Problem:- Grocery Billing System

Scenario:

You are designing a billing system for a grocery store. The system needs to calculate the total bill for customers based on the items they purchase. Each item has a fixed price, and customers can buy multiple quantities of the same item. Discounts can be applied to certain items, and the program should account for these discounts when calculating the total bill.

Explanation:

You need to write a program that calculates the final bill for a customer. The program will take the prices of items, the quantities purchased by the customer, and the discounts (if any) on specific items. The final bill is the sum of the discounted prices for all purchased items.

Input Format:

- The first line contains an integer n, the number of items in the store.
- The second line contains n integers representing the prices of the items.
- The third line contains an integer m, the number of items purchased by the customer.
- The next m lines each contain three integers: index, quantity, and discount, where:
    - `index` is the 0-based index of the item purchased.
    - `quantity` is the number of units purchased.
    - `discount` is the percentage discount on the item (0 ≤ discount ≤ 100).

Output Format:

Output a single integer representing the total bill for the customer.

Constraints:

- 1 ≤ n ≤ 1000

- 1 ≤ prices ≤ 10
- 3
- 3
-
- 1 ≤ m ≤ 1000
- 0 ≤ index < n
- 1 ≤ quantity ≤ 100
- 0 ≤ discount ≤ 100

Sample Input:
```
5
100 200 300 400 500
3
0 2 10
3 1 20
4 4 0
```

Sample Output:
```
2500
```

Explanation of Output:

- The store has items priced at 100, 200, 300, 400, and 500.
- The customer purchases 2 units of item 0 with a 10% discount.
  [(100 - 10% of 100) × 2 = 90 × 2 = 180]
- The customer purchases 1 unit of item 3 with a 20% discount.
  [(400 - 20% of 400) × 1 = 320 × 1 = 320]
- The customer purchases 4 units of item 4 with no discount.
  [500 × 4 = 2000]
- The total bill is:
  [180 + 320 + 2000 = 2500]

Solution :

```
def calculate_total_bill():

    n = int(input())

    prices = list(map(int, input().split()))
    m = int(input())

    total_bill = 0

    for _ in range(m):
        index, quantity, discount = map(int, input().split())
        discounted_price = prices[index] * (1 - discount / 100)
        total_bill += discounted_price * quantity

    print(int(total_bill))

calculate_total_bill()
```

Problem 2:- Add walls

Background:
In a bustling city, you are tasked with building a series of connected tunnels represented by a doubly linked list of integers. Each integer represents a section's length, but to prevent collapse, you must strategically add walls (represented by 0) whenever the combined length of consecutive sections exceeds a certain threshold.
- First, build a doubly linked list from the given input and then proceed to solve the problem.

Objective:
Given a doubly linked list of N integers representing tunnel sections and a threshold K, your objective is to iterate through the list, maintaining a cumulative sum of section

lengths. Whenever the sum exceeds K, insert a wall (0) before the current section and continue with the remaining sections with cumulative sum now starting from after the wall.

Input Format:
The input to your program consists of the tunnel section lengths:
- The first line contains an integer N, the number of sections.
- The second line contains N integers, where each integer is the length of a section.
- The third line contains an integer K, the threshold.

Output Format:
Print the final linked list after adding walls between sections.

Constraints:
- 1 ≤ N ≤ 10^5 (number of sections)
- 1 ≤ K ≤ 500 (threshold)

Sample Input 1:
5
1 2 3 4 5
4

Sample Output 1:
1 2 0 3 0 4 0 5

Explanation 1:
First start iterating through the sections. The cumulative sum is calculated from the previous wall.
- The cumulative sum of section lengths is 1 + 2 + 3 = 6 on the third section. So, add a wall behind 3.
- The cumulative sum of section lengths is 3 + 4 on the fourth section. So, add a wall behind 4.
- The cumulative sum of section lengths is 4 + 5 on the fourth section. So, add a wall behind 5.

Sample Input 2:
10
1 1 1 1 1 1 1 1 1 1
4

Sample Output 2:
1 1 1 1 0 1 1 1 1 0 1 1

Explanation 2:
- Iterate through the doubly linked list and every time the cumulative sum exceeds 4, build a wall behind the current node. The cumulative sum is calculated from the previous wall.

Sample Input 3:
5
6 1 1 3 2
5

Sample Output 3:
0 6 0 1 1 3 0 2

Explanation 3:
- Start iterating through the doubly linked list. The cumulative sum is calculated from the previous wall.
- The cumulative sum of section lengths is 6 on the first section. So, add a wall behind 6.
- The cumulative sum of section lengths is 6 + 1 on the second section. So, add a wall behind 1.
- The cumulative sum of section lengths is 1 + 1 + 3 = 5 on the fourth section. Since it does not exceed the threshold, no need to add a wall behind 3.
- The cumulative sum of section lengths is 1 + 1 + 3 + 2 = 7 on the fourth section. So, add a wall behind 2.

Solution:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

```
class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node
        new_node.prev = last

    def insert_before(self, next_node, data):
        if not next_node:
            return
        new_node = Node(data)
        new_node.prev = next_node.prev
        new_node.next = next_node
        if next_node.prev:
            next_node.prev.next = new_node
        else:
            self.head = new_node
        next_node.prev = new_node

    def process_list(self, K):
        current = self.head
        current_sum = 0
        while current:
            current_sum += current.data
            if current_sum > K:
                self.insert_before(current, 0)
                current_sum = current.data
            current = current.next

    def print_list(self):
        current = self.head
        while current:
            print(current.data, end=" ")
            current = current.next
        print()
```

```
N = int(input())
numbers = list(map(int, input().split()))
K = int(input())

# Building the doubly linked list
dll = DoublyLinkedList()
for num in numbers:
    dll.append(num)


dll.process_list(K)
dll.print_list()
```