#include <cstdio>

#include <cassert>

#include <cstdlib>

// 这里定义了一些与题目规模或常量有关的宏

#define MAX\_DISK\_NUM (10 + 1) // 最大硬盘数(示例中预留10个+1)

#define MAX\_DISK\_SIZE (16384 + 1) // 每块硬盘最大存储单元数量(示例中预留16384+1)

#define MAX\_REQUEST\_NUM (30000000 + 1) // 最大读请求数量

#define MAX\_OBJECT\_NUM (100000 + 1) // 最大对象数量

#define REP\_NUM (3) // 冗余副本数量(要求3份)

#define FRE\_PER\_SLICING (1800) // 时间片相关(示例中只演示，不深入使用)

#define EXTRA\_TIME (105) // 额外的时间片数量

// 请求结构体，用于保存某次读请求的信息

typedef struct Request\_ {

int object\_id; // 需要读取的对象ID

int prev\_id; // 链表式指向上一个请求的ID(用于回溯)

bool is\_done; // 该请求是否已完成

} Request;

// 对象结构体，用于保存对象信息

typedef struct Object\_ {

int replica[REP\_NUM + 1]; // 记录对象的三个副本在哪些硬盘上

int\* unit[REP\_NUM + 1]; // 每个副本对应的存储单元编号数组

int size; // 对象大小(单位: 对象块数)

int last\_request\_point; // 记录最后一次发起读请求的编号(用于回溯、取消等)

bool is\_delete; // 标记该对象是否已被删除

} Object;

// 全局数组

Request request[MAX\_REQUEST\_NUM];

Object object[MAX\_OBJECT\_NUM];

/\* 一些全局变量：

\* 一共T + EXTRA\_TIME 个时间片

\* M种对象标签，编号1到M，输入数据保证1 <= M <= 16

\* N块硬盘，编号1到N，输入数据保证1 <= N <= 10

\* 每块硬盘由V个存储单元构成，存储单元编号1到V，输入数据保证1 <= V <= 16384，任何时间存储系统中空余的存储单元数占总存储单元数的至少10%。

\* 每个磁头在每个时间片最多消耗G token，输入数据保证64 <= G <= 1000

\*/

int T, M, N, V, G;

int disk[MAX\_DISK\_NUM][MAX\_DISK\_SIZE]; // 模拟硬盘，disk[i][j]存储对象ID

int disk\_point[MAX\_DISK\_NUM]; // 如果需要按顺序分配存储单元，可以在此记录下一个可分配位置

// 输出当前时间片，并读取(丢弃)判题器给的"TIMESTAMP"

void timestamp\_action()

{

int timestamp;

// 读入格式："TIMESTAMP X"

// %\*s表示跳过一个字符串(即"TIMESTAMP")，读入后面的X

scanf("%\*s%d", &timestamp);

// 按照题目规则，原样输出"TIMESTAMP X"

printf("TIMESTAMP %d\n", timestamp);

fflush(stdout);

}

// 从硬盘上删除object\_unit[]所对应的存储单元(即清零)

void do\_object\_delete(const int\* object\_unit, int\* disk\_unit, int size)

{

for (int i = 1; i <= size; i++) {

disk\_unit[object\_unit[i]] = 0; // 将硬盘对应存储单元清空

}

}

// 删除动作处理函数

void delete\_action()

{

int n\_delete;

int abort\_num = 0; // 需要被取消的读请求计数

static int \_id[MAX\_OBJECT\_NUM];

// 读入本时间片删除的对象数量n\_delete

scanf("%d", &n\_delete);

// 依次读入要删除的对象id

for (int i = 1; i <= n\_delete; i++) {

scanf("%d", &\_id[i]);

}

// 统计所有要删除对象所关联的未完成读请求(后续需要输出并取消)

for (int i = 1; i <= n\_delete; i++) {

int id = \_id[i];

int current\_id = object[id].last\_request\_point;

// 回溯该对象最近所有读请求，如果存在 is\_done == false，就要被取消

while (current\_id != 0) {

if (request[current\_id].is\_done == false) {

abort\_num++;

}

current\_id = request[current\_id].prev\_id;

}

}

// 输出总的取消请求数量

printf("%d\n", abort\_num);

// 逐个对象输出需要取消的请求编号，并真正删除硬盘上的数据

for (int i = 1; i <= n\_delete; i++) {

int id = \_id[i];

int current\_id = object[id].last\_request\_point;

// 输出所有未完成的读请求编号

while (current\_id != 0) {

if (request[current\_id].is\_done == false) {

printf("%d\n", current\_id);

}

current\_id = request[current\_id].prev\_id;

}

// 真正删除对应硬盘上的内容

for (int j = 1; j <= REP\_NUM; j++) {

do\_object\_delete(object[id].unit[j],

disk[object[id].replica[j]],

object[id].size);

}

// 标记该对象已删除

object[id].is\_delete = true;

}

fflush(stdout);

}

// 在 disk\_unit 对应硬盘中，为 object\_id 分配 size 个单元，并写入(示例中写入就是标记disk\_unit[i] = object\_id)

void do\_object\_write(int\* object\_unit, int\* disk\_unit, int size, int object\_id)

{

int current\_write\_point = 0;

// 这里从第1个存储单元开始扫描，找到空闲单元就写入

for (int i = 1; i <= V; i++) {

if (disk\_unit[i] == 0) {

disk\_unit[i] = object\_id;

object\_unit[++current\_write\_point] = i;

if (current\_write\_point == size) {

break;

}

}

}

// 如果没找到足够空间则会断言失败

assert(current\_write\_point == size);

}

/\*

\* n\_write：代表这一时间片写入对象的个数。 输入数据保证总写入次数小于等于100000。

\* 接下来 n\_write 行，每行三个数 obj\_id[i]、 obj\_size[i]、 obj\_tag[i]，代表当前时间片写入的对象编

号， 对象大小，对象标签编号。

\* 输入数据保证 obj\_id 为1开始每次递增1的整数， 且1 ≤ 𝑜𝑏𝑗\_𝑠𝑖𝑧𝑒[𝑖] ≤ 5，1 ≤ 𝑜𝑏𝑗\_𝑡𝑎𝑔[𝑖] ≤ 𝑀

\*/

// 写入动作处理函数

void write\_action()

{

int n\_write;

// 读取本时间片要写入的对象数量

scanf("%d", &n\_write);

// 对每个要写入的对象依次处理

for (int i = 1; i <= n\_write; i++) {

int id, size;

// 读入对象id、大小，tag(本示例里忽略了tag，用%\*d跳过)

scanf("%d%d%\*d", &id, &size);

// 初始化该对象的last\_request\_point等信息

object[id].last\_request\_point = 0;

for (int j = 1; j <= REP\_NUM; j++) {

// 副本放到 (id + j) % N + 1 的硬盘上(演示用)

object[id].replica[j] = (id + j) % N + 1;

// 为对象的第 j 副本分配一个int数组，用于记录写在哪些存储单元

object[id].unit[j] = static\_cast<int\*>(malloc(sizeof(int) \* (size + 1)));

object[id].size = size;

object[id].is\_delete = false;

// 调用写入函数，把这份副本写到指定硬盘

do\_object\_write(object[id].unit[j],

disk[object[id].replica[j]],

size,

id);

}

/\*

\* 输出包含4 ∗ 𝑛\_𝑤𝑟𝑖𝑡𝑒行，每4行代表一个对象：

\* 第一行一个整数 obj\_id[i]，表示该对象的对象编号。

\* 接下来一行，第一个整数 rep[1]表示该对象的第一个副本写入的硬盘编号，接下来对象大小(obj\_size)个整数 unit[1][j]，代表第一个副本第𝑗个对象块写入的存储单元编号。

\* 第三行，第四行格式与第二行相同，为写入第二，第三个副本的结果。

\*/

printf("%d\n", id);

for (int j = 1; j <= REP\_NUM; j++) {

printf("%d", object[id].replica[j]);

for (int k = 1; k <= size; k++) {

printf(" %d", object[id].unit[j][k]);

}

printf("\n");

}

}

fflush(stdout);

}

/\*

\* n\_read：代表这一时间片读取对象的个数。 输入数据保证总读取次数小于等于30000000。

\* 接下来 n\_read 行，每行两个数 req\_id[i]、 obj\_id[i]，代表当前时间片读取的请求编号和请求的对象

编号。 输入数据保证读请求编号为 1 开始每次递增 1 的整数， 读取的对象在请求到来的时刻一定在存储系

统中。

\*/

// 读取动作处理函数

void read\_action()

{

int n\_read;

int request\_id, object\_id;

// 本时间片有多少个读请求

scanf("%d", &n\_read);

// 读取所有读请求，并串成“单向链表”，方便回溯

for (int i = 1; i <= n\_read; i++) {

scanf("%d%d", &request\_id, &object\_id);

// 记录该读请求对应的对象、上一个请求ID

request[request\_id].object\_id = object\_id;

request[request\_id].prev\_id = object[object\_id].last\_request\_point;

// 将该请求链到对象的末尾

object[object\_id].last\_request\_point = request\_id;

// 刚到来的请求标记未完成

request[request\_id].is\_done = false;

}

// 用于演示的静态变量，表示“当前正在处理的请求”和“读到第几步”

static int current\_request = 0;

static int current\_phase = 0;

// 如果当前没有正在处理的请求，而且本时间片新来了读请求，就先把最新那个设置为“当前请求”

if (!current\_request && n\_read > 0) {

current\_request = request\_id;

}

// 每个时间片需要输出N行磁头动作，然后再输出一个数字(上报完成请求数量)

if (!current\_request) {

// 如果当前没有要处理的请求，则所有磁头都无动作

for (int i = 1; i <= N; i++) {

printf("#\n");

}

// 无请求完成

printf("0\n");

}

else {

// 有正在进行的请求

current\_phase++;

object\_id = request[current\_request].object\_id;

// 输出N行磁头动作(示例演示：只有副本[1]的硬盘会做一点读或跳过)

for (int i = 1; i <= N; i++) {

// 如果这是该对象第一份副本所在的硬盘

if (i == object[object\_id].replica[1]) {

// 演示：奇数phase时做一次“j(ump)/read”的切换，偶数phase时输出"r#"或类似

if (current\_phase % 2 == 1) {

// 假装在奇数phase时，用“j x”动作

printf("j %d\n", object[object\_id].unit[1][current\_phase / 2 + 1]);

}

else {

// 偶数phase时，输出"r#"让磁头读

printf("r#\n");

}

}

else {

// 其他硬盘无动作

printf("#\n");

}

}

// 如果phase达到对象大小 \* 2，就说明已将该对象的全部块“读取”完

if (current\_phase == object[object\_id].size \* 2) {

// 如果对象已被删除，就上报0(表示没有请求完成)

if (object[object\_id].is\_delete) {

printf("0\n");

}

else {

// 否则上报1个请求完成，并输出该请求编号

printf("1\n%d\n", current\_request);

// 标记此请求已完成

request[current\_request].is\_done = true;

}

// 重置

current\_request = 0;

current\_phase = 0;

}

else {

// 否则本时间片不上报任何完成

printf("0\n");

}

}

fflush(stdout);

}

// 程序结束前释放所有动态申请的对象副本单元

void clean()

{

for (auto& obj : object) {

for (int i = 1; i <= REP\_NUM; i++) {

if (obj.unit[i] == nullptr)

continue;

free(obj.unit[i]);

obj.unit[i] = nullptr;

}

}

}

int main()

{

// 读入T, M, N, V, G

scanf("%d%d%d%d%d", &T, &M, &N, &V, &G);

// 此处根据赛题输入格式，示例中直接跳过一些输入(3 \* M行，每行ceil(T/1800)个数)

for (int i = 1; i <= M; i++) {

for (int j = 1; j <= (T - 1) / FRE\_PER\_SLICING + 1; j++) {

scanf("%\*d");

}

}

for (int i = 1; i <= M; i++) {

for (int j = 1; j <= (T - 1) / FRE\_PER\_SLICING + 1; j++) {

scanf("%\*d");

}

}

for (int i = 1; i <= M; i++) {

for (int j = 1; j <= (T - 1) / FRE\_PER\_SLICING + 1; j++) {

scanf("%\*d");

}

}

// 输出"OK"表示预处理阶段完成

printf("OK\n");

fflush(stdout);

// 初始化硬盘指针(在本示例中不太用得到)

for (int i = 1; i <= N; i++) {

disk\_point[i] = 1;

}

// 主循环：从时间片1到T + EXTRA\_TIME

for (int t = 1; t <= T + EXTRA\_TIME; t++) {

timestamp\_action(); // 1. 时间片对齐

delete\_action(); // 2. 删除操作

write\_action(); // 3. 写入操作

read\_action(); // 4. 读取操作

}

// 收尾：释放动态分配的对象副本空间

clean();

return 0;

}