## VK Cup 2012 Round 3

# A. Variable, or There and Back Again

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Life is not easy for the perfectly common variable named Vasya. Wherever it goes, it is either assigned a value, or simply ignored, or is being used!

Vasya's life goes in states of a program. In each state, Vasya can either be used (for example, to calculate the value of another variable), or be assigned a value, or ignored. Between some states are directed (oriented) transitions.

A *path* is a sequence of states $v_1, v_2, ..., v_x$, where for any $1 \le i < x$ exists a transition from $v_i$ to $v_{i+1}$.

Vasya's value in state $v$ is interesting to the world, if exists path $p_1, p_2, ..., p_k$ such, that $p_i = v$ for some $i$ $(1 \le i \le k)$, in state $p_1$ Vasya gets assigned a value, in state $p_k$ Vasya is used and there is no state $p_i$ (except for $p_1$) where Vasya gets assigned a value.

Help Vasya, find the states in which Vasya's value is interesting to the world.

### Input

The first line contains two space-separated integers $n$ and $m$ $(1 \le n, m \le 10^5)$ — the numbers of states and transitions, correspondingly.

The second line contains space-separated $n$ integers $f_1, f_2, ..., f_n$ $(0 \le f_i \le 2)$, $f_i$ described actions performed upon Vasya in state $i$: $0$ represents ignoring, $1$ — assigning a value, $2$ — using.

Next $m$ lines contain space-separated pairs of integers $a_i, b_i$ $(1 \le a_i, b_i \le n, a_i \ne b_i)$, each pair represents the transition from the state number $a_i$ to the state number $b_i$. Between two states can be any number of transitions.

### Output

Print $n$ integers $r_1, r_2, ..., r_n$, separated by spaces or new lines. Number $r_i$ should equal $1$, if Vasya's value in state $i$ is interesting to the world and otherwise, it should equal $0$. The states are numbered from $1$ to $n$ in the order, in which they are described in the input.

**Sample test(s)**

input

```
4 3
1 0 0 2
1 2
2 3
3 4
```

output

```
1
1
1
1
```

input

```
3 1
1 0 2
1 3
```

output

```
1
0
1
```

input

```
3 1
2 0 1
1 3
```

output

```
0
0
0
```

### Note

In the first sample the program states can be used to make the only path in which the value of Vasya interests the world, 1 $\longrightarrow$ 2 $\longrightarrow$ 3 $\longrightarrow$ 4; it includes all the states, so in all of them Vasya's value is interesting to the world.

The second sample the only path in which Vasya's value is interesting to the world is , — 1 $\longrightarrow$ 3; state 2 is not included there.

In the third sample we cannot make from the states any path in which the value of Vasya would be interesting to the world, so the value of Vasya is never interesting to the world.

# B. Ancient Berland Hieroglyphs

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarpus enjoys studying Berland hieroglyphs. Once Polycarp got hold of two ancient Berland pictures, on each of which was drawn a circle of hieroglyphs. We know that no hieroglyph occurs twice in either the first or the second circle (but in can occur once in each of them).

Polycarpus wants to save these pictures on his laptop, but the problem is, laptops do not allow to write hieroglyphs circles. So Polycarp had to break each circle and write down all of its hieroglyphs in a clockwise order in one line. A line obtained from the first circle will be called $a$, and the line obtained from the second one will be called $b$.

There are quite many ways to break hieroglyphic circles, so Polycarpus chooses the method, that makes the length of the largest substring of string $a$, which occurs as a subsequence in string $b$, maximum.

Help Polycarpus — find the maximum possible length of the desired substring (subsequence) if the first and the second circles are broken optimally.

The *length* of string $s$ is the number of characters in it. If we denote the length of string $s$ as $|s|$, we can write the string as $s = s_1s_2\dots s_{|s|}$.

A *substring* of $s$ is a non-empty string $x = s[a\dots b] = s_a s_{a+1}\dots s_b$ $(1 \le a \le b \le |s|)$. For example, "`code`" and "`force`" are substrings of "`codeforces`", while "`coders`" is not.

A *subsequence* of $s$ is a non-empty string $y = s[p_1p_2\dots p_{|y|}] = s_{p_1}s_{p_2}\dots s_{p_{|y|}}$ $(1 \le p_1 < p_2 < \dots < p_{|y|} \le |s|)$. For example, "`coders`" is a subsequence of "`codeforces`".

## Input

The first line contains two integers $l_a$ and $l_b$ $(1 \le l_a, l_b \le 1000000)$ — the number of hieroglyphs in the first and second circles, respectively.

Below, due to difficulties with encoding of Berland hieroglyphs, they are given as integers from $1$ to $10^6$.

The second line contains $l_a$ integers — the hieroglyphs in the first picture, in the clockwise order, starting with one of them.

The third line contains $l_b$ integers — the hieroglyphs in the second picture, in the clockwise order, starting with one of them.

It is guaranteed that the first circle doesn't contain a hieroglyph, which occurs twice. The second circle also has this property.

## Output

Print a single number — the maximum length of the common substring and subsequence. If at any way of breaking the circles it does not exist, print `0`.

### Sample test(s)

input
```
5 4
1 2 3 4 5
1 3 5 6
```
output
```
2
```

input
```
4 6
1 3 5 2
1 2 3 4 5 6
```
output
```
3
```

input
```
3 3
1 2 3
3 2 1
```
output
```
2
```

## Note

In the first test Polycarpus picks a string that consists of hieroglyphs 5 and 1, and in the second sample — from hieroglyphs 1, 3 and 5.

# C. Machine Programming

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

One remarkable day company "X" received $k$ machines. And they were not simple machines, they were mechanical programmers! This was the last unsuccessful step before switching to android programmers, but that's another story.

The company has now $n$ tasks, for each of them we know the start time of its execution $s_i$, the duration of its execution $t_i$, and the company profit from its completion $c_i$. Any machine can perform any task, exactly one at a time. If a machine has started to perform the task, it is busy at all moments of time from $s_i$ to $s_i + t_i - 1$, inclusive, and it cannot switch to another task.

You are required to select a set of tasks which can be done with these $k$ machines, and which will bring the maximum total profit.

## Input

The first line contains two integer numbers $n$ and $k$ ($1 \le n \le 1000$, $1 \le k \le 50$) — the numbers of tasks and machines, correspondingly.

The next $n$ lines contain space-separated groups of three integers $s_i, t_i, c_i$ ($1 \le s_i, t_i \le 10^9$, $1 \le c_i \le 10^6$), $s_i$ is the time where they start executing the $i$-th task, $t_i$ is the duration of the $i$-th task and $c_i$ is the profit of its execution.

## Output

Print $n$ integers $x_1, x_2, ..., x_n$. Number $x_i$ should equal $1$, if task $i$ should be completed and otherwise it should equal $0$.

If there are several optimal solutions, print any of them.

**Sample test(s)**

input

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

output

```
0 1 1
```

input

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

output

```
1 1 0 0 1
```

## Note

In the first sample the tasks need to be executed at moments of time 2 ... 8, 1 ... 3 and 4 ... 4, correspondingly. The first task overlaps with the second and the third ones, so we can execute either task one (profit 5) or tasks two and three (profit 6).

# D. Minimum Diameter

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given $n$ points on the plane. You need to delete exactly $k$ of them $(k < n)$ so that the diameter of the set of the remaining $n$ - $k$ points were as small as possible. The diameter of a set of points is the maximum pairwise distance between the points of the set. The diameter of a one point set equals zero.

## Input

The first input line contains a pair of integers $n, k$ $(2 \le n \le 1000,\ 1 \le k \le 30,\ k < n)$ — the numbers of points on the plane and the number of points to delete, correspondingly.

Next $n$ lines describe the points, one per line. Each description consists of a pair of integers $x_i, y_i$ $(0 \le x_i, y_i \le 32000)$ — the coordinates of the $i$-th point. The given points can coincide.

## Output

Print $k$ different space-separated integers from $1$ to $n$ — the numbers of points to delete. The points are numbered in the order, in which they are given in the input from $1$ to $n$. You can print the numbers in any order. If there are multiple solutions, print any of them.

**Sample test(s)**

input

```
5 2
1 2
0 0
2 2
1 1
3 3
```

output

```
5 2
```

input

```
4 1
0 0
0 0
1 1
1 1
```

output

```
3
```

# E. Polycarpus and Tasks

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarpus has many tasks. Each task is characterized by three integers $l_i$, $r_i$ and $t_i$. Three integers $(l_i, r_i, t_i)$ mean that to perform task $i$, one needs to choose an integer $s_i$ $(l_i \le s_i;\ s_i + t_i - 1 \le r_i)$, then the task will be carried out continuously for $t_i$ units of time, starting at time $s_i$ and up to time $s_i + t_i - 1$, inclusive. In other words, a task is performed for a continuous period of time lasting $t_i$, should be started no earlier than $l_i$, and completed no later than $r_i$.

Polycarpus's tasks have a surprising property: for any task $j, k$ (with $j < k$) $l_j < l_k$ and $r_j < r_k$.

Let's suppose there is an ordered set of tasks $A$, containing $|A|$ tasks. We'll assume that $a_j = (l_j, r_j, t_j)$ $(1 \le j \le |A|)$. Also, we'll assume that the tasks are ordered by increasing $l_j$ with the increase in number.

Let's consider the following recursive function $f$, whose argument is an ordered set of tasks $A$, and the result is an integer. The function $f(A)$ is defined by the greedy algorithm, which is described below in a pseudo-language of programming.

- Step 1. $B = \varnothing$, $ans = 0$.
- Step 2. We consider all tasks in the order of increasing of their numbers in the set $A$. Lets define the current task counter $i = 0$.
- Step 3. Consider the next task: $i = i + 1$. If $i > |A|$ fulfilled, then go to the 8 step.
- Step 4. If you can get the task done starting at time $s_i$ = max$(ans + 1, l_i)$, then do the task $i$: $s_i$ = max$(ans + 1, l_i)$, $ans = s_i + t_i - 1$, $B = B \cup a_i$. Go to the next task (step 3).
- Step 5. Otherwise, find such task $b_k \in B$, that first, task $a_i$ can be done at time $s_i$ = max$\left(f(B \setminus b_k) + 1, l_i\right)$, and secondly, the value of $ans - f((B \cup a_i) \setminus b_k)$ is positive and takes the maximum value among all $b_k$ that satisfy the first condition. If you can choose multiple tasks as $b_k$, choose the one with the maximum number in set $A$.
- Step 6. If you managed to choose task $b_k$, then $ans = f((B \cup a_i) \setminus b_k)$, $B = (B \cup a_i) \setminus b_k$. Go to the next task (step 3).
- Step 7. If you didn't manage to choose task $b_k$, then skip task $i$. Go to the next task (step 3).
- Step 8. Return $ans$ as a result of executing $f(A)$.

Polycarpus got entangled in all these formulas and definitions, so he asked you to simulate the execution of the function $f$, calculate the value of $f(A)$.

## Input

The first line of the input contains a single integer $n$ $(1 \le n \le 10^5)$ — the number of tasks in set $A$.

Then $n$ lines describe the tasks. The $i$-th line contains three space-separated integers $l_i$, $r_i$, $t_i$ $(1 \le l_i \le r_i \le 10^9,\ 1 \le t_i \le r_i - l_i + 1)$ — the description of the $i$-th task.

It is guaranteed that for any tasks $j, k$ (considering that $j < k$) the following is true: $l_j < l_k$ and $r_j < r_k$.

## Output

For each task $i$ print a single integer — the result of processing task $i$ on the $i$-th iteration of the cycle (step 3) in function $f(A)$. In the $i$-th line print:

- 0 — if you managed to add task $i$ on step 4.
- -1 — if you didn't manage to add or replace task $i$ (step 7).
- $res_i$ $(1 \le res_i \le n)$ — if you managed to replace the task (step 6): $res_i$ equals the task number (in set $A$), that should be chosen as $b_k$ and replaced by task $a_i$.

**Sample test(s)**

input

```
5
1 8 5
2 9 3
3 10 3
8 11 4
11 12 2
```

output

```
0 0 1 0 -1
```

input

```
13
1 8 5
2 9 4
3 10 1
4 11 3
8 12 5
9 13 5
10 14 5
11 15 1
12 16 1
13 17 1
14 18 3
```

```
15 19 3
16 20 2
```

output

```
0 0 0 2 -1 -1 0 0 0 0 7 0 12
```