**Jumps**

Given a sequence of elements(numbers), calculate the longest possible sequence of 'jumps' from each number.

Each 'jump' must be made according to the following rules:

* You can only 'jump' on a number that is greater than the current one;
* You can 'jump' on a number, only if there isn't one with a greater value between;
* You can 'jump' only from left to right;

**Input**

Read from the standard input

* On the first line, you will find the number N
  + The number of elements
* On the second line you will find N numbers, separated by a space
  + The elements themselves

The input will be correct and in the described format, so there is no need to check it explicitly.

**Output**

Print to the standard output

* On the first line, print the length of the longest sequence of jumps
* On the second line, print the lengths of the sequences, starting from each element

**Constraints**

* The N will always be less than 103344

**Sample Tests**

**Input**

6

1 4 2 6 3 4

**Output**

2

2 1 1 0 1 0

**Explanation**

* Element 1:
  + 1 -> 4 -> 6 (2 jumps)
* Element 2:
  + 4 -> 6 (1 jump)
* Element 3:
  + 2 -> 6 (1 jump)
* Element 4:
  + 6 (0 jumps)
* Element 5:
  + 3 -> 4 (1 jump)
* Element 6:
  + 4 -> (0 jumps)

**Input**

5

1 1 1 1 1

**Output**

0

0 0 0 0 0

**Students order**

Alpha students love learning new stuff. They are also aware that switching seats in the classroom helps them learn new material more effectively. You are given the names of **N** students and **K** seat changes. Your task is to implement an algorithm that shows the students' final seating order after all seat changes have been made.

**Input**

* Read from the standard input.
* On the first line, you'll find the numbers **N** and **K** separated by a white space.
  + **N** - the number of students.
  + **K** - the number of seat changes.
* On the next line there will be **N** student names separated by a white space.
* The next **K** lines are pairs of student names. Each pair represents a change of seats where the first student sits on the left of the second student.
  + The names are separated by a white space.

**Output**

* Print on the standard output.
* On a single line, print the final order of the student names.

**Constraints**

* 1 <= N <= 2000
* 1 <= K <= 100 000
* Each name contains only alphanumeric characters.
* All names are unique.

**Sample tests**

**Input**

5 3

Gosho Tosho Penka Miro Stanka

Miro Gosho

Gosho Stanka

Stanka Miro

**Output**

Stanka Miro Tosho Penka Gosho

**Explanation**

* **Miro** takes a seat next to **Gosho**. The order becomes **Miro Gosho Tosho Penka Stanka**
* **Gosho** takes a seat next to **Stanka**. The order becomes **Miro Tosho Penka Gosho Stanka**
* **Stanka** takes a seat next to **Miro**. The order becomes **Stanka Miro Tosho Penka Gosho**

**Input**

7 4

Emo Misho Ivanka Ginka Vancho Stancho Sashka

Emo Misho

Misho Emo

Misho Sashka

Sashka Stancho

**Output**

Emo Ivanka Ginka Vancho Sashka Stancho Misho

**HDNL Toy**

Steve found a new toy to play with. It's called HDNL (High Definition Native Language). He doesn't know what it is used for, he just finds it interesting. HDNL works by *defining homeomorphic endofunctors mapping submanifolds of a Hilbert space*. Sadly, when Steve is looking at HDNL, he isn't always able to imagine how all it would look in the end. Each line of HDNL consists of **a letter and a number** and opens a tag (like HTML tag). The letter is important, though Steve can't remember why. The number defines the **level of nesting**. Steve wants to see how he can nest all the tags such that the **level of nesting** of inner tags is **bigger** than that of outer tags. Your task is to write a program for Steve which shows nicely indented and closed HDNL tags.

**Input**

* On the first line of input, a number **N** is read - the number of HDNL lines to follow
* Each of the next **N** lines will be a Latin letter glued to positive number

**Output**

* There should be **N** \* 2 lines
* Each output line should contain either an opening or a closing tag
* Use 1 space for indentation

**Constraints**

* 1 <= **N** <= 100 000
* 1 <= **level of nesting** <= 1000

**Sample tests**

**Input**

4

h1

r5

d2

a0

**Output**

<h1>

<r5>

</r5>

<d2>

</d2>

</h1>

<a0>

</a0>

**Input**

9

a1

b2

c3

d3

e2

f3

g2

h1

i2

**Output**

<a1>

<b2>

<c3>

</c3>

<d3>

</d3>

</b2>

<e2>

<f3>

</f3>

</e2>

<g2>

</g2>

</a1>

<h1>

<i2>

</i2>

</h1>