远古计算机（oldcomputer）

这是一道提交答案题。

【题目背景】

企鹅大陆是一片神奇的土地，在厚厚的冰层下埋藏着一个巨大的宝藏。探险家企鹅豆豆挖穿冰层到达了宝库，但他发现了一个令人发愁的问题。一共有五座宝库，每个宝库是由某些远古计算机控制的。由于年代久远，这些计算机里的程序已经消失不见了，只有给这些计算机重新填写代码并且顺利运行，输出了正确结果才能触发开门的机关。

【题目描述】

每一个宝库大门由一个计算机集群控制，计算机之间用数据线相连以便传输数据。但是有很多数据线已经损坏了，所以只留下了一部分连线。一开始数据线上没有数据，当一台计算机向数据线上写入时，数据线上就有了一个整数。每条数据线上最多可以同时传输一个整数，当整数被读取后便会消失，数据线就又回到没有数据的状态。

每台远古计算机有两个储存单元，分别名为 `a` 和 `b`，每个储存单元能够储存一个 −2147483648 到 2147483647 之间的整数。

每个时刻，每台远古计算机可以执行一条指令，一共有以下几种指令：

- mov `reg` `val` 将储存单元 `reg` 的值赋值为 `val` 的值
- add `reg` `val` 给储存单元 `reg` 加上 `val` 的值
- dec `reg` `val` 给储存单元 `reg` 减去 `val` 的值
- mul `reg` `val` 给储存单元 `reg` 乘以 `val` 的值
- div `reg` `val` 给储存单元 `reg` 除以 `val` 的值，这里的除法是向零取整的除法，如 $\frac{-5}{2} = -2$
- and `reg` `val` 给储存单元 `reg` 二进制与上 `val` 的值
- or `reg` `val` 给储存单元 `reg` 二进制或上 `val` 的值
- xor `reg` `val` 给储存单元 `reg` 二进制异或上 `val` 的值
- jmp `val` 跳转到整个程序第 `val` 条语句，语句从程序开头开始，用从 `1` 开始的正整数计数
- jz `reg` `val` 如果 `reg` 的值为 0，那么跳转到第 `val` 行
- jnz `reg` `val` 如果 `reg` 的值不为 0，那么跳转到第 `val` 行
- jgz `reg` `val` 如果 `reg` 的值严格大于 0，那么跳转到第 `val` 行
- jsz `reg` `val` 如果 `reg` 的值严格小于 0，那么跳转到第 `val` 行
- read `x` `reg` 从远古计算机 x 读取一个数到储存单元 `reg` 当中，如果数据线上缓存了一个数字，将读取这个数字并返回，否则等待下个周期再次尝试读取。$x = 0$ 时视为从标准输入数据读取一个数。

- write `val` `x` 将 `val` 的数值向远古计算机 x 方向所在数据线写入，当且仅当数据线上没有存有数据才会成功写入，否则等待下个周期再次尝试写入。$x = 0$ 时视为向标准输出数据写出一个数。

`reg` 表示一个储存单元，只能为 `a` 或者 `b` 之一。

`val` 表示一个储存单元或者一个数字的值，比如填入 `a` 表示 `a` 中储存的值或者填入 `233` 表示 `233` 这个数字。

一台远古计算机读写指令中 `x` 只有与当前远古计算机直接有数据线相连，或者 `x=0` 才被视为合法指令。

每台远古计算机的标准输入输出独立，远古计算机之间互不影响，即每台远古计算机都有独立的一个标准输入端和一个标准输出端。

每个周期计算时，所有需要执行 `write` 指令的远古计算机先计算，然后需要执行 `read` 指令的远古计算机再计算，需要执行其余指令的远古计算机最后计算。

在读取时，如果标准输入数据没有任何可以继续读取的数据，该远古计算机将进入永远等待状态。

一台远古计算机如果执行完了最后一条指令，将会重新从第一条指令开始执行。

如果一台远古计算机没有任何指令，该计算机将永远处于等待状态。

指令计数是从 `1` 开始的正整数。

一条数据线上最多只能暂存一个数据，两台计算机之间只有一条数据线，即可能读取自己上一轮写入的数据，如果两端的远古计算机同时读取或写入同一条数据线上的数据，结果将不可预知。

不存在写入标准输入的方法或是从标准输出当中读取数据的方法。

比如如下样例是从 `1` 号计算机的标准输入读入两个数，并从 `2` 号计算机的标准输出输出两个数之和。

```
node 1
read 0 a
read 0 b
write a 1
write b 1
node 2
read 1 a
read 1 b
add a b
write a 0
```

而以下写法是错误的

```
node 1
read 0 a
```

```
read 0 b
add a b
write a 0
node 2
mov a a
```

因为正确答案中，一号远古计算机的标准输出为空，而二号远古计算机的标准输出才是两个数之和。

【输入格式】

这是一道提交答案题，共有 5 组输入数据，这些数据命名为 ***oldcomputer1.in*** ~ ***oldcomputer5.in***。

这些文件描述了远古计算机之间的连线状态。

文件的第一行是三个非负整数 x、n 和 m，表示测试点编号、远古计算机的个数与他们之间的连线条数，远古计算机是从 1 到 n 标号的。

接下来 m 行，每行两个正整数 x,y，表示计算机 x 和计算机 y 之间通过数据线直接相连。

【输出格式】

对于每组输入数据，你需要提交相应的输出文件 ***oldcomputer1.out*** ~ ***oldcomputer5.out***。

这些文件描述了每台远古计算机的代码内容。

文件由多个代码块组成，每个代码块的具体格式如下：

第一行为一个字符串 node 与一个 1 到 n 之间的整数 a，由一个空格隔开，表示接下来是计算机 a 的指令。

接下来多行为该计算机的具体指令内容。

每台计算机的指令应当最多出现一次，否则将会出现未知错误。

所有计算机的指令总条数应当不超过 10^6 行。

【子任务】

子任务 1：

1 号远古计算机的标准输入将会有不超过 100 个非负整数，按照原顺序输出到 1 号远古计算机的输出当中。

子任务 2：

1 号远古计算机一个非负整数 k ，按照原输入顺序将斐波那契数列第 k 项输出到 1 号点的标准输出当中，输入数据保证第 k 项不超过 10^9。斐波那契数列通项公式为 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2} (2 \le n)$。

子任务 3：

1 号远古计算机的标准输入将会有不超过 100 个非负整数，按照原顺序输出到 n 号远古计算机的输出当中。

子任务 4：

第 1 到 50 号远古计算机分别输入 1 个数，将这 50 个数从 51 到 100 号远古计算机输出，输出顺序任意，每个远古计算机输出数字个数任意。

子任务 5：

第 1 到 10 号远古计算机各输入 1 个数，将这些数对应从 100 到 91 号远古计算机输出，即 i 号点输入的数需要从 $101 - i$ 号点输出。

【评分方式】

对于每个测试点，有三个参数 $a1, a2, a3$。

如果选手提交的代码在 $a1$ 个周期内正确输出了答案，将会得到该测试点 100% 的分。

如果选手提交的代码在 $a2$ 个周期内正确输出了答案，将会得到该测试点 60% 的分。

如果选手提交的代码在 $a3$ 个周期内正确输出了答案，将会得到该测试点 30% 的分。

即在每个周期结束时，会分别进行一次答案正确性判断，以最早正确的一次为准。

【提示】

以下为 `checker` 的正确使用方式。

- `./checker graph.in input.in code.out x -detail`
  - 其中 `graph.in` 为连线关系描述文件。
  - `input.in` 为输入数据。
  - `code.out` 为选手提交的文件。
  - `x` 为一个非负整数表示执行周期数。
  - `-detail` 为一个可选字段，附加上这个字段后将会输出每个周期每个远古计算机执行的代码内容到 `detail.out`。

  - checker 将会运行选手的代码，输出 x 个周期结束时的运行结果到 result.out 中。

- 输入数据 input.in 格式要求如下：
  - 一共 n 行，每行多个整数，表示对应的计算机的标准输入内容。
- 输出数据 result.out 格式如下：
  - 第一行一个字符串为返回的信息。
  - 如果正确运行，还会有 n 行，每行多个整数，表示对应的计算机的标准输出内容。

下发文件一共包括 1.in,2.in,3.in,4.in,5.in,checker。

【子任务分值】

在本场比赛中，测试点（或子任务）的分值分布与你是否为集训队选手有关。本题的分值设置如下：

子任务编号	1	2	3	4	5
集训队分值	10	15	15	30	30
非集训队分值	20	20	20	20	20