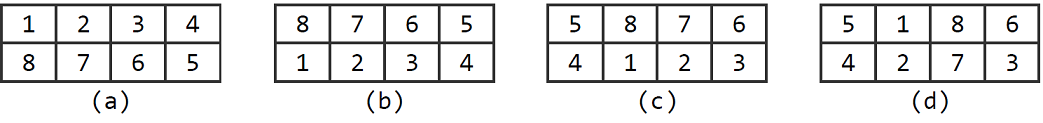
**THU2015 spring 4-1 Toy**

**描述**

ZC神最擅长逻辑推理，一日，他给大家讲述起自己儿时的数字玩具。

该玩具酷似魔方，又不是魔方。具体来说，它不是一个3 \* 3 \* 3的结构，而是4 \* 2的结构。



按照该玩具约定的玩法，我们可反复地以如下三种方式对其做变换：

**A． 交换上下两行。比如，图(a)经此变换后结果如图(b)所示。**

**B． 循环右移（ZC神从小就懂得这是什么意思的）。比如，图(b)经此变换后结果如图(c)所示。**

**C． 中心顺时针旋转。比如，图(c)经此变换后结果如图(d)所示。**

ZC神自小就是这方面的天才，他往往是一只手还没揩干鼻涕，另一只手已经迅速地将处于任意状态的玩具复原至如图(a)所示的初始状态。物质极其匮乏的当年，ZC神只有一个这样的玩具；物质极大丰富的今天，你已拥有多个处于不同状态的玩具。现在，就请将它们全部复原吧。

**输入**

第一行是一个正整数，即你拥有的魔方玩具总数N。

接下来共N行，每行8个正整数，表示该玩具的当前状态。

这里，魔方状态的表示规则为：前四个数自左向右给出魔方的第一行，后四个数自右向左给出第二行。比如，初始状态表示为“1 2 3 4 5 6 7 8”。

**输出**

共N行，各含一个整数，依次对应于复原各玩具所需执行变换的最少次数。

特别地，若某个玩具不可复原，则相应行输出-1。

**输入样例**

2

1 2 3 4 5 6 7 8

8 6 3 5 4 2 7 1

**输出样例**

0

2

**限制**

对于60%的数据，N = 1

对于100%的数据，1 <= N <= 1,000

时间：1 sec

空间：20MB

**一级提示**

状态转换图及其搜索

**THU2015 spring 4-2 Whistory**

**描述**

历史的车轮缓缓碾过，碾过的历史刻在车轮上。

车轮上刻有一圈共n个英文小写字母，滚滚向前每碾过一圈就留下一条印迹，这类印迹都是长度为n的字符串。显然，从不同位置开始碾，可能得到不同的印 迹。既然历史的车轮滚滚向前，故不必考虑向后碾的情况。于是，一个车轮最多对应于n条不同的印迹；反之，不同的印迹也可能来自相同的车轮。

也就是说，两条印迹**若能通过循环移位相互转换，则它们来自同一条车轮**。所谓的循环左移k位，是指将字符串"s1 ... sk sk+1 ... sn "转换为"sk+1 ... sn s1 ... sk "。

现在，已知不同车轮的n条印迹，请你核对它们所属的车轮。

**输入**

第一行有两个数，即印迹总数m和车轮周长n。

接下来的m行分别给出m条印迹。

**输出**

共m行，依次对应于m条印迹。

若第i（i=0,1,...,m-1）条印迹与之前的第k条（k=0,1,...,i-1）印迹来自同一车轮，则输出编号k；若存在多个这样的k，则输出最小者。否则（第i条印迹所来自的车轮不同于之前的所有车轮），则输出i。

**输入样例**

5 8

whistory

historyw

farewell

wellfare

orywhist

**输出样例**

0

0

2

2

0

**限制**

1 <= m <= 1,000，m是正整数

1 <= n <= 10,000，n是正整数

第一行m，n以空格分隔

之后m行印迹全部为小写英文字母，每条印迹长度均为n

时间：2s

空间：256MB

**一级提示**

1.KMP算法

2.Hash

3.循环移位的最小表示

**THU2015 spring 4-3 Min**

**描述**

目前我们已经进入了大数据（Big Data）时代，所需要处理的数据规模越来越大，不幸的是，领导交给你的机器仍然是一台老旧的电脑，其内存之低令人发指。不过领导要求你完成的任务却很严苛：希望能在海量的数据中，选出他希望的值。

具体来说，他给了你一块装有大量数据的磁盘，每个数据只包括一个int类型（有符号32bit整数），现在他希望你从中选出**最小**的k个，这本来是一个简单的任务，不过考虑到受限的内存，简单的任务也变得复杂起来。

**输入**

为了加快输入输出速度，输入文件采用二进制格式存储。前两个数字n和k，表示你需要从n个数字中选出最小的k个。之后随即连续存储了n个整数。

**输出**

输出包括k行，依次为**从大到小**的最小的k个数字，考虑到输入中的数字可能相同，这里输出的数字也可能相同。

**输入样例**

6

3

1

4

-5

1

4

-2

特别注意：实际输入均为二进制文件，所有数据连续存储，并没有行的概念。

**输出样例**

1

-2

-5

**限制**

1 <= n <= 10,000,000

1 <= k <= 50,000

k <= n

输入的数字均为有符号32位整数。

你只有非常有限的空间来完成任务。

时间：2 sec

空间：12MB

**一级提示**

自己测试时，如果对读入二进制数据不熟悉，可以以文本的形式读取数据；提交到OJ时，参考如下代码修改读取数据的方式：

int n, k;

fread(&n, sizeof(int), 1, stdin);

fread(&k, sizeof(int), 1, stdin);

int \*data = new int[n]; // 此处仅为示意，若直接开这么大的数组会超过内存限制

fread(data, sizeof(int), n, stdin);

输出请不要使用二进制方式输出，直接以人可读的方式输出结果即可。

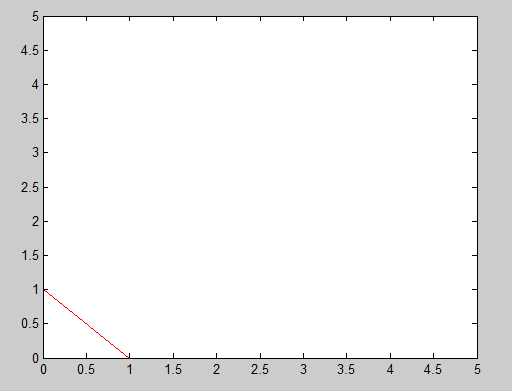
**THU2015 spring 4-4 Dynalarm**

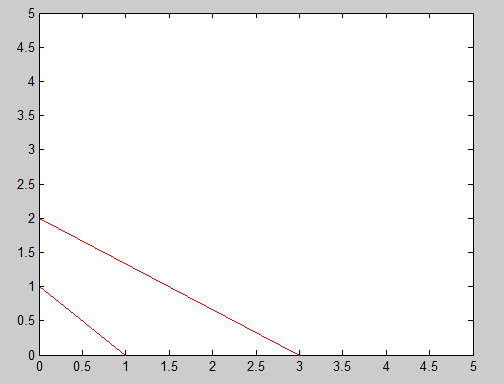
**题目描述**

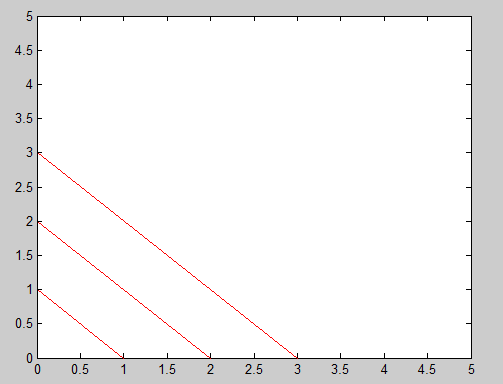
Duck公司越做越大，开始建新的厂房了。报警装置也越做越智能，有自动调整功能了：新安装一对报警装置时，若红外线相交，则这些相交的红外线的发 射器、接收器会重新配对；拆除一对报警装置时，失配的发射器、接收器也会重新匹配；如图所示。现在，施工者一边安装、拆除报警器，一边询问点处于哪个区 域。

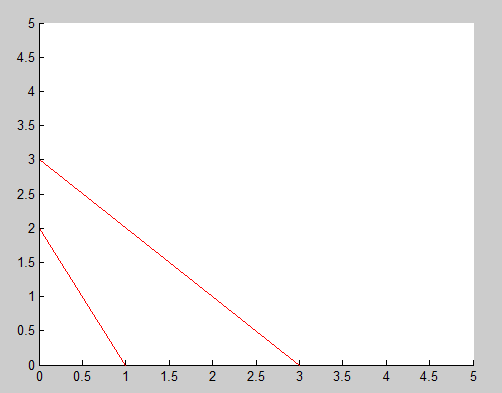
在你对施工队大发雷霆之后，他们放弃了让你拿着全键盘指挥的想法，现在你只需输入0~9这10个数字，即可轻松配置报警装置的安装、拆除和测试。

当有t个报警装置时，区域从左到右分别编号0、1、…、t。若点正好在红外线上，则视为处于左边的区域。

0 1 1 =>

0 3 2 =>

0 2 3 =>

2 1 0 =>

**输入格式**

第一行一个整数n，表示接下来有n个操作。

接下来n行，每行是以下三种操作之一

0 a b//Install：表示安装一组报警装置，发射器安装在(a, 0)，接收器安装在(0, b)

1 x y//Query：询问点(x, y)处于哪个区域

2 i j//Remove：拆除从(0, 0)开始数第i个发射器、第j个接收器

**输出格式**

若干行，每行对应一个Query，只含一个整数，表示当时点所在区域的编号

**输入样例**

6

0 1 1

1 2 2

0 3 2

0 2 3

2 1 0

1 2 2

**输出样例**

1

2

**限制**

1 <= n <= 1,000,000

0 < {a, b, x, y} < 2^31

输入保证发射器、接收器的位置不重复

输入保证拆除操作合法

时间：2sec

空间：256MB

**一级提示**

1.编程作业2-1

2.使用高效插入/排序/搜索的数据结构管理报警器数据

**任务调度(Schedule)**

**Description**

A HPS cluster is equipped with a unique task scheduler. To be simple, it is assumed that this cluster doesn’t support multiple tasks running at the same time, such that only one task is allowed to be in running state at any moment. Initially, the priority of ever task is denoted by an integer which is called priority number. **The smaller priority number stands for high priority. If two tasks have same task number, the priority is decided in the ASCII order of task name.** Following this policy, resources, such as CPU, are always occupied by the task with minimum priority number. When one task is finished, the one with minimum priority number in the rest tasks is picked to execute. The finished task won’t quit immediately. The priority number is doubled and put back to the task set. Once the priority number is greater or equal to **2^32**, this task is deleted from the task set.

Given initial priority setting of every task, your job is to predict the running order of a batch of tasks.

**Input**

First line contains two integers, says n and m. n stands for the number of tasks in initial state. m stands for the length of predicted sequence. Every line is ended by a line break symbol. In each one of the following n lines, an integer and a string are included. This string is shorter than 8, which only contains lowercase letters and numbers. The integer is priority number and the string is the task name. The integer and string is separated by space.

**Output**

At most m lines, each one contains a string. Output the name of tasks according to the order that tasks are executed. If the number of executed tasks is less than m, then output all the executed tasks.

**Example**

Input

3 3

1 hello

2 world

10 test

Output

hello

hello

world

**Restrictions**

0 <= n <= 4,000,000

0 <= m <= 2,000,000

0 < Priority number < 2^32

No tasks have same name

Time: 2 sec

Memory: 512 MB

**Hints**

Priority queue

**描述**

某高性能计算集群（HPC cluster）采用的任务调度器与众不同。为简化起见，假定该集群不支持多任务同时执行，故同一时刻只有单个任务处于执行状态。初始状态下，每个任务都由称作优先级数的一个整数指定优先级，**该数值越小优先级越高**；**若优先级数相等，则任务名ASCII字典顺序低者优先**。此后，CPU等资源总是被优先级数最小的任务占用；每一任务计算完毕，再选取优先级数最小下一任务。不过，这里的任务在计算结束后通常并不立即退出，而是将优先级数加倍（加倍计算所需的时间可以忽略）并继续参与调度；只有在优先级数不小于**2^32**时，才真正退出

你的任务是，根据初始优先级设置，按照上述调度原则，预测一批计算任务的执行序列。

**输入**

第一行为以空格分隔的两个整数n和m，n为初始时的任务总数，m为所预测的任务执行序列长度，每行末尾有一个换行符

以下n行分别包含一个整数和一个由不超过8个小写字母和数字组成的字符串。前者为任务的初始优先级数，后者为任务名。数字和字符串之间以空格分隔

**输出**

最多m行，各含一个字符串。按执行次序分别给出执行序列中前m个任务的名称，若执行序列少于m，那么输出调度器的任务处理完毕前的所有任务即可。

**样例**

见英文题面

**限制**

0 ≤ n ≤ 4,000,000

0 ≤ m ≤ 2,000,000

0 < 每个任务的初始优先级 < 2^32

不会有重名的任务

时间：2 sec

内存：512 MB

**提示**

优先级队列

**循环移位(Cycle)**

**Description**

Cycle shifting refers to following operation on the sting. Moving first letter to the end and keeping rest part of the string. For example, apply cycle shifting on ABCD will generate BCDA. Given any two strings, to judge if arbitrary times of cycle shifting on one string can generate the other one.

**Input**

There m lines in the input, while each one consists of two strings separated by space. Each string only contains uppercase letter 'A'~'Z'.

**Output**

For each line in input, output YES in case one string can be transformed into the other by cycle shifting, otherwise output NO.

**Example**

Input

AACD CDAA

ABCDEFG EFGABCD

ABCD ACBD

ABCDEFEG ABCDEE

Output

YES

YES

NO

NO

**Restrictions**

0 <= m <= 5000

1 <= |S1|, |S2| <= 10^5

Time: 2 sec

Memory: 256 MB

**描述**

所谓循环移位是指。一个字符串的首字母移到末尾, 其他字符的次序保持不变。比如ABCD经过一次循环移位后变成BCDA

给定两个字符串，判断它们是不是可以通过若干次循环移位得到彼此

**输入**

由m行组成，每行包含两个由大写字母'A'~'Z'组成的字符串，中间由空格隔开

**输出**

对于每行输入，输出这两个字符串是否可以通过循环移位得到彼此：YES表示是，NO表示否

**样例**

见英文题面

**限制**

0 ≤ m ≤ 5000

1 ≤ |S1|, |S2| ≤ 10^5

时间：2 sec

内存：256 MB