A. Login Verification

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

When registering in a social network, users are allowed to create their own convenient login to make it easier to share contacts, print it on business cards, etc.

Login is an arbitrary sequence of lower and uppercase latin letters, digits and underline symbols («_»). However, in order to decrease the number of frauds and user-inattention related issues, it is prohibited to register a login if it is *similar* with an already existing login. More precisely, two logins s and t are considered similar if we can transform s to t via a sequence of operations of the following types:

- transform lowercase letters to uppercase and vice versa;
- change letter «O» (uppercase latin letter) to digit «0» and vice versa;
- change digit «1» (one) to any letter among «l» (lowercase latin «L»), «I» (uppercase latin «i») and vice versa, or change one of these letters to other.

For example, logins «Codeforces» and «codef0rces» as well as «OOOOOOOOOOOOOOOOOOOOOO_lol» and «OOOOOOOOOOOOOOOOOOOOOO_loI» are considered similar whereas «Codeforces» and «Code_forces» are not.

You're given a list of existing logins with no two *similar* amonst and a newly created user login. Check whether this new login is *similar* with any of the existing ones.

Input

The first line contains a non-empty string s consisting of lower and uppercase latin letters, digits and underline symbols («_») with length not exceeding 50 — the login itself.

The second line contains a single integer n ($1 \le n \le 1\,000$) — the number of existing logins.

The next n lines describe the existing logins, following the same constraints as the user login (refer to the first line of the input). It's guaranteed that no two existing logins are similar.

Output

Print «Yes» (without quotes), if user can register via this login, i.e. none of the existing logins is *similar* with it.

Otherwise print «No» (without quotes).

Examples

input
1_wat 2 2_wat wat_1
output
Yes

input
000 3 00 ooA oOo
output
No

input
_i_ 3 __i_ _1_ I
output
No

input
La0 3 2a0 La1 1a0
output
No

input
abc 1 aBc
output
No

input
0Lil 2 LIL0 0Ril
output
Yes

Note

In the second sample case the user wants to create a login consisting of three zeros. It's impossible due to collision with the third among the existing.

In the third sample case the new login is similar with the second one.

B. Chat

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

There are times you recall a good old friend and everything you've come through together. Luckily there are social networks — they store all your message history making it easy to know what you argued over 10 years ago.

More formal, your message history is a sequence of messages ordered by time sent numbered from 1 to n where n is the total number of messages in the chat.

Each message might contain a link to an earlier message which it is a reply to. When opening a message x or getting a link to it, the dialogue is shown in such a way that k previous messages, message x and k next messages are visible (with respect to message x). In case there are less than k messages somewhere, they are yet all shown.

Digging deep into your message history, you always read all visible messages and then go by the link in the current message x (if there is one) and continue reading in the same manner.

Determine the number of messages you'll read if your start from message number t for all t from 1 to n. Calculate these numbers independently. If you start with message x, the initial configuration is x itself, k previous and k next messages. Messages read multiple times are considered as one.

Input

The first line contains two integers n and k ($1 \le n \le 10^5$, $0 \le k \le n$) — the total amount of messages and the number of previous and next messages visible.

The second line features a sequence of integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < i$), where a_i denotes the i-th message link destination or zero, if there's no link from i. All messages are listed in chronological order. It's guaranteed that the link from message x goes to message with number strictly less than x.

Output

Print n integers with i-th denoting the number of distinct messages you can read starting from message i and traversing the links while possible.

Examples

input
6 0 0 1 1 2 3 2
output
1 2 2 3 3 3

input
10 1 0 1 0 3 4 5 2 3 7 0
output
2 3 3 4 5 6 6 6 8 2

input
2 2 0 1
output
2 2

Note

Consider $i = 6$ in sample case one. You will read message 6, then 2, then 1 and then there will be no link to go.

In the second sample case $i = 6$ gives you messages $5, 6, 7$ since $k = 1$, then $4, 5, 6$, then $2, 3, 4$ and then the link sequence breaks. The number of distinct messages here is equal to 6.

C. Dependency management

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Polycarp is currently developing a project in Vaja language and using a popular dependency management system called Vamen. From Vamen's point of view both Vaja project and libraries are treated projects for simplicity.

A project in Vaja has its own uniqie non-empty name consisting of lowercase latin letters with length not exceeding 10 and version — positive integer from 1 to 10^6. Each project (keep in mind that it is determined by both its name and version) might depend on other projects. For sure, there are no cyclic dependencies.

You're given a list of project descriptions. **The first** of the given projects is the one being developed by Polycarp at this moment. Help Polycarp determine all projects that his project depends on (directly or via a certain chain).

It's possible that Polycarp's project depends on two different versions of some project. In this case collision resolving is applied, i.e. for each such project the system chooses the version that minimizes the distance from it to Polycarp's project. If there are several options, the newer (with the maximum version) is preferred. This version is considered actual; **other versions and their dependencies are ignored.**

More formal, choose such a set of projects of minimum possible size that the following conditions hold:

- Polycarp's project is chosen;
- Polycarp's project depends (directly or indirectly) on all other projects in the set;
- no two projects share the name;
- for each project x that some other project in the set depends on we have either x or some y with other version and shorter chain to Polycarp's project chosen. In case of ties the newer one is chosen.

Output all Polycarp's project's dependencies (Polycarp's project itself should't be printed) in lexicographical order.

Input

The first line contains an only integer n ($1 \le n \le 1\,000$) — the number of projects in Vaja.

The following lines contain the project descriptions. Each project is described by a line consisting of its name and version separated by space. The next line gives the number of direct dependencies (from 0 to n - 1) and the dependencies themselves (one in a line) in arbitrary order. Each dependency is specified by its name and version. The projects are also given in arbitrary order, but the first of them is always Polycarp's. Project descriptions are separated by one empty line. Refer to samples for better understanding.

It's guaranteed that there are no cyclic dependencies.

Output

Output all Polycarp's project's dependencies in lexicographical order.

Examples

input
4 a 3 2 b 1 c 1 b 2 0 b 1

```
1
b 2

c 1
1
b 2
```

output

```
2
b 1
c 1
```

input

```
9
codehorses 5
3
webfrmk 6
mashadb 1
mashadb 2

commons 2
0

mashadb 3
0

webfrmk 6
2
mashadb 3
commons 2

extra 4
1
extra 3

extra 3
0

extra 1
0

mashadb 1
1
extra 3

mashadb 2
1
extra 1
```

output

```
4
commons 2
extra 1
mashadb 2
webfrmk 6
```

input

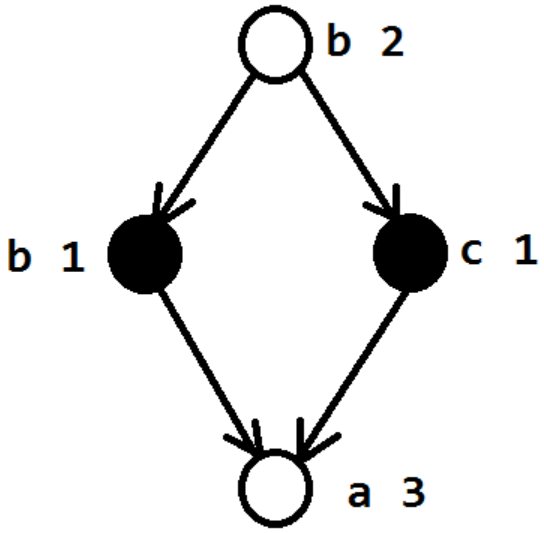
```
3
abc 1
2
abc 3
cba 2

abc 3
0

cba 2
0
```

output

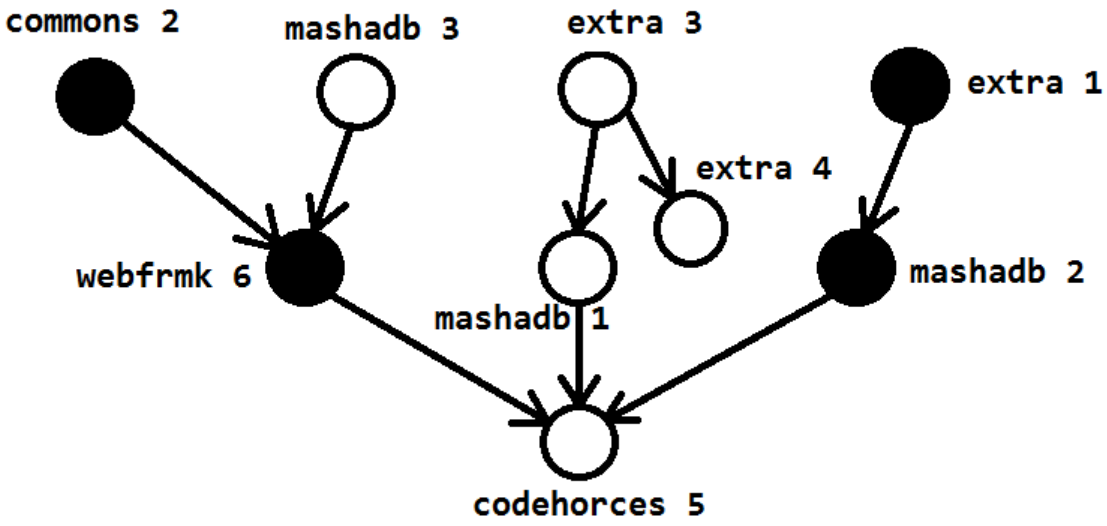
```
1
cba 2
```

Note

The first sample is given in the pic below. Arrow from A to B means that B directly depends on A. Projects that Polycarp's project «a» (version 3) depends on are painted black.



The second sample is again given in the pic below. Arrow from A to B means that B directly depends on A. Projects that Polycarp's project «codehorses» (version 5) depends on are paint it black. Note that «extra 1» is chosen instead of «extra 3» since «mashadb 1» and all of its dependencies are ignored due to «mashadb 2».



D. Autocompletion

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

Arcady is a copywriter. His today's task is to type up an already well-designed story using his favorite text editor.

Arcady types words, punctuation signs and spaces one after another. Each letter and each sign (including line feed) requires one keyboard click in order to be printed. Moreover, when Arcady has a non-empty prefix of some word on the screen, the editor proposes a possible autocompletion for this word, more precisely one of the already printed words such that its prefix matches the currently printed prefix if this word is unique. For example, if Arcady has already printed «codeforces», «coding» and «codeforces» once again, then there will be no autocompletion attempt for «cod», but if he proceeds with «code», the editor will propose «codeforces».

With a single click Arcady can follow the editor's proposal, i.e. to transform the current prefix to it. Note that no additional symbols are printed after the autocompletion (no spaces, line feeds, etc). What is the minimum number of keyboard clicks Arcady has to perform to print the entire text, if he is not allowed to move the cursor or erase the already printed symbols?

A word here is a contiguous sequence of latin letters bordered by spaces, punctuation signs and line/text beginnings/ends. Arcady uses only lowercase letters. For example, there are 20 words in «it's well-known that tic-tac-toe is a paper-and-pencil game for two players, x and o.».

Input

The only line contains Arcady's text, consisting only of lowercase latin letters, spaces, line feeds and the following punctuation signs: «.», «,», «?», «!», «'» and «-». The total amount of symbols doesn't exceed $3 \cdot 10^5$. It's guaranteed that all lines are non-empty.

Output

Print a single integer — the minimum number of clicks.

Examples

input
snow affects sports such as skiing, snowboarding, and snowmachine travel. snowboarding is a recreational activity and olympic and paralympic sport.
output
141

input
'co-co-co, codeforces?!'
output
25

input
thun-thun-thunder, thunder, thunder thunder, thun-, thunder thun-thun-thunder, thunder thunder, feel the thunder lightning then the thunder thunder, feel the thunder lightning then the thunder thunder, thunder
output
183

Note

In sample case one it's optimal to use autocompletion for the first instance of «`snowboarding`» after typing up «`sn`» and for the second instance of «`snowboarding`» after typing up «`snowb`». This will save 7 clicks.

In sample case two it doesn't matter whether to use autocompletion or not.