题目描述

模拟商场优惠打折，有三种 优惠券 可以用，满减券、打折券和无门槛券。

满减券：满100减10，满200减20，满300减30，满400减40，以此类推不限制使用；

打折券：固定折扣92折，且打折之后 向下取整 ，每次购物只能用1次；

无门槛券：一张券减5元，没有使用限制。

每个人结账使用优惠券时有以下限制：

每人每次只能用两种优惠券，并且同一种优惠券必须一次用完，不能跟别的穿插使用（比如用一张满减，再用一张打折，再用一张满减，这种顺序不行）。

求不同使用顺序下每个人用完券之后得到的最低价格和对应使用优惠券的总数；如果两种顺序得到的价格一样低，就取使用优惠券数量较少的那个。

输入描述

第一行三个数字m,n,k，分别表示每个人可以使用的满减券、打折券和无门槛券的数量;

第二行一个数字x, 表示有几个人购物;

后面x行数字，依次表示是这几个人打折之前的商品总价。

输出描述

输出每个人使用券之后的最低价格和对应使用优惠券的数量

用例

输入	3 2 5 3 100 200 400
输出	65 6 135 8 275 8
说明	输入： 第一行三个数字m,n,k，分别表示每个人可以使用的满减券、打折券和无门槛券的数量。 输出： 第一个人使用 1 张满减券和5张无门槛券价格最低。（100-10=90, 90-5*5=65） 第二个人使用 3 张满减券和5张无门槛券价格最低。（200-20-10-10=160, 160 – 5*5 = 135） 第二个人使用 3 张满减券和5张无门槛券价格最低。（400-40-30-30=300, 300 – 5*5=275）

题目解析

本题的解题思路如下，首先实现满减，打折，无门槛的逻辑：

- 满减逻辑，只要总价price大于等于100，且还有满减券，则不停price -= Math.floor(price / 100) * 10; 直到总价price小于100，或者满减券用完。
- 打折逻辑，按照题目意思，打折券只能使用一次，因此无论打折券有多少张，都只能使用一次，因此只要打折券数量大于等于1，那么price = Math.floor(price * 0.92);
- 无门槛逻辑，只要总价price大于0，且还有无门槛券，则不停price -= 5; 直到price小于等于0，或者无门槛券用完。

接下来就是求上面三种逻辑的任选2个的排列：

假设满减是M，打折是N，无门槛是K，则有排列如下：

- MN、NM
- MK、KM

题目解析

本题的解题思路如下，首先实现满减，打折，无门槛的逻辑：

- 满减逻辑，只要总价price大于等于100，且还有满减券，则不停price -= Math.floor(price / 100) * 10; 直到总价price小于100，或者满减券用完。
- 打折逻辑，按照题目意思，打折券只能使用一次，因此无论打折券有多少张，都只能使用一次，因此只要打折券数量大于等于1，那么price = Math.floor(price * 0.92);
- 无门槛逻辑，只要总价price大于0，且还有无门槛券，则不停price -= 5; 直到price小于等于0，或者无门槛券用完。

接下来就是求上面三种逻辑的任选2个的排列：

假设满减是M，打折是N，无门槛是K，则有排列如下：

- MN、NM
- MK、KM
- NK、KN

注意，券的使用对顺序敏感。

因此，求出以上排列后，对每个人的总价使用六种方式减价，只保留减价最多，用券最少的那个。

根据网友iygvh提供的优化思路：

对于无门槛券的使用，无门槛券总是在最后使用才会最优。

对于满减来说，无门槛肯定是最后使用最优惠，

对于92折来说，

- 先用无门槛后打折(x-5y)*0.92 = x*0.92 - 5*0.92*y
- 先打折后用无门槛 x*0.92 - 5y

对比可以看出，先92折，再无门槛最优惠，因此确实可以直接排除KM和KN的情况，即先无门槛的情况。

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])]);
```

JavaScript算法源码

复制

```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) {
  while (price >= 100 && m > 0) {
    price -= Math.floor(price / 100) * 10; // 假设price=340，那么可以优惠 340/100 * 10 = 30元
    m--;
  }
  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);
    n--;
  }
  return [price, n];
}
```

```javascript
/**
 * @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) {
  while (price >= 100 && m > 0) {
    price -= Math.floor(price / 100) * 10; // 假设price=340，那么可以优惠 340/100 * 10 = 30元
    m--;
  }
  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);
    n--;
  }
  return [price, n];
}

/**
 * @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

```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
      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) ->
```

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：剩余多
      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) {
    while (price >= 100 && m > 0) {
      price -= price / 100 * 10; // 假设price=340，那么可以优惠 340/100 * 10 = 30元
      m--;
    }
    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));
      n--;
    }
    return new int[] {price, n};
  }

  /**
   * @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;
```

```
    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));
      n--;
    }
    return new int[] {price, n};
  }

  /**
   * @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）
    """
    while price >= 100 and m > 0:
        price -= int(price / 100) * 10
        m -= 1

    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)
        n -= 1
    return price, n


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])))
```

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）
    """
    while price >= 100 and m > 0:
        price -= int(price / 100) * 10
        m -= 1

    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)
        n -= 1
    return price, n


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] + resM

    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])))
```