题目描述

某网上商场举办优惠活动，发布了满减、打折、无门槛3种优惠券，分别为：

- 每满100元优惠10元，无使用数限制，如100~199元可以使用1张减10元，200~299可使用2张减20元，以此类推；
- 92折券，1次限使用1张，如100元，则优惠后为92元；
- 无门槛5元优惠券，无使用数限制，直接减5元。

优惠券使用限制

- 每次最多使用2种优惠券，2种优惠可以叠加（优惠叠加时以优惠后的价格计算），以购物200元为例，可以先用92折券优惠到184元，再用1张满减券优惠10元，最终价格是174元，也可以用满减券2张优惠20元为180元，再使用92折券优惠到165（165.6向下取整），不同使用顺序的优惠价格不同，以最优惠价格为准。在一次购物种，同一类型优惠券使用多张时必须一次性使用，不能分多次拆开使用（不允许先使用1张满减券，再用打折券，再使用一张满减券）。

问题

- 请设计实现一种解决方法，帮助购物者以最少的优惠券获得最优的优惠价格。优惠后价格越低越好，同等优惠价格，使用的优惠券越少越好，可以允许某次购物不使用优惠券。

约定

- 优惠活动每人只能参加一次，每个人的优惠券种类和数量是一样的。

输入描述

- 第一行：每个人拥有的优惠券数量（数量取值范围为[0,10]），按满减、打折、无门槛的顺序输入
- 第二行：表示购物的人数n（$1 \le n \le 10000$）
- 最后n行：每一行表示某个人优惠前的购物总价格（价格取值范围(0, 1000]，都为整数）。
- 约定：输入都是符合题目设定的要求的。

输出描述

- 每行输出每个人每次购物优惠后的最低价格以及使用的优惠券总数量
- 每行的输出顺序和输入的顺序保持一致

备注

1. 优惠券数量都为整数，取值范围为[0, 10]
2. 购物人数为整数，取值范围为[1, 10000]
3. 优惠券的购物总价为整数，取值范围为 (0, 1000]
4. 优惠后价格如果是小数，则向下取整，输出都为整数。

用例

输入	3 2 5 3 100 200 400
输出	65 6 155 7 338 4
	输入： • 第一行：3种优惠券数量分别为：满减券3张，打折券2张，无门槛5张 • 第二行：总共3个人购物 • 第三行：第一个人购物优惠前价格为100元

用例

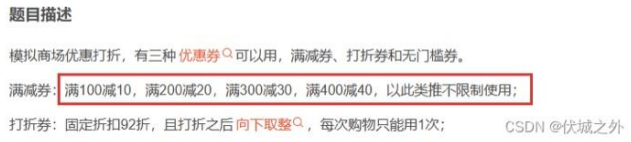
输入	3 2 5 3 100 200 400
输出	65 6 155 7 338 4
说明	输入： • 第一行：3种优惠券数量分别为：满减券3张，打折券2张，无门槛5张 • 第二行：总共3个人购物 • 第三行：第一个人购物优惠前价格为100元 • 第四行：第二个人购物优惠前价格为200元 • 第五行：第三个人购物优惠前价格为400元 输入3个人，输出3行结果，同输入的顺序，对应每个人的优惠结果，如下： • 第一行输出：先使用1张满减券优惠到90元，再使用5张无门槛券优惠到25元，最终价格是65元，总共使用6张优惠券。 • 第二行输出：先使用2张满减券优惠到180元，再使用5张无门槛券优惠到25元，最终价格是155元，总共使用7张优惠券。 • 第三行输出：先使用1张92折券优惠到368元，再使用3张满减券优惠到30元，最终价格是338元，总共使用4张优惠券。

题目解析

本题和华为OD机试 - 模拟商场优惠打折_伏城之外的博客-CSDN博客

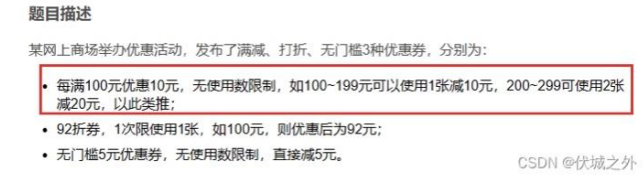
非常类似，但是关于满减券的使用逻辑不同，导致最终实现也不同。本题和上面链接题目应该属于AB卷题目，防止作弊的。

模拟商场优惠打折，这题关于满减券的使用逻辑：



即只要符合满减要求，就可以满减，直到满减券用完，比如其用例中

而本题中，满减券使用是有限制的



即：满100，最多使用1张减10元的券，满200最多使用2张减10元的券，满300可以使用3张减10元的券....

因此，我们只要基于华为OD机试 - 模拟商场优惠打折_伏城之外的博客-CSDN博客

中满减的逻辑微调即可。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let m, n, k, x;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [m, n, k] = lines[0].split(" ").map(Number);
  }
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let m, n, k, x;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [m, n, k] = lines[0].split(" ").map(Number);
  }

  if (lines.length === 2) {
    x = parseInt(lines[1]);
  }

  if (x && lines.length === x + 2) {
    lines.shift();
    lines.shift();

    const arr = lines.map(Number);

    getResult(arr, m, n, k);

    lines.length = 0;
  }
});

/**
 *
 * @param {*} prices 几个人打折之前的商品总价
 * @param {*} m 满减券数量
 * @param {*} n 打折券数量
 * @param {*} k 无门槛券数量
 */
function getResult(prices, m, n, k) {
  for (let price of prices) {
    const ans = [];

    const resM = M(price, m); // 先满减

    const resMN_N = N(resM[0], n); // 满减后打折
    ans.push([resMN_N[0], m + n - (resM[1] + resMN_N[1])]); // resMN_N[0]是“满减后打折”的剩余总价，  m + n -

    const resMK_K = K(resM[0], k); // 满减后无门槛
    ans.push([resMK_K[0], m + k - (resM[1] + resMK_K[1])]);

    const resN = N(price, n); // 先打折

    const resNM_M = M(resN[0], m); // 打折后满减
    ans.push([resNM_M[0], n + m - (resN[1] + resNM_M[1])]);

    const resNK_K = K(resN[0], k); // 打折后无门槛
    ans.push([resNK_K[0], n + k - (resN[1] + resNK_K[1])]);

    ans.sort((a, b) => (a[0] === b[0] ? a[1] - b[1] : a[0] - b[0])); // 对ans进行排序，排序规则是：优先按剩余总价

    console.log(ans[0].join(" "));
  }
}

/**
 * @param {*} price 总价
 * @param {*} m 满减券数量
 * @returns 总价满减后结果，对应数组含义是 [用券后剩余总价， 剩余满减券数量]
 */
function M(price, m) {
  const maxCount = price / 100; // 满100最多用1张满减券，满200最多用2张满减券.....，price总价最多使用price/100张券
  const count = Math.min(m, maxCount); // 实际可使用的满减券数量

  price -= count * 10;
  m -= count;

  return [price, m];
}

/**
 * @param {*} price 总价
 * @param {*} n 打折券数量
 * @returns 总价打折后结果，对应数组含义是 [用券后剩余总价， 剩余打折券数量]
 */
function N(price, n) {
  if (n >= 1) {
    price = Math.floor(price * 0.92);
  }
  return [price, n - 1];
}
```

```
    const resNK_K = K(resN[0], k); // 打折后无门槛
    ans.push([resNK_K[0], n + k - (resN[1] + resNK_K[1])]);

    ans.sort((a, b) => (a[0] === b[0] ? a[1] - b[1] : a[0] - b[0])); // 对ans进行排序，排序规则是：优先按剩余总价

    console.log(ans[0].join(" "));
  }
}

/**
 * @param {*} price 总价
 * @param {*} m 满减券数量
 * @returns 总价满减后结果，对应数组含义是 [用券后剩余总价， 剩余满减券数量]
 */
function M(price, m) {
  const maxCount = price / 100; // 满100最多用1张满减券，满200最多用2张满减券....，price总价最多使用price/100张券
  const count = Math.min(m, maxCount); // 实际可使用的满减券数量

  price -= count * 10;
  m -= count;

  return [price, m];
}

/**
 * @param {*} price 总价
 * @param {*} n 打折券数量
 * @returns 总价打折后结果，对应数组含义是 [用券后剩余总价， 剩余打折券数量]
 */
function N(price, n) {
  if (n >= 1) {
    price = Math.floor(price * 0.92);
  }
  return [price, n - 1];
}

/**
 * @param {*} price 总价
 * @param {*} k 无门槛券数量
 * @returns 无门槛券用后结果，对应数组含义是 [用券后剩余总价， 剩余无门槛券数量]
 */
function K(price, k) {
  while (price > 0 && k > 0) {
    price -= 5;
    price = Math.max(price, 0); // 无门槛券过多会导致优惠后总价小于0，此时我们应该避免
    k--;
  }
  return [price, k];
}
```

Java算法源码

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();
    int n = sc.nextInt();
    int k = sc.nextInt();
    int x = sc.nextInt();

    int[] arr = new int[x];
    for (int i = 0; i < x; i++) {
      arr[i] = sc.nextInt();
    }

    getResult(arr, m, n, k);
  }

  public static void getResult(int[] arr, int m, int n, int k) {
    for (int i = 0; i < arr.length; i++) {
      Integer[][] ans = new Integer[4][2]; // 4的含义对应4种使用券的方式：MN,NM,MK,NK，  2的含义对应每种方式下：剩余
      int price = arr[i];

      int[] resM = M(price, m); // 先满减
      int[] resN = N(price, n); // 先打折

      // MN
      int[] resMN_N = N(resM[0], n); // 满减后打折
      ans[0] =
          new Integer[] {
            resMN_N[0], m + n - resM[1] - resMN_N[1]
          }; // resMN_N[0]是“满减后打折”的剩余总价，  m + n - resM[1] - resMN_N[1] 是 该种用券方式的：总券数 m+n,
      // resM[1] + resMN_N[1]，因此使用掉的券数：  m+n - (resM[1] + resMN_N[1])

      // NM
      int[] resNM_M = M(resN[0], m); // 打折后满减
      ans[1] = new Integer[] {resNM_M[0], n + m - resN[1] - resNM_M[1]};

      // MK
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();
    int n = sc.nextInt();
    int k = sc.nextInt();
    int x = sc.nextInt();

    int[] arr = new int[x];
    for (int i = 0; i < x; i++) {
      arr[i] = sc.nextInt();
    }

    getResult(arr, m, n, k);
  }

  public static void getResult(int[] arr, int m, int n, int k) {
    for (int i = 0; i < arr.length; i++) {
      Integer[][] ans = new Integer[4][2]; // 4的含义对应4种使用券的方式：MN,NM,MK,NK，  2的含义对应每种方式下：剩余总
      int price = arr[i];

      int[] resM = M(price, m); // 先满减
      int[] resN = N(price, n); // 先打折

      // MN
      int[] resMN_N = N(resM[0], n); // 满减后打折
      ans[0] =
          new Integer[] {
            resMN_N[0], m + n - resM[1] - resMN_N[1]
          }; // resMN_N[0]是“满减后打折”的剩余总价，  m + n - resM[1] - resMN_N[1] 是 该种用券方式的：总券数 m+n，
      // resM[1] + resMN_N[1]，因此使用掉的券数：  m+n - (resM[1] + resMN_N[1])

      // NM
      int[] resNM_M = M(resN[0], m); // 打折后满减
      ans[1] = new Integer[] {resNM_M[0], n + m - resN[1] - resNM_M[1]};

      // MK
      int[] resMK_K = K(resM[0], k); // 满减后无门槛
      ans[2] = new Integer[] {resMK_K[0], m + k - resM[1] - resMK_K[1]};

      // NK
      int[] resNK_K = K(resN[0], k); // 打折后无门槛
      ans[3] = new Integer[] {resNK_K[0], n + k - resN[1] - resNK_K[1]};

      Arrays.sort(
          ans,
          (a, b) ->
              a[0].equals(b[0])
                  ? a[1] - b[1]
                  : a[0] - b[0]); // 对ans进行排序，排序规则是：优先按剩余总价升序，如果剩余总价相同，则再按“使用掉的券数
      System.out.println(ans[0][0] + " " + ans[0][1]);
    }
  }

  /**
   * @param price 总价
   * @param m 满减券数量
   * @return 总价满减后结果，对应数组含义是 [用券后剩余总价，  剩余满减券数量]
   */
  public static int[] M(int price, int m) {
    int maxCount = price / 100; // 满100最多用1张满减券，满200最多用2张满减券.....，price总价最多使用price/100张券
    int count = Math.min(maxCount, m); // 实际可使用的满减券数量

    price -= count * 10; // 每张满减券只能减10元
    m -= count;

    return new int[] {price, m};
  }

  /**
   * @param price 总价
   * @param n 打折券数量
   * @return 总价打折后结果，对应数组含义是 [用券后剩余总价，  剩余打折券数量]
   */
  public static int[] N(int price, int n) {
    if (n >= 1) {
      price = (int) Math.floor((price * 0.92));
    }
    return new int[] {price, n - 1};
  }

  /**
   * @param price 总价
   * @param k 无门槛券数量
   * @return 无门槛券用后结果，对应数组含义是 [用券后剩余总价，  剩余无门槛券数量]
   */
  public static int[] K(int price, int k) {
```

```
            ans,
            (a, b) ->
                a[0].equals(b[0])
                    ? a[1] - b[1]
                    : a[0] - b[0]); // 对ans进行排序，排序规则是：优先按剩余总价升序，如果剩余总价相同，则再按"使用掉的券数
        System.out.println(ans[0][0] + " " + ans[0][1]);
    }
  }

  /**
   * @param price 总价
   * @param m 满减券数量
   * @return 总价满减后结果，对应数组含义是 [用券后剩余总价， 剩余满减券数量]
   */
  public static int[] M(int price, int m) {
    int maxCount = price / 100; // 满100最多用1张满减券，满200最多用2张满减券...., price总价最多使用price/100张券
    int count = Math.min(maxCount, m); // 实际可使用的满减券数量

    price -= count * 10; // 每张满减券只能减10元
    m -= count;

    return new int[] {price, m};
  }

  /**
   * @param price 总价
   * @param n 打折券数量
   * @return 总价打折后结果，对应数组含义是 [用券后剩余总价， 剩余打折券数量]
   */
  public static int[] N(int price, int n) {
    if (n >= 1) {
      price = (int) Math.floor((price * 0.92));
    }
    return new int[] {price, n - 1};
  }

  /**
   * @param price 总价
   * @param k 无门槛券数量
   * @return 无门槛券用后结果，对应数组含义是 [用券后剩余总价， 剩余无门槛券数量]
   */
  public static int[] K(int price, int k) {
    while (price > 0 && k > 0) {
      price -= 5;
      price = Math.max(price, 0); // 感谢m0_71826536提供的思路，当无门槛券过多时，是有可能导致优惠后总价低于0的情况的，
      k--;
    }
    return new int[] {price, k};
  }
}
```

Python算法源码

```
m, n, k = map(int, input().split())

x = int(input())

prices = []
for i in range(x):
    prices.append(int(input()))


def fullSubtraction(price, m):
    """
    满减规则
    :param price: 总价
    :param m: 满减券数量
    :return: 总价满减后结果，对应数组含义是 (用券后剩余总价， 剩余满减券数量)
    """
    maxCount = int(price / 100)  # 满100最多用1张满减券，满200最多用2张满减券...., price总价最多使用price/100张券
    count = min(m, maxCount)  # 实际可使用的满减券数量

    price -= count * 10
    m -= count

    return price, m


def discount(price, n):
    """
    打折规则
    :param price: 总价
    :param n: 打折券数量
    :return: 总价打折后结果，对应数组含义是 (用券后剩余总价， 剩余打折券数量)
    """
    if n >= 1:
        price = int(price * 0.92)
    return price, n - 1


def thresholdFree(price, k):
    """
    无门槛你规则
```

Python算法源码

复制

```python
m, n, k = map(int, input().split())

x = int(input())

prices = []
for i in range(x):
    prices.append(int(input()))


def fullSubtraction(price, m):
    """
    满减规则
    :param price: 总价
    :param m: 满减券数量
    :return: 总价满减后结果，对应数组含义是（用券后剩余总价， 剩余满减券数量）
    """
    maxCount = int(price / 100)  # 满100最多用1张满减券，满200最多用2张满减券....，price总价最多使用price/100张券
    count = min(m, maxCount)  # 实际可使用的满减券数量

    price -= count * 10
    m -= count

    return price, m


def discount(price, n):
    """
    打折规则
    :param price: 总价
    :param n: 打折券数量
    :return: 总价打折后结果，对应数组含义是（用券后剩余总价， 剩余打折券数量）
    """
    if n >= 1:
        price = int(price * 0.92)
    return price, n - 1


def thresholdFree(price, k):
    """
    无门槛你规则
    :param price: 总价
    :param k: 无门槛券数量
    :return: 门槛券用后结果，对应数组含义是（用券后剩余总价， 剩余无门槛券数量）
    """
    while price > 0 and k > 0:
        price -= 5
        price = max(price, 0)  # 无门槛券过多会导致优惠后总价小于0，此时我们应该避免
        k -= 1
    return price, k


for price in prices:
    ans = []

    resM = fullSubtraction(price, m)  # 先满减

    resMN_N = discount(resM[0], n)  # 满减后打折
    ans.append((resMN_N[0], m + n - (resM[1] + resMN_N[1])))  # m + n 是满减后打折方式的总券数量， resM[1] + res

    resMK_K = thresholdFree(resM[0], k)  # 满减后无门槛
    ans.append((resMK_K[0], m + k - (resM[1] + resMK_K[1])))

    resN = discount(price, n)  # 先打折

    resNM_M = fullSubtraction(resN[0], m)  # 打折后满减
    ans.append((resNM_M[0], n + m - (resN[1] + resNM_M[1])))

    resNK_K = thresholdFree(resN[0], k)  # 打折后无门槛
    ans.append((resNK_K[0], n + k - (resN[1] + resNK_K[1])))

    # 对ans进行排序，排序规则是：优先按剩余总价升序，如果剩余总价相同，则再按"使用掉的券数量"升序
    ans.sort(key=lambda x: (x[0], x[1]))

    print(" ".join(map(str, ans[0])))
```