = cost - (discountApplicable * cost) / 100.0;
            total += discountedCost; // Add the cost after discount to the total.
        }
        return total; // Return the total bill
    }
}
```

C++ Solution

```cpp
#include <vector>
#include <unordered_map>
using namespace std;

class Cashier {
public:
    // Constructor with initialization of the cashier system.
    Cashier(int n, int discount, vector<int>& products, vector<int>& prices) {
        customerCount = 0; // Initialize the customer count to zero.
        this->checkoutFrequency = n; // Set the nth customer checkout frequency.
        this->discountPercentage = discount; // Set the discount percentage.

        // Store the prices for each product.
        for (int i = 0; i < products.size(); ++i) {
            productPrices[products[i]] = prices[i];
        }
    }

    // Method to calculate the bill for the current customer.
    double getBill(vector<int>& productIds, vector<int>& productAmounts) {
        customerCount++; // Increment customer count for each checkout.
        // Check if the current customer is eligible for discount.
        int isDiscountEligible = (customerCount % checkoutFrequency == 0) ? discountPercentage : 0;

        double totalCost = 0; // Initialize total cost of the bill.
        // Calculate total cost of the current bill.
        for (int i = 0; i < productIds.size(); ++i) {
            int productPrice = productPrices[productIds[i]];
            totalCost += productPrice * productAmounts[i];
        }

        // Apply discount if the customer is eligible.
        if (isDiscountEligible > 0) {
            double discountAmount = (totalCost * isDiscountEligible) / 100.0;
            totalCost -= discountAmount;
        }

        return totalCost; // Return the final bill amount.
    }

private:
    int customerCount; // Counter to track the number of customers served.
    int checkoutFrequency; // The frequency at which a discount is given.
    int discountPercentage; // The percentage of the discount applied.
    unordered_map<int, int> productPrices; // Mapping from product ID to its price.
};
```

Typescript Solution

```typescript
// Define the type for product pricing map.
type ProductPriceMap = { [productId: number]: number };

// Variables simulating the properties of the Cashier class
let customerCount = 0; // Counter to track the number of customers served.
let checkoutFrequency: number; // The frequency at which a discount is given.
let discountPercentage: number; // The percentage of the discount applied.
let productPrices: ProductPriceMap = {}; // Mapping from product ID to its price.

// Function used for initializing the cashier system.
function initialize(n: number, discount: number, products: number[], prices: number[]): void {
    customerCount = 0; // Resetting the customer count to zero.
    checkoutFrequency = n; // Setting the nth customer checkout frequency.
    discountPercentage = discount; // Setting the discount percentage.

    // Storing the prices for each product.
    for (let i = 0; i < products.length; i++) {
        productPrices[products[i]] = prices[i];
    }
}

// Function to calculate the bill for the current customer.
function getBill(productIds: number[], productAmounts: number[]): number {
    customerCount++; // Incrementing the customer count for each checkout.
    // Check if the current customer is eligible for a discount.
    const isDiscountEligible = customerCount % checkoutFrequency === 0 ? discountPercentage : 0;

    let totalCost = 0; // Initialize the total cost of the bill.

    // Calculate the total cost of the current bill.
    for (let i = 0; i < productIds.length; i++) {
        const productPrice = productPrices[productIds[i]];
        totalCost += productPrice * productAmounts[i];
    }

    // Apply a discount if the customer is eligible.
    if (isDiscountEligible > 0) {
        const discountAmount = (totalCost * isDiscountEligible) / 100.0;
        totalCost -= discountAmount;
    }

    return totalCost; // Return the final bill amount.
}
```

Time and Space Complexity

Time Complexity:

The time complexity of the `__init__` method is `O(n)` where `n` is the number of `products` because we loop through each product-price pair to create a dictionary, which is a one-time set-up operation.

The `getBill` method has a time complexity of `O(k)` where `k` is the number of items in the `product` list corresponding to a customer's purchase, as we loop through each product-amount pair to calculate the cost.

Overall, when considering the getBill method, which will be called multiple times, the time complexity is dominated by `O(k)`.

Space Complexity:

The space complexity of the `__init__` method is also `O(n)` due to the dictionary storing product-price pairs.

The `getBill` method uses a constant amount of space, hence the space complexity is `O(1)`.

Therefore, the overall space complexity, taking into account the dictionary created in the initializer, is `O(n)`.