题目描述

城门是一个传统的安全设施，N 个人围成一圈，按顺时针从 1 到N编号。

编号为 1的人从 1开始喊数，下一个人喊的数字为上一个人的数字加 1，但是当将要喊出来的数字是 7的倍数或者数字本身含有 7的话，下面的这个数字要跳过喊出来，而是喊过了。

假设城门处于安全状态则 N个人都没有失误的在正确的顺序完成了“过”，当喊到数字N时，可以进行最今人喊“过”的次数。

现给定一个长度为M的数组，存储了打乱顺序的每个人喊“过”的次数，请把它还原成正确的顺序，原数组的第 i个元素存储编号为i的人喊“过”的次数。

输入描述

输入为一行，为空格分隔的喊“过”的次数，注意M大于等于1，N不超过200，而每个数字个数不为M。

输出描述

输出为一行，为顺序正确的喊“过”的次数，也是空格分隔。

用例

输入	0 1 0
输出	1 0 0
说明	一共只有一次喊“过”，那只会发生在最最编号7时，按题意，编号为1的人会喊到7，故输出1 0 0。
注意	结果数字允许一定重了，也可以是0，1等，输出这四次数据是1 0 0。

输入	0 0 0 2 1
输出	0 2 0 1 0
说明	一共有三次喊“过”，按编号7 14 17，编号3，编号为的人会喊到7，编号为4的人会喊到14，最后编号为5的人会喊到17。

题目解析

这题通过输入的字符串可以提取出两个信息：

- 人数
- 叫过的次数

通过叫过的次数，我们可以推测出最后一次过轮的时候叫了最几，比如叫了3次过，则说明最7，14，17，而第三次说道是第17，

然后就是通过人数，结合“过”的值，就可以反编推出人叫了“过”。

比如5个人，一共喊过了，则

- 7必然是 7 % 5 = 2，第2个人喊的
- 14必然是 14 % 5 = 4，第4个人喊的
- 17必然是 17 % 5 = 2，第2个人喊的

因此5个人的喊“过”的次数分别是：
0 2 0 1 0

Java算法源码

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

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

    int[ge=[] arr =
        Arrays.stream(sc.nextline().split(" ").map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static String getResult(Integer[] arr) {
    // 计算总人数，即数组的长度
    int totalNo = Arrays.stream(arr).reduce(0,Integer::sum).orElse(0);

    // 创建一个数组
    int n = arr.length;
    // 创建一个数组，最多可能有这么多
    int[] p = new int[n];

    // 初始化数组，所有元素都设为0
    Arrayslist<Integer> ge = new Arrayslist<>();

    int i = 7;
    while (ge.size() != totalNo) {
      if (i % 7 == 0 || (i + "").contains("7")) {
        ge.add(i);
      }
      i++;
    }

    for (Integer ide : ge) {
      p[ide % n - 1]++;
    }

    StringJoiner sj = new StringJoiner(" ");
    for (int t : p) {
      sj.add(t + "");
    }
    return sj.toString();
  }
}
```

JS算法源码

```
// 引入readline模块，用于处理输入输出
const readline = require("readline");

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

rl.on("line", (line) => {
  const arr = line.split(" ").map(Number);

  // 计算总人数，即数组的长度
  let totalNo = arr.reduce((p, c) => p + c);

  // 创建一个数组
  const n = arr.length;
  // 创建一个数组，最多可能有这么多
  const p = new Array(n).fill(0);

  // 初始化数组，所有元素都设为0
  const ge = [];

  let i = 7;
  while (ge.length != totalNo) {
    if (i % 7 == 0 || String(i).indexOf(7) != -1) {
      ge.push(i);
    }
    i++;
  }

  ge.forEach((ide) => {
    p[ide % n - 1]++;
  });

  console.log(p.join(" "));
});
```

Python算法源码

```
# 输入处理
arr = list(map(int, input().split()))


# 主函数
def getResult():
    # 计算总人数，即数组的长度
    totalNo = sum(arr)

    # 创建一个数组
    n = len(arr)
    # 创建一个数组，最多可能有这么多
    p = [0]*n

    # 初始化数组，所有元素都设为0
    ge = []

    i = 7
    while len(ge) != totalNo:
        if i % 7 == 0 or str(i).find("7") != -1:
            ge.append(i)
        i += 1

    for ide in ge:
        p[ide % n - 1] += 1

    return    .join(map(str, p))


# 输出结果
print(getResult())
```

解法二

另一种，采用数组解法，所谓环形数组，指数组的尾部元素的下一个是数组头部元素

Java算法源码

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

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

    Integer[] arr =
        Arrays.stream(sc.nextline().split(" ").map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static String getResult(Integer[] arr) {
    // 计算总人数，即数组的长度
    int totalNo = Arrays.stream(arr).reduce(0,Integer::sum).orElse(0);

    // 创建一个数组
    int n = arr.length;
    // 创建一个数组，最多可能有这么多
    int[] p = new int[n];

    // 初始化数组，所有元素都设为0
    int[ ] = 0;
    int j = 0;
    while (totalNo > 0) {
      if (i % 7 == 0 || (i + "").contains("7")) {
        totalNo--;
        p[j]++;
      }
      i++;
      j++;

      if (j == n) j = 0;
    }

    StringJoiner sj = new StringJoiner(" ");
    for (int t : p) {
      sj.add(t + "");
    }
    return sj.toString();
  }
}
```

JS算法源码

```
// 引入readline模块，用于处理输入输出
const readline = require("readline");

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

rl.on("line", (line) => {
  const arr = line.split(" ").map(Number);

  // 创建一个数组
  const n = arr.length;
  // 创建一个数组，最多可能有这么多
  const p = new Array(n).fill(0);

  // 计算总人数，即数组的长度
  let totalNo = arr.reduce((p, c) => p + c);

  // 用来存储结果，初始化为0，长度为n
  let i = 1;
  let j = 0;
  while (totalNo) {
    if (i % 7 == 0 || String(i).indexOf(7) != -1) {
      totalNo--;
      p[j]++;
    }

    i++;
    j++;

    if (j == n) {
      j = 0;
    }
  }

  console.log(p.join(" "));
});
```

Python算法源码

```
# 输入处理
arr = list(map(int, input().split()))


# 主函数
def getResult():
    # 计算总人数，即数组的长度
    totalNo = sum(arr)

    # 创建一个数组
    n = len(arr)
    # 创建一个数组，最多可能有这么多
    p = [0]*n

    i = 1

    j = 0
    while totalNo > 0:
        if i % 7 == 0 or str(i).find("7") != -1:
            totalNo -= 1
            p[j] += 1
        i += 1
        j += 1

        if j >= n:
            j = 0

    return " ".join(map(str, p))


# 输出结果
print(getResult())
```