Codeforces Round #363 (Div. 1)

A. Vacations

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Vasya has n days of vacations! So he decided to improve his IT skills and do sport. Vasya knows the following information about each of this n days: whether that gym opened and whether a contest was carried out in the Internet on that day. For the i-th day there are four options:

1. on this day the gym is closed and the contest is not carried out;
2. on this day the gym is closed and the contest is carried out;
3. on this day the gym is open and the contest is not carried out;
4. on this day the gym is open and the contest is carried out.

On each of days Vasya can either have a rest or write the contest (if it is carried out on this day), or do sport (if the gym is open on this day).

Find the minimum number of days on which Vasya will have a rest (it means, he will not do sport and write the contest at the same time). The only limitation that Vasya has — *he does not want to do the same activity on two consecutive days: it means, he will not do sport on two consecutive days, and write the contest on two consecutive days.*

Input

The first line contains a positive integer n $(1 \le n \le 100)$ — the number of days of Vasya's vacations.

The second line contains the sequence of integers $a_1, a_2, ..., a_n$ $(0 \le a_i \le 3)$ separated by space, where:

- a_i equals 0, if on the i-th day of vacations the gym is closed and the contest is not carried out;
- a_i equals 1, if on the i-th day of vacations the gym is closed, but the contest is carried out;
- a_i equals 2, if on the i-th day of vacations the gym is open and the contest is not carried out;
- a_i equals 3, if on the i-th day of vacations the gym is open and the contest is carried out.

Output

Print the minimum possible number of days on which Vasya will have a rest. Remember that Vasya refuses:

- to do sport on any two consecutive days,
- to write the contest on any two consecutive days.

Examples

input
4 1 3 2 0
output
2

input
7 1 3 3 2 1 2 3
output
0

input
2 2 2
output
1

Note

In the first test Vasya can write the contest on the day number 1 and do sport on the day number 3. Thus, he will have a rest for only 2 days.

In the second test Vasya should write contests on days number 1, 3, 5 and 7, in other days do sport. Thus, he will not have a rest for a single day.

In the third test Vasya can do sport either on a day number 1 or number 2. He can not do sport in two days, because it will be contrary to the his limitation. Thus, he will have a rest for only one day.

B. Fix a Tree

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

A tree is an undirected connected graph without cycles.

Let's consider a rooted undirected tree with n vertices, numbered 1 through n. There are many ways to represent such a tree. One way is to create an array with n integers $p_1, p_2, ..., p_n$, where p_i denotes a parent of vertex i (here, for convenience a root is considered its own parent).

For this rooted tree the array p is $[2, 3, 3, 2]$.

Given a sequence $p_1, p_2, ..., p_n$, one is able to restore a tree:

1. There must be exactly one index r that $p_r = r$. A vertex r is a root of the tree.
2. For all other n - 1 vertices i, there is an edge between vertex i and vertex p_i.

A sequence $p_1, p_2, ..., p_n$ is called valid if the described procedure generates some (any) rooted tree. For example, for $n = 3$ sequences `(1,2,2)`, `(2,3,1)` and `(2,1,3)` **are not** valid.

You are given a sequence $a_1, a_2, ..., a_n$, not necessarily valid. Your task is to change the minimum number of elements, in order to get a valid sequence. Print the minimum number of changes and an example of a valid sequence after that number of changes. If there are many valid sequences achievable in the minimum number of changes, print any of them.

Input

The first line of the input contains an integer n ($2 \le n \le 200\,000$) — the number of vertices in the tree.

The second line contains n integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le n$).

Output

In the first line print the minimum number of elements to change, in order to get a valid sequence.

In the second line, print any valid sequence possible to get from $(a_1, a_2, ..., a_n)$ in the minimum number of changes. If there are many such sequences, any of them will be accepted.

Examples

input
4 2 3 3 4
output
1 2 3 4 4

input
5 3 2 2 5 3
output
0 3 2 2 5 3

input
8 2 3 5 4 1 6 6 7
output
2 2 3 7 8 1 6 6 7

Note

In the first sample, it's enough to change one element. In the provided output, a sequence represents a tree rooted in a vertex 4 (because $p_4 = 4$), which you can see on the left drawing below. One of other correct solutions would be a sequence `2 3 3 2`, representing a tree rooted in vertex 3 (right drawing below). On both drawings, roots are painted red.

In the second sample, the given sequence is already valid.

C. LRU

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

While creating high loaded systems one should pay a special attention to caching. This problem will be about one of the most popular caching algorithms called LRU (Least Recently Used).

Suppose the cache may store no more than k objects. At the beginning of the workflow the cache is empty. When some object is queried we check if it is present in the cache and move it here if it's not. If there are more than k objects in the cache after this, the least recently used one should be removed. In other words, we remove the object that has the smallest time of the last query.

Consider there are n videos being stored on the server, all of the same size. Cache can store no more than k videos and caching algorithm described above is applied. We know that any time a user enters the server he pick the video i with probability p_i. The choice of the video is independent to any events before.

The goal of this problem is to count for each of the videos the probability it will be present in the cache after 10^{100} queries.

Input

The first line of the input contains two integers n and k ($1 \le k \le n \le 20$) — the number of videos and the size of the cache respectively. Next line contains n real numbers p_i ($0 \le p_i \le 1$), each of them is given with no more than two digits after decimal point.

It's guaranteed that the sum of all p_i is equal to 1.

Output

Print n real numbers, the i-th of them should be equal to the probability that the i-th video will be present in the cache after 10^{100} queries. You answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Namely: let's assume that your answer is a, and the answer of the jury is b. The checker program will consider your answer correct, if .

Examples

input
3 1 0.3 0.2 0.5
output
0.3 0.2 0.5

input
2 1 0.0 1.0
output
0.0 1.0

input
3 2 0.3 0.2 0.5
output
0.675 0.4857142857142857 0.8392857142857143

input
3 3 0.2 0.3 0.5
output
1.0 1.0 1.0

D. Limak and Shooting Points

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

Bearland is a dangerous place. Limak can't travel on foot. Instead, he has k magic teleportation stones. Each stone can be used **at most once**. The i-th stone allows to teleport to a point (ax_i, ay_i). Limak can use stones **in any order**.

There are n monsters in Bearland. The i-th of them stands at (mx_i, my_i).

The given $k + n$ points are pairwise distinct.

After each teleportation, Limak can shoot an arrow in some direction. An arrow will hit the first monster in the chosen direction. Then, both an arrow and a monster disappear. It's dangerous to stay in one place for long, so Limak can shoot only one arrow from one place.

A monster should be afraid if it's possible that Limak will hit it. How many monsters should be afraid of Limak?

Input

The first line of the input contains two integers k and n ($1 \le k \le 7$, $1 \le n \le 1000$) — the number of stones and the number of monsters.

The i-th of following k lines contains two integers ax_i and ay_i ($-10^9 \le ax_i, ay_i \le 10^9$) — coordinates to which Limak can teleport using the i-th stone.

The i-th of last n lines contains two integers mx_i and my_i ($-10^9 \le mx_i, my_i \le 10^9$) — coordinates of the i-th monster.

The given $k + n$ points are pairwise distinct.

Output

Print the number of monsters which should be afraid of Limak.

Examples

input

```
2 4
-2 -1
4 5
4 2
2 1
4 -1
1 -1
```

output

```
3
```

input

```
3 8
10 20
0 0
20 40
300 600
30 60
170 340
50 100
28 56
90 180
-4 -8
-1 -2
```

output

```
5
```

Note

In the first sample, there are two stones and four monsters. Stones allow to teleport to points $(-2, -1)$ and $(4, 5)$, marked blue in the drawing below. Monsters are at $(4, 2)$, $(2, 1)$, $(4, -1)$ and $(1, -1)$, marked red. A monster at $(4, -1)$ shouldn't be afraid because it's impossible that Limak will hit it with an arrow. Other three monsters can be hit and thus the answer is 3.

In the second sample, five monsters should be afraid. Safe monsters are those at $(300, 600)$, $(170, 340)$ and $(90, 180)$.

E. Cron

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

Sometime the classic solution are not powerful enough and we have to design our own. For the purpose of this problem you have to implement the part of the system of task scheduling.

Each task should be executed at some particular moments of time. In our system you may set the exact value for the second, minute, hour, day of the week, day and month, when the task should be executed. Moreover, one can set a special value `-1` that means any value of this parameter is valid.

For example, if the parameter string is `-1 59 23 -1 -1 -1`, the problem will be executed every day at 23:59:00, 23:59:01, 23:59:02, ..., 23:59:59 (60 times in total).

Seconds, minutes and hours are numbered starting from zero, while day, months and days of the week are numbered starting from one. The first day of the week is Monday.

There is one special case that is treated separately. If both day of the week and day are given (i.e. differ from `-1`) to execute the task only one of these two (at least one, if both match this is fine too) parameters should match the current time (of course, all other parameters should match too). For example, the string of parameters `0 0 12 6 3 7` means that the task will be executed both on Saturday, July 2nd, 2016 and on Sunday, July 3rd, 2016 at noon.

One should not forget about the existence of the leap years. The year is leap if it's number is divisible by 400, or is not divisible by 100, but is divisible by 4. Each leap year has 366 days instead of usual 365, by extending February to 29 days rather than the common 28.

The current time is represented as the number of seconds passed after 00:00:00 January 1st, 1970 (Thursday).

You are given the string of six parameters, describing the moments of time the task should be executed. You are also given a number of moments of time. For each of them you have to find the first moment of time strictly greater than the current when the task will be executed.

Input

The first line of the input contains six integers s, m, h, day, $date$ and $month$ ($0 \le s, m \le 59$, $0 \le h \le 23$, $1 \le day \le 7$, $1 \le date \le 31$, $1 \le month \le 12$). Each of the number can also be equal to -1. It's guaranteed, that there are infinitely many moments of time when this task should be executed.

Next line contains the only integer n ($1 \le n \le 1000$) — the number of moments of time you have to solve the problem for. Each of the next n lines contains a single integer t_i ($0 \le t_i \le 10^{12}$).

Output

Print n lines, the i-th of them should contain the first moment of time strictly greater than t_i, when the task should be executed.

Examples

input

```
-1 59 23 -1 -1 -1
6
1467372658
1467417540
1467417541
1467417598
1467417599
1467417600
```

output

```
1467417540
1467417541
1467417542
1467417599
1467503940
1467503940
```

input

```
0 0 12 6 3 7
3
1467372658
1467460810
1467547200
```

output

```
1467460800
1467547200
1468065600
```

Note

The moment of time 1467372658 after the midnight of January 1st, 1970 is 11:30:58 July 1st, 2016.

F. Coprime Permutation

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

Two positive integers are coprime if and only if they don't have a common divisor greater than 1.

Some bear doesn't want to tell Radewoosh how to solve some algorithmic problem. So, Radewoosh is going to break into that bear's safe with solutions. To pass through the door, he must enter a permutation of numbers 1 through n. The door opens if and only if an entered permutation $p_1, p_2, ..., p_n$ satisfies:

In other words, two different elements are coprime if and only if their indices are coprime.

Some elements of a permutation may be already fixed. In how many ways can Radewoosh fill the remaining gaps so that the door will open? Print the answer modulo $10^9 + 7$.

Input

The first line of the input contains one integer n ($2 \le n \le 1\,000\,000$).

The second line contains n integers $p_1, p_2, ..., p_n$ ($0 \le p_i \le n$) where $p_i = 0$ means a gap to fill, and $p_i \ge 1$ means a fixed number.

It's guaranteed that if $i \neq j$ and $p_i, p_j \ge 1$ then $p_i \neq p_j$.

Output

Print the number of ways to fill the gaps modulo $10^9 + 7$ (i.e. modulo 1000000007).

Examples

input
4 0 0 0 0
output
4

input
5 0 0 1 2 0
output
2

input
6 0 0 1 2 0 0
output
0

input
5 5 3 4 2 1
output
0

Note

In the first sample test, none of four element is fixed. There are four permutations satisfying the given conditions: (1,2,3,4), (1,4,3,2), (3,2,1,4), (3,4,1,2).

In the second sample test, there must be $p_3 = 1$ and $p_4 = 2$. The two permutations satisfying the conditions are: (3,4,1,2,5), (5,4,1,2,3).