简单的 LRU 问题

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

众所周知，电脑能够运行远大于其内存的程序，这得益于**局部性原理**，也就是说你现在用到的内存及其周围的内存空间，在不久的将来也会被频繁的调用，想想你写的 **for** 语句就能明白。

因此操作系统会动态的将暂时用不到的内存块复制转移到了外存中，同时将正在或不久的将来会用到的部分留在了内存中。物理上内存的总量并没有增加，但逻辑上你能使用的内存变多了。

现在我们简单的模拟一下操作系统的工作。你有一大块内存空间，它被平均地分为了 n 块，硬盘中也有大量的空间，它们被分为了若干和内存中块大小一样的块。当操作系统想要访问某一块空间时，它会查看内存中时候存在这一块，如果有就直接访问，如果没有，会去外存中寻找到它，并将它放入内存。如果此时内存已满，就不得不将内存中现有的块替换出去。

你正在使用一个极其消耗内存的程序，频繁的访问了一系列内存空间。在此我们将其简化为一个数字序列 a，表示你依次访问了这些内存空间。

操作系统采用了 **最近最久未使用算法(LRU)**，也就是选择最近最长时间没有使用的块予以淘汰。注意这一步只发生在有新的块想从外存进入内存。

聪明的你想必已经理解了这一切，请你画一张表格描述这一切叭。

如果你没搞懂什么是 **LRU**，可以阅读以下算法:

对于每个新的「内存块」请求，执行以下操作:

- 0. 如果已经有相同的「内存块」存在于内存中，将该「内存块」置为最低优先级，原先优先级低于它的自动提升一级。
- 1. 否则如果内存中的 n 个块没有摆满，直接将新的「内存块」请求作为最低优先级放入内存中。
- 2. 否则内存中 n 个块已被摆满，将当前内存中优先级最高的块清除出内存，将其他内存块优先级自动提升一级，将新的「内存块」作为最低优先级放入内存中。

Input

第一行两个数字 $n, m (1 \le n \le 5, 1 \le m \le 100)$ 表示内存被平均分为了 n 个块，操作系统共按序访问了 m 个内存块。

第二行一个长度为 m 的序列 a，$a_i (0 \le a_i \le 65535)$ 表示第 i 个被访问的内存块的序号。

Output

输出共 $m+1$ 行 $n+1$ 列。

第一行标记内存块淘汰的优先级，数字越小优先级越高。

接下去 m 行每行 $n+1$ 列，第一列表示访问内存的次数，下标从 0 开始，之后 n 列按淘汰优先级列出每个内存块的编号，无编号则留空。

所有内存块编号采用 4 位 16 进制数表示，优先级和访存次数用 2 位 16 进制数表示。

具体格式参考样例。

Examples

standard input	standard output
3 4 0 1 2 1	+------+--------+--------+--------+ | | 0x00 | 0x01 | 0x02 | +------+--------+--------+--------+ | 0x00 | 0x0000 | | | +------+--------+--------+--------+ | 0x01 | 0x0000 | 0x0001 | | +------+--------+--------+--------+ | 0x02 | 0x0000 | 0x0001 | 0x0002 | +------+--------+--------+--------+ | 0x03 | 0x0000 | 0x0002 | 0x0001 | +------+--------+--------+--------+
3 4 1 2 65535 255	+------+--------+--------+--------+ | | 0x00 | 0x01 | 0x02 | +------+--------+--------+--------+ | 0x00 | 0x0001 | | | +------+--------+--------+--------+ | 0x01 | 0x0001 | 0x0002 | | +------+--------+--------+--------+ | 0x02 | 0x0001 | 0x0002 | 0xffff | +------+--------+--------+--------+ | 0x03 | 0x0002 | 0xffff | 0x00ff | +------+--------+--------+--------+