# Croatian Open Competition in Informatics

Round 2, November 13th 2021

## Tasks

| Task | Time limit | Memory limit | Points |
|---|---|---|---|
| **Kaučuk** | 1 second | 512 MiB | 50 |
| **Kutije** | 1 second | 512 MiB | 70 |
| **Hiperkocka** | 1 second | 512 MiB | 110 |
| **Magneti** | 1 second | 512 MiB | 110 |
| **Osumnjičeni** | 1 second | 512 MiB | 110 |
| **Total** | | | 450 |

# Task Kaučuk

Davor the scientist writes his papers in LaTeX[1]. Inspired by LaTeX, he came up with *Kaučuk*. Kaučuk is a very simple program for preparing a piece of text for print. It allows you to number the headings of sections, subsections and subsubsections and prepare them for print.

Kaučuk has only three different commands:

- The `section` command starts a new section. All sections from the input are numbered in the output with positive integers starting from 1, in the order they appear in the input (see first example).
- The `subsection` command starts a new subsection. Subsections are numbered using two numbers: the number of the section which contains the subsection, and the number of the subsection within the section (see second example). In each section the subsection numbering starts again from 1 (see third example).
- The `subsubsection` command starts a new subsubsection which uses three numbers: the number of the section, the number of the subsection and the number of the subsubsection within the subsection, in a manner similar to subsections within sections (see second example).

It is guaranteed that in the input, each subsection is contained in some section, and each subsubsection is contained in some subsection. Davor might be a fine scientist, but programming is not his strong suit, which is why he is asking you to help him write a program which prepares a text written in Kaučuk for print.

## Input

The first line contains a positive integer $n$ ($1 \leq n \leq 100$), the number of lines of Kaučuk commands.

The following $n$ lines contain Davor's Kaučuk code. Each line of code is made up from two strings of characters, separated by a single space: the type of section (`section`, `subsection` or `subsubsection`) and its title. Each title is made up from at most 20 lowercase letters.

## Output

In $n$ lines you should number and print the titles of the sections, subsections and subsubsections from Davor's code.

## Scoring

| Subtask | Points | Constraints |
|---|---|---|
| 1 | 10 | $1 \leq n \leq 3$ |
| 2 | 10 | The Kaučuk code will contain only the `section` command. |
| 3 | 10 | The Kaučuk code will contain only the `section` and `subsection` commands. |
| 4 | 20 | No additional constraints. |

[1] a text program used for preparing documents for print

## Examples

**input**

```
3
section zivotinje
section boje
section voce
```

**output**

```
1 zivotinje
2 boje
3 voce
```

**input**

```
4
section zivotinje
subsection macke
subsection psi
subsubsection mops
```

**output**

```
1 zivotinje
1.1 macke
1.2 psi
1.2.1 mops
```

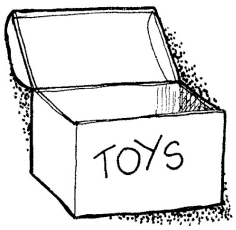
**input**

```
4
section zivotinje
subsection psi
section voce
subsection ananas
```

**output**

```
1 zivotinje
1.1 psi
2 voce
2.1 ananas
```

# Task Kutije




Martin has $n$ boxes labeled with positive integers from 1 to $n$. Each box contains a toy. The toys are also labeled with positive integers from 1 to $n$ and in such a way that initially the toy with label $i$ is contained in the box with label $i$.

From time to time, Martin calls one of his $m$ friends to come over and hang out. Once they meet up, his friend takes the toys out of the boxes and starts having fun with them. In the meantime, Martin is more interested in the boxes. Once they become bored, his friend puts the toys back into the boxes. However, he doesn't necessarily put every toy in the box it was taken from.

Martin has noticed that each of his $m$ friends scrambles the toys in the same way each time. More precisely, each of his friends has his own array of $n$ positive integers $p_1, ..., p_n$ which determines the way he will put the toys back into the boxes. Every positive integer from 1 to $n$ appears exactly once in this array. His friend scrambles the toys in such a way that at the end of their meeting the box with label $i$ contains the toy that was in the box with label $p_i$ at the start of their meeting. Notice that, because every positive integer from 1 to $n$ appears exactly once in the array, after all the toys are back in the boxes, each box will again have exactly one toy in it.

Martin is now interested in answering questions of the following form: he is wondering whether it is possible that the toy with label $a$ (which is initially in box with label $a$) can end up in the box with label $b$ via a sequence of meetups with his friends. A sequence of meetups means that Martin can call whichever friends he wants and in any order. He can call a friend multiple times, or not at all. Martin is interested in answering $q$ such questions.

## Input

The first line contains positive integers $n$, $m$ and $q$ - the number of boxes (also toys), the number of Martin's friends and the number of questions, respectively.

The $k$-th of the following $m$ lines contains an array of positive integers $p_1, ..., p_n$ that is used by Martin's $k$-th friend for putting the toys back into the boxes. Each positive integer from 1 to $n$ appears exactly once in the array.

Each of the following $q$ lines contains two positive integers $a$ and $b$ ($1 \leq a, b \leq n$) which represent a question.

## Output

In $q$ lines print the answer to the given qustions, in order: `DA` if it is possible to get the toy in question to the desired box, and `NE` otherwise.

## Scoring

In every subtask, it holds that $1 \leq n, m \leq 1000$, $1 \leq q \leq 500\,000$.

| Subtask | Points | Constraints |
|---|---|---|
| 1 | 15 | $m = 1$ |
| 2 | 10 | $1 \leq n, m, q \leq 100$. Additionally, for each question for which the answer is `DA`, there exists a sequence of at most two meetups which achieves the desired result. |
| 3 | 10 | $1 \leq n, m, q \leq 100$ |
| 4 | 35 | No additional constraints. |

## Examples

**input**

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

**output**

```
DA
NE
DA
```

**input**

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

**output**

```
DA
DA
NE
NE
```

**input**

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

**output**

```
DA
NE
```

**Clarification of the first example:**

For the first question, the toy with label 1 is already initially in the box with label 1 so the answer is immediately `DA`.

For the second question, notice that however many times Martin calls his friend over, the boxes with labels 1 and 2 never change their content, so the answer is `NE`.

For the third question, notice that after each meetup, the contents of boxes 3 and 4 get exchanged, so after only one meetup the toy with label 3 will find itself in box with label 4 and the answer is `DA`.

# Task Hiperkocka

*...it's dark in the cube, it's dark in the cube...*

Five in the morning. Daniel wakes up, he opens his eyes. His head hurts a bit. He can still hear the ringing in his ears.

He comes to realize that he has found himself at a playground, in a big metal box.

*...I was in the cube, I was in the cube...*

He remembers a similar situation he found himself in, three years ago, COCI round 2, task Kocka.

*...I'm in the cube again, I'm in the cube again...*

But this time, things are much more complicated... Daniel is in an $n$-dimensional hipercube $\mathcal{Q}_n$. $2^{n-1}$ identical copies of a tree $\mathcal{T}$ with $n$ edges are scattered around him. It soon became clear to him that salvation lies in tiling the edges of the hipercube with the trees.

Formally, a *hipercube* $\mathcal{Q}_n$ is a graph with nodes $0, 1, \ldots 2^n - 1$, in which nodes $x$ and $y$ are connected if and only if their bitwise *xor* is a power of two.

A tree can be *placed* on the hipercube so that:

- each node of the tree corresponds to some node of the hipercube
- those nodes have to be distinct
- if there is an edge between two nodes in the tree, then there has to be an edge between the corresponding nodes in the hipercube.

A *tiling* of the hipercube is done by placing several trees so that each edge of the hipercube belongs to at most one tree.

Your task is to tile the hipercube $\mathcal{Q}_n$ with as many copies of the given tree $\mathcal{T}$, which has $n$ edges.

## Input

The first line contains a positive integer $n$ ($1 \leq n \leq 16$), the dimension of the hipercube.

Each of the following $n$ lines contains two integers $x$ and $y$ ($0 \leq x, y \leq n$, $x \neq y$) which denote that the nodes $x$ and $y$ are connected by an edge in tree $\mathcal{T}$.

## Output

In the first line print the number of trees in your tiling.

Each of the following lines should describe a placement of a single copy of the tree $\mathcal{T}$.

In the $i$-th line print $n+1$ numbers $a_0^{(i)}, a_1^{(i)} \ldots a_n^{(i)}$. These numbers denote that the $i$-th tree is placed so that the hipercube node $a_j^{(i)}$ corresponds to the tree node $j$, for all $j = 0, \ldots, n$.

## Scoring

If your solution correctly places $k$ trees, you will receive $f(k) \cdot 110$ points for that test case, where

$$f(k) = \begin{cases} 0.7 \cdot k/2^{n-1} & \text{if } k < 2^{n-1} \\ 1 & \text{if } k = 2^{n-1}. \end{cases}$$

Of course, if your solution is not correct, you will receive 0 points.

Your total number of points is equal to the minimum number of points your solution receives over all of the test cases.

It is possible to prove that there always exists a solution which uses all of the $2^{n-1}$ trees.

## Examples

**input**

```
1
0 1
```

**output**

```
1
0 1
```

**input**

```
2
0 1
1 2
```

**output**

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

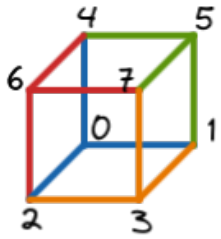
**input**

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

**output**

```
4
0 1 2 4
3 1 2 7
5 1 4 7
6 2 4 7
```

**Clarification of the third example:**



# Task Magneti

Little Marko is bored of playing with shady cryptocurrencies such as Shiba Inu or XRC, which is why he decided to play with magnets. He has $n$ different magnets and a board which has $l$ available empty slots in a row, in which the magnets can be placed. Each pair of adjacent slots is exactly one centimeter apart. Each of the $n$ magnets has a radius of activity that is equal to $r_i$. This means that it will attract all magnets that are located strictly less than $r_i$ centimeters away (regardless of the radius of activity of the other magnet). It is possible that some magnets have the same radius of activity, but they are considered as different magnets.

Marko doesn't like it when the magnets attract each other, so he is interested in the number of ways to place the magnets on the board so that no magnet attracts any other. All of the magnets should be placed on the board, and each empty slot may contain at most one magnet. Two ways of placing the magnets are considered different if there is magnet which is at a different position in the first way than in the second way. As the required number can be quite large, you should output it modulo $10^9 + 7$.

## Input

The first line contains positive integers $n$ and $l$, the number of magnets and the number of empty slots.

The second line contains $n$ positive integers $r_i$ ($1 \leq r_i \leq l$), the radii of activity of the $n$ magnets.

## Output

Print the required number of ways to place the magnets on the board so that no magnet attracts any other, modulo $10^9 + 7$.

## Scoring

In every subtask, it holds that $1 \leq n \leq 50$ and $n \leq l \leq 10\,000$.

| Subtask | Points | Constraints |
|---|---|---|
| 1 | 10 | $r_1 = r_2 = ... = r_n$ |
| 2 | 20 | $1 \leq n \leq 10$ |
| 3 | 30 | $1 \leq n \leq 30$, $n \leq l \leq 300$ |
| 4 | 50 | No additional constraints. |

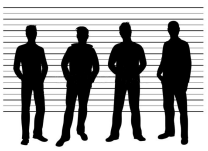
## Examples

| **input** | **input** | **input** |
|---|---|---|
| `1 10`<br>`10` | `4 4`<br>`1 1 1 1` | `3 4`<br>`1 2 1` |
| **output** | **output** | **output** |
| `10` | `24` | `4` |

**Clarification of the second example:**
All permutations of the magnets are valid because no two magnets can attract each other.

**Clarification of the third example:**
If we denote the magnets with `1`, `2` and `3`, and an empty slot with `_`, the possible arrangements of magnets are `13_2`, `31_2`, `2_13` and `2_31`.

# Task Osumnjičeni



In a police investigation, $n$ suspects were identified and now it's up to the witnesses to try to find the perpetrator. The height of every suspect $i$ was measured, but due to the unreliability of measurement, it is known only that their height is a real number from the interval from $l_i$ to $r_i$ (inclusive). At most one of the suspects is the perpetrator, and it could be the case that none of them are.

A signle *lineup* consists of choosing two positive integers $a$ and $b$ ($1 \leq a \leq b \leq n$), then taking the suspects $a$, $a+1$, ..., $b$ to a separate room so that the witnesses could try to identify the perpetrator. As the witnesses could be confused if two of the suspects have the same height, a *lineup* is allowed only if it is possible to guarantee that no two suspects will have the same height. During a *lineup*, the witnesses will always be able to identify the perpetrator if he is among the chosen suspects, or they will be able to tell that he is not among them.

The lead investigator is now interested in answering questions of the following form: "If I were certain that the label of the perpetrator could only be between $p$ and $q$ ($p \leq q$), what is the minimum number of *lineups* needed in the worst case so that the witnesses are able to find the perpetrator, or report that he is not among the suspects?" Help the lead investigator answer $q$ of such questions.

## Input

The first line contains a positive integer $n$, the number of suspects.

The following $n$ lines contain two positive integers $l_i$ and $r_i$ ($1 \leq l_i \leq r_i \leq 10^9$) which represent the possible height range of the suspect with label $i$.

The next line contains a positive integer $q$, the number of questions.

The following $q$ lines contain two positive integers $p_i$ and $q_i$ ($1 \leq p_i \leq q_i \leq n$) which determine a question.

## Output

In $q$ lines print the answers to the corresponding questions: the minimum required number of *lineups*.

## Scoring

In every subtask, it holds that $1 \leq n, q \leq 200\ 000$.

| Subtask | Points | Constraints |
|---|---|---|
| 1 | 10 | $q = 1$, $p_1 = 1$, $q_1 = n$ |
| 2 | 10 | $1 \leq n \leq 5000$, $1 \leq q \leq 5000$ |
| 3 | 20 | $1 \leq n \leq 5000$, $1 \leq q \leq 200\ 000$ |
| 4 | 20 | $1 \leq n \leq 200\ 000$, $1 \leq q \leq 100$ |
| 5 | 50 | No additional constraints. |

## Examples

**input**

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

**output**

```
1
1
2
```

**input**

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

**output**

```
1
1
1
```

**input**

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

**output**

```
3
1
3
```

**Clarification of the third example:**

For the first and the third question, it is sufficient to have three *lineups*: one consists of the suspect 1, one consists of the suspects 2 and 3, and one consists of the suspects 4 and 5.

For the second question, it is sufficient to have one *lineup* which consists of the suspects 3, 4 and 5.