Problem A. 括号序列计分

输入文件:	标准输入 *(standard input)*
输出文件:	标准输出 *(standard output)*
时间限制:	1 秒
内存限制:	512 mebibytes

给一个括号序列，定义如下

1. `()` 是一个括号序列
2. 如果 A 是括号序列，那么 `(A)` 是括号序列
3. 如果 A, B 是括号序列，那么 `AB` 是括号序列

括号序列的得分如下

1. `()` 的得分是 1
2. 如果 A 是括号序列，记其得分为 S_A，那么 `(A)` 是括号序列，其得分为 $2S_A$
3. 如果 A, B 是括号序列，记其得分分别为 S_A 和 S_B，那么 `AB` 是括号序列，其得分为 $S_A + S_B$

最终求序列的得分，结果对 `12345678910` 取模。

输入格式

第一行，一个整数 $1 \le N \le 100000$，表示序列长度。下接 N 行，每行一个整数 `0` 或者 `1`，分别表示 `'('` 和 `')'`，其中 `0` 是 `'('`。

输出格式

一个整数，得分取模后的值。

样例

标准输入 *(standard input)*	标准输出 *(standard output)*
6 0 0 1 1 0 1	3

Problem B. 冰镇矩阵

输入文件:	标准输入 *(standard input)*
输出文件:	标准输出 *(standard output)*
时间限制:	4 秒
内存限制:	512 mebibytes

给一个 n 行 m 列的矩阵，每个元素上为一个正整数。

而这些正整数太大了，想要把矩阵重新标记正整数，使得新的矩阵中最大的正整数尽可能小。但要求新矩阵的每一行、每一列中，原来相同的元素依然相同，并且元素间的相对大小不变。

正式地讲，记原矩阵第 i 行、第 j 列记录的正整数为 $a_{i,j}$，新矩阵的为 $a'_{i,j}$。

1. 若第 i 行中，存在某 p, q，有 $a_{i,p} = a_{i,q}$，则要求 $a'_{i,p} = a'_{i,q}$；
2. 若第 i 行中，存在某 p, q，有 $a_{i,p} < a_{i,q}$，则要求 $a'_{i,p} < a'_{i,q}$；
3. 若第 j 列中，存在某 p, q，有 $a_{p,j} = a_{q,j}$，则要求 $a'_{p,j} = a'_{q,j}$；
4. 若第 j 列中，存在某 p, q，有 $a_{p,j} < a_{q,j}$，则要求 $a'_{p,j} < a'_{q,j}$。

并最小化 $\max_{i,j} a'_{i,j}$。

请你输出最小化的新矩阵中的最大值，$\max_{i,j} a'_{i,j}$。

输入格式

第一行两个整数 n, m，满足 $1 \le n \cdot m \le 1,000,000$；接下来 n 行，每行 m 个正整数 $a_{i,j}$，为题意中的原矩阵，满足 $1 \le a_{i,j} \le 10^9$。

输出格式

输出一个正整数，最小化的新矩阵中的最大值，$\max_{i,j} a'_{i,j}$。

样例

标准输入 *(standard input)*	标准输出 *(standard output)*
3 4 6710 1080 2259 2259 2259 4227 2163 1080 1760 2690 2259 8070	5
5 3 310889877 310889877 906359590 426975884 975636129 634531283 906359590 887315273 906359590 950614318 975636129 416214114 493027209 310889877 356380710	7

样例解释

第一个矩阵可以重新标记为：

```
4 1 3 3
3 5 2 1
1 4 3 5
```

第二个矩阵可以重新标记为:

```
1 1 5
2 7 4
5 2 5
6 7 3
3 1 2
```

Problem C. 斯波利特平衡术

输入文件:	标准输入 *(standard input)*
输出文件:	标准输出 *(standard output)*
时间限制:	1 秒
内存限制:	512 mebibytes

现有一长度为 n，下标从 1 开始计数的数列 $\{a_i = i\}$ 的数列，即依次为 $1, 2, \cdots, n-1, n$。

“斯波利特平衡术” 是一种修改数列的操作，可以将数列中，下标在区间 $[l, r]$ 中的元素进行翻转，并且能够给出被翻转的元素的和。

正式的来讲，翻转 $[l, r]$ 中的元素，指的是得出一个新的数列 $\{a'_i\}$。其中，若 $i \notin [l, r]$，则 $a'_i = a_i$；若 $i \in [l, r]$，则 $a'_i = a_{r-i+l}$。

比如对数列 $4, 2, 3, 5, 1$，进行参数为 $l = 1$，$r = 4$ 的操作，数列变为 $5, 3, 2, 4, 1$，并给出元素和 $5 + 3 + 2 + 4 = 14$。

操作共被执行了 m 次，每次的参数 l, r 也均给出。

请你计算每次操作得到的元素和，并在最后给出整个数列经过多次操作后的最终序列。

输入格式

第一行两个整数 n, m，满足 $1 \le n, m \le 100,000$；接下来 m 行，每行两个整数 l, r，为题意中操作的参数，满足 $1 \le l \le r \le n$。

输出格式

输出共 $m + 1$ 行。

前 m 行中，第 i 行一个整数，为第 i 个操作得到的元素和；最后一行中共 n 个整数，为经过多次操作后最终的序列。

样例

标准输入 *(standard input)*	标准输出 *(standard output)*
8 4 8 8 3 7 8 8 5 7	8 25 8 12 1 2 7 6 3 4 5 8
24 10 8 21 22 22 11 24 20 22 23 23 17 18 1 1 24 24 20 22 14 18	203 22 212 45 17 23 1 18 45 50 1 2 3 4 5 6 7 21 20 19 24 23 22 11 12 10 9 8 13 14 15 16 17 18

样例解释

第一个样例中，序列 1 2 3 4 5 6 7 8 被操作了 4 次。第一次操作后序列没有变化，给出元素和 8；第二次操作后序列变为 1 2 7 6 5 4 3 8，给出元素和 25；第三次操作后序列没有变化，给出元素和 8；第四次操作后序列变为 1 2 7 6 3 4 5 8，给出元素和 12。

Problem D. 前 k 小和

输入文件:	标准输入 *(standard input)*
输出文件:	标准输出 *(standard output)*
时间限制:	1 秒
内存限制:	512 mebibytes

今有 k 个**正**整数数组，每个数组大小均为 k，故有 k^k 种方式从每个数组中恰选出一个数，然后求这些数的和，现在想问你这 k^k 个和中，前 k 小是哪些。

输入格式

第一行，一个整数 $1 \leq k \leq 1,000$；接下来 k 行，每行 k 个整数，表示一个整数数组，每个数不超过 $1,000,000$。

输出格式

共一行，k 个整数，升序排列前 k 小的和。

样例

标准输入 *(standard input)*	标准输出 *(standard output)*
3 1 8 5 9 2 5 10 7 6	9 10 12

样例解释

选择 $\{1,2,6\}$,$\{1,2,7\}$,$\{1,5,6\}$ 三组。

Problem E. npm - Node.js

输入文件:	标准输入 *(standard input)*
输出文件:	标准输出 *(standard output)*
时间限制:	1 秒
内存限制:	512 mebibytes

npm 是 Node.js 的包管理器，总共管理了 n 个包，每个包都有一个是否被安装的状态，用户使用 npm 安装或卸载了总共 q 次。

这些包之间有依赖关系，**保证依赖关系会是一棵树**，每个包会有个 id，从 0 开始标记。

0 号包没有依赖的包，$1 \le i \le n-1$ 号包只依赖一个包，这个依赖的包的 id 记作 f_i。

如果要安装 i 号包，则 i 号包的依赖包也应当被安装，并以此递归安装所有依赖包；如果要卸载 i 号包，则所有直接依赖或间接依赖（即可以递归依赖到）i 号包的包，均要被卸载。

接下来以上信息都提供给你，需要你来帮忙计算一下，每次用户使用 npm 安装或卸载包后，有多少个包的状态被修改了。

（当然，有些时候用户可能会安装已安装的包，或卸载已卸载的包。这种情况下并没有包的状态被修改，所以答案为 0。）

输入格式

第一行，一个整数 $1 \le n \le 100,000$；第二行，有 $n-1$ 个正整数，分别为 $f_1, f_2, \cdots, f_{n-1}$，$0 \le f_i < n$，$f_i \ne i$，**数据保证依赖关系会是一棵树**；第三行，一个整数 $1 \le q \le 100,000$。

接下来 q 行，每行有两个整数 $o_i \in \{0,1\}$, $0 \le x_i < n$，分别表示对应的操作和被操作的包；若 $o_i = 0$，则意味着包 x_i 被卸载；若 $o_i = 1$，则意味着包 x_i 被安装。

输出格式

输出共 q 行，每行一个整数，表示第 i 个操作后，有多少个包的状态被修改了。

样例

标准输入 *(standard input)*	标准输出 *(standard output)*
7 0 0 0 1 1 5 5 1 5 1 6 0 1 1 4 0 0	3 1 3 2 3
10 0 6 0 8 1 5 8 6 2 8 1 2 0 5 0 2 0 9 1 4 1 8 1 4 0 0	5 3 0 0 4 0 0 6

样例解释

第一个样例中：

1. 为了安装 5 号包，需要递归地安装 1 和 0 号包，共 3 个包状态被改变；
2. 为了安装 6 号包，需要递归地安装 5，1 和 0 号包，但有两个包已经安装过了，所以共 1 个包状态被改变；
3. 为了卸载 1 号包，需要卸载掉所有直接或间接依赖 1 号包的包，共 3 个包状态被改变；
4. 为了安装 4 号包，需要递归地安装 1 和 0 号包，共 2 个包状态被改变；
5. 为了卸载 0 号包，由于所有的包都直接或间接地依赖它，所以所有安装过的包都要卸载，共 3 个包状态被改变。