# Problem A. ASCII Area

| | |
|---|---|
| Input file: | `ascii.in` |
| Output file: | `ascii.out` |

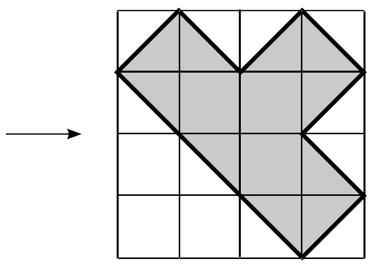
Long time ago, most of PCs were equipped with video cards that worked only in text mode. If the programmer wanted to show a picture on a screen, he had to use pseudographics or ASCII art like this:

```
^..^
(00)
/  \
()()
```

In this problem you are given a polygon, drawn using ASCII art. Your task is to calculate its area.

The picture is formed using characters '`.`', '`\`', and '`/`'. Each character represents a unit square of the picture. Character '`.`' represents an empty square, character '`/`' — a square with a segment from the lower left corner to the upper right corner, and character '`\`' — a square with a segment from the upper left corner to the lower right corner.

```
/\/\
\../
.\.\
..\/
```



## Input

The first line of the input file contains integer numbers $h$ and $w$ ($2 \le h, w \le 100$) — height and width of the picture. Next $h$ lines contain $w$ characters each — the picture drawn using ASCII art.

It is guaranteed that the picture contains exactly one polygon without self-intersections and self-touches.

## Output

Print to the output file one integer number — the area of the polygon.

## Sample input and output

| ascii.in | ascii.out |
|---|---|
| `4 4`<br>`/\/\`<br>`\../`<br>`.\.\`<br>`..\/` | `8` |

# Problem B. Binary Encoding

| | |
|---|---|
| Input file: | `binary.in` |
| Output file: | `binary.out` |

*Binary encoding* represents integer numbers from 0 to $2^n - 1$ inclusive with $n$ bits, each bit is either 0 or 1. In binary encoding most significant digit is written first. Two binary codes are compared from left to right, from the most significant digit to the least significant one. Binary codes are uniquely assigned to numbers from 0 to $2^n - 1$ is a such a way, that corresponding binary codes for numbers in ascending order are themselves in ascending order. For example, binary encoding for numbers from 0 to 7 is:

| | | | |
|---|---|---|---|
| 0 | 000 | 4 | 100 |
| 1 | 001 | 5 | 101 |
| 2 | 010 | 6 | 110 |
| 3 | 011 | 7 | 111 |

*Truncated binary encoding* generalizes binary encoding to represent integer numbers from 0 to $m - 1$ inclusive, where $m$ may or may not be equal to $2^n$ for some integer number $n$. Unlike binary encoding, truncated binary encoding uses different number of bits to represent different numbers. If $n$ is the smallest integer number such that $m \le 2^n$, then truncated binary encoding represents each number from 0 to $m-1$ with $n$ or with $n-1$ bits. Some numbers are encoded with $n-1$ bits only if $m < 2^n$; when $m = 2^n$ truncated binary encoding is the same as binary encoding. Truncated binary codes are compared from left to right, just like binary codes.

Truncated binary encoding also satisfies the following rules:

- Smaller integer numbers are represented with smaller or the same number of bits as larger ones.
- Truncated binary codes for numbers in ascending order are themselves in ascending order.
- Truncated binary codes for numbers from 0 to $m - 1$ are unique.
- No code with $n - 1$ bits is a prefix of any code with $n$ bits.
- The total number of bits used to represent numbers from 0 to $m - 1$ is minimal.

For example, truncated binary encoding for numbers from 0 to 5 is:

| | | | |
|---|---|---|---|
| 0 | 00 | 4 | 110 |
| 1 | 01 | 5 | 111 |
| 2 | 100 | | |
| 3 | 101 | | |

Your task is to encode numbers from 0 to $m - 1$ with truncated binary encoding.

## Input

The input file contains integer number $m$ ($2 \le m \le 10000$).

## Output

Write to the output file $m$ lines. These $m$ lines shall contain truncated binary encoding of numbers from 0 to $m - 1$ in ascending order.

## Sample input and output

| `binary.in` | `binary.out` |
|---|---|
| `6` | `00`<br>`01`<br>`100`<br>`101`<br>`110`<br>`111` |

## Problem C. Caption

| | |
|---|---|
| Input file: | `caption.in` |
| Output file: | `caption.out` |

Advanced Caption Machines (ACM) produces *electronic captions* that are used as labels, signs, and tags in various brick-and-mortar stores. They range from small tags that are used on the shelves of the stores to the large signs for the rows. Electronic captions use flip disks, electronic ink and other similar technologies to display one line of text so that this text can be electronically changed as needed. The common to all these display technologies that are used by ACM is that they represent a text with $m \times n$ grid of pixels that can be individually electronically turned on or off and indefinitely retain their state. For example, if turned off pixels are represented with '`.`' and turned on pixels are represented with '`*`', then one of the ways to display the text "ACM ICPC" on a $5 \times 53$ grid of pixels is:

```
.....*....****.*...*.........*....****.****...****...
....*.*..*.....**.**.........*...*.....*...*.*.......
...*...*.*.....*.*.*.........*...*.....****..*.......
...*****.*.....*...*.........*...*.....*.....*.......
...*...*..****.*...*.........*....****.*......****...
```

The advantage of an electronic caption is that energy is consumed only to flip the state of individual pixels. The total energy required to change displayed text to some other text is proportional to the number of pixels flipped.

ACM is mindful about nature conservation. The whole concept and marketing model of ACM's business is built around preservation of natural resources. Without ACM's captions stores had to print out new labels, signs, and tags whenever they had to change the layout of goods in the store, thus throwing old labels, signs, and tags away. ACM had decided to go even further and had figured out that each change of text on their electronic captions requires some electrical energy which should be conserved, too, because electrical energy is still mostly produced from non-renewable fossil fuels with their harmful emissions.

Fortunately, when one text is changed to the other text on an electronic caption, there is always a leeway in how the text can be laid out on $m \times n$ grid of pixels. The text is always written in a fixed-width font where each letter is represented by a $m \times k$ grid of pixels. However, the spacing between the letters in a caption can vary from $s_{min}$ pixels to $s_{max}$ pixels. The left-right position of the text in a caption can also vary. Together, that gives enough freedom to optimize the text update procedure, so that the number of pixels that need to change is minimized, thus minimizing the energy expenditure.

For example, the optimal way to change the text on the caption above to "NEERC" while maintaining spacing between the letters from 1 to 2 pixels is shown below. Only 61 pixels will have to be flipped (34 pixels will be turned off and 27 pixels will be turned on).

```
...................*...*..*****..*****.****...****...
...................**..*..*......*.....*...*.*.......
...................*.*.*..***....***...****..*.......
...................*..**..*......*.....*.*...*.......
...................*...*..*****..*****.*..*...****...
```

Your team is assigned with a task to write a procedure that finds the optimal caption layout for the new caption text given the text that it currently contains, so that the number of pixels to update is minimized.

## Input

The first line of the input file contains 5 integer numbers $m$, $n$, $k$, $s_{min}$, and $s_{max}$, where $m$ $(5 \le m \le 30)$ is the number of rows on the caption, $n$ $(5 \le n \le 2000)$ is the number of columns on the caption, $k$ $(5 \le k \le 30)$ is the width of each letter in the font, $s_{min}$ and $s_{max}$ $(0 \le s_{min} \le s_{max} \le 30)$ are the minimal and the maximal allowed spacing between letters in pixels correspondingly.

The following $m$ lines of the input file contain description of the font. Each line of the font description contains $t(k+3)-1$ characters, where $t$ $(1 \le t \le 26)$ is the number of Latin letters that are defined in this font. The grid with $m$ rows and $t(k+3)-1$ columns on those $m$ lines is composed of $m \times k$ grids of characters '`.`' and '`*`' defining the font for uppercase Latin letters from A to Z. The letters that are defined appear on the first line before the corresponding grids. Everything is arranged in the same way as in the sample input below. The first of those $m$ lines uses a total of $2t-1$ spaces as delimiters, subsequent lines use $3t-1$ spaces each. Letters do not necessary appear in alphabetic order, but each letter is defined at most once.

The space character is assumed to be implicitly defined in any font as $m \times k$ grid of '`.`'. The spacing between spaces and other letters is bound by the same $s_{min}$ and $s_{max}$ constraints, the space is treated just as a letter.

The next line contains the text that is currently displayed on the electronic caption. This string has $c_{cur}$ characters $(1 \le c_{cur} \le 30)$ – uppercase Latin letters from A to Z and spaces. There are no leading or trailing spaces.

The line after that contains $c_{cur}$ non-negative integer numbers. Each number defines the spacing (in pixels) before the corresponding letter or space of the currently displayed string. The first number is the spacing from the left side of the caption to the first letter, the second number is the spacing from the first letter to the second letter or space, etc. The whole string fits on the caption. The spacing for the currently displayed string does not have to obey $s_{min}$ and $s_{max}$ limits.

The next line contains the new text that should be displayed on the electronic caption. This string has $c_{new}$ characters $(1 \le c_{new} \le 30)$ – uppercase Latin letters from A to Z and spaces. There are no leading or trailing spaces.

All Latin letters that are used for the current and the new text are defined in the font description.

## Output

Write to the output file a single line with $c_{new}$ integer numbers, denoting the optimal spacing for the new text. The first number is the spacing from the left side of the caption to the first letter and should be non-negative, the second number is the spacing from the first letter to the second letter or space, etc. The spacing between the letters and space characters should be between $s_{min}$ and $s_{max}$ pixels inclusive. The text shall fit on the electronic caption. There is always at least one way to fit the text on the electronic caption satisfying the above constraints. If there are multiple optimal answers, write any of them.

## Sample input and output

`caption.in`

```
5 53 5 1 2
A ..*.. C .**** E ***** I ..*.. M *...* N *...* P ****. R ****.
  .*.*.   *....   *....   ..*..   **.**   **..*   *...*   *...*
  *...*   *....   ***..   ..*..   *.*.*   *.*.*   ****.   ****.
  *****   *....   *....   ..*..   *...*   *..**   *....   *.*..
  *...*   .****   *****   ..*..   *...*   *...*   *....   *..*.
ACM ICPC
3 1 1 1 1 1 1 1
NEERC
```

`caption.out`

```
19 2 2 1 1
```

## Problem D. Dictionary Size

| | |
|---|---|
| Input file: | `dictionary.in` |
| Output file: | `dictionary.out` |

The government of Disleksik Piple's Ripublyc had decided to improve literacy level of its citizens. To this end, the government issued a decree with a full list of dictionary words.

The rules for construction of new words were also greatly simplified: an *approved word* must either be a dictionary word or consist of two parts, where the first part must be a dictionary word or its non-empty prefix, and the second part – a dictionary word or its non-empty suffix.

The Institute of Language Simplification has assigned you the task to count the number of different approved words that can be constructed from the given dictionary.

### Input

The first line of the input file contains the number of dictionary words $n$ ($1 \le n \le 10\,000$). The following $n$ lines contain dictionary words, one word per line. Dictionary words are composed of lowercase Latin letters and are at least 1 and at most 40 letters in length.

The Institute of Language Simplification did not necessary do a good job of cleaning the input data, so their list may contain duplicates.

### Output

The output file must contain a single integer – the number of different approved words.

### Sample input and output

| dictionary.in | dictionary.out |
|---|---|
| 3<br>abc<br>def<br>abef | 60 |

# Problem E. Eve

| | |
|---|---|
| Input file: | `eve.in` |
| Output file: | `eve.out` |

Mitochondrial DNA is the Deoxyribonucleic acid molecule that is contained in mitochondria within cells of an organism.

Mitochondrial DNA is normally passed to a child exclusively from its mother. Because of this fact, it is possible to speak of "Mitochondrial Eve" which refers to the most recent common matrilineal ancestor of the entire population. Matrilineal ancestry is traced through female line: mother, grandmother, etc.

Mitochondrial Eve of the Earth's human population is estimated to have lived around 200 000 years ago, most likely in the East Africa.

In this problem, we consider a certain population of the same biological species (eukaryotic and anisogamous). The population has been observed for a period of time, and all births and deaths were clearly recorded. For some of the individuals within the population, their mitochondrial DNA was sequenced. It is assumed that each individual in the observed population received its mitochondrial DNA from its mother without any mutations.

Your task is to find out, whether all individuals currently alive have the same mitochondrial DNA.

## Input

The first line of the input file contains integer $n$ ($1 \le n \le 100\,000$), the number of individuals that were alive at the beginning of the observation period. IDs of these individuals are integers from 1 to $n$.

Next $n$ lines contain one character each. The $i$-th of these lines describes the sex of the individual with ID $i$. '`M`' stands for male, '`F`' stands for female.

The next line contains integer $m$ ($0 \le m \le 100\,000$), the total number of births and deaths that happened during the observation period.

Next $m$ lines contain description of birth and death events, listed in chronological order.

A birth event is described by three space-separated tokens: the ID of the father, the ID of the mother, and a character that describes the sex of the child ('`M`' for male, '`F`' for female). The ID given to the offspring is the smallest positive integer that hasn't been used as an ID by this point of time.

A death event is described by a single negative integer, whose absolute value equals the ID of the individual that died.

The next line contains integer $k$ ($0 \le k \le n + m$), the number of sequenced mitochondrial DNAs.

Each of the next $k$ lines contains two space-separated integers, the ID of the individual whose mitochondrial DNA has been sequenced, and the unique identifier of the mitochondrial DNA of that individual. Unique identifiers of two mitochondrial DNAs are the same if and only if the corresponding sequenced mitochondrial DNAs are the same. All unique identifiers of the mitochondrial DNAs are integers from 1 to $10^9$.

All the data given in the input file is consistent and non-contradictory. Each individual's mitochondrial DNA was sequenced at most once. At least one individual was alive at the end of the observation period.

## Output

The output file must contain a single word:

- `YES` – if it can be derived that all the individuals that are alive at the end of the experiment have the same mitochondrial DNA;
- `NO` – if it can be derived that some of the individuals that are alive at the end of the experiment have different mitochondrial DNA;
- `POSSIBLY` – if none of the cases above takes place.

## Sample input and output

| eve.in | eve.out |
|---|---|
| 4<br>M<br>F<br>M<br>F<br>12<br>3 4 F<br>1 2 M<br>1 2 M<br>3 4 F<br>-3<br>-2<br>-4<br>-1<br>6 5 M<br>7 5 F<br>-7<br>-6<br>0 | YES |
| 3<br>F<br>F<br>M<br>3<br>3 2 M<br>3 1 F<br>-3<br>2<br>4 100500<br>5 100500 | YES |
| 3<br>M<br>F<br>M<br>2<br>1 2 M<br>3 2 F<br>0 | POSSIBLY |
| 2<br>M<br>F<br>2<br>1 2 M<br>-2<br>2<br>1 2011<br>2 2012 | NO |

# Problem F. Flights

| | |
|---|---|
| Input file: | `flights.in` |
| Output file: | `flights.out` |

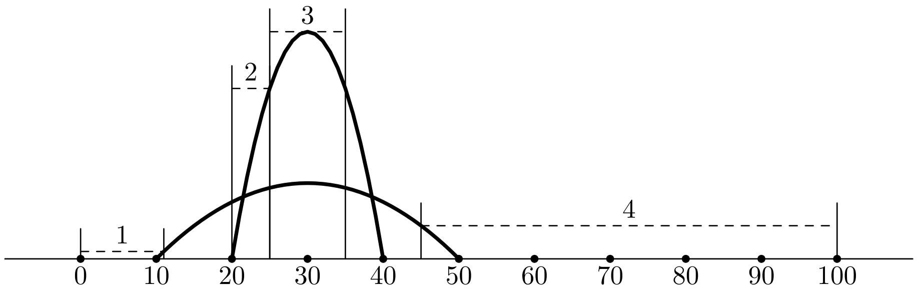
Army is busy: military exercises had started yesterday. All types of forces are doing, hopefully, a good job. For example, artillery is launching missiles, while aviation is delivering supplies to infantry.

The military ground space is a straight line. Aviation bases and infantry regiments are located somewhere on the line, and artillery is launching ballistic missiles everywhere. All missile launches are planned (don't forget, it's just an exercise), each at a certain time along a certain trajectory. Aviation flights are also planned in certain time and space intervals. Everything will be fine, but there are those missiles, which can be deadly even during exercises!

You should help aviation generals to plan the minimal safe altitude for each flight. Given the information about flight's time and space intervals, the minimal safe altitude for a flight is the minimal altitude such that all missile trajectories in the corresponding time and space interval are at or below this altitude. If there are no missiles in the flight's time and space interval, then the minimal safe altitude is defined to be zero.

Ballistic missiles are launched from the ground, which is defined to have a zero altitude, and fly along a vertically symmetrical parabola. Missile speed is ignored for this problem, missiles are assumed to follow their trajectory instantaneously.

For example, the picture below shows trajectories of two ballistic missiles in solid lines, and the minimal safe altitudes for four different flights in dashed lines. Vertical lines delimit space intervals of each flight in this sample. Time intervals of the flights in this sample include the launch of both missiles.



## Input

The first line of input contains a single integer $n$ — the number of missile launches planned ($1 \le n \le 50\,000$).

The following $n$ lines describe one missile launch each. Each line contains three integers: the missile launch point $p$ and coordinates of the highest point of missile trajectory $x$ and $y$ ($0 \le p < x \le 50\,000$, $0 < y \le 50$); $p$ and $x$ are coordinates along the military ground line, $y$ gives the altitude of the highest point of missile trajectory. Missiles are launched one by one every minute in the order they are described in the input.

Next line contains the single integer $m$ — the number of flights planned ($1 \le m \le 20\,000$).

The following $m$ lines describe one flight each. Each line contains four integers: $t_1$ and $t_2$ ($1 \le t_1 \le t_2 \le n$) — the time interval for the flight, and $x_1$ and $x_2$ ($0 \le x_1 \le x_2 \le 50\,000$) — the space interval for the flight along the military ground line. Time and space intervals are inclusive of their endpoints. Time moment 1 corresponds to the first missile launch, and time moment $n$ corresponds to the last one.

## Output

For each flight write a separate line with the minimal safe altitude, with absolute error not exceeding $10^{-4}$.

## Sample input and output

| flights.in | flights.out |
|---|---|
| 2<br>10 30 10<br>20 30 30<br>4<br>1 2 0 11<br>1 2 20 25<br>1 2 25 35<br>1 2 45 100 | 0.975<br>22.5<br>30.0<br>4.375 |
| 2<br>0 10 10<br>30 40 10<br>6<br>1 2 0 32<br>1 1 19 35<br>2 2 0 32<br>1 2 15 35<br>1 2 21 27<br>1 2 2 100 | 10.0<br>1.9<br>3.6<br>7.5<br>0.0<br>10.0 |

# Problem G. GCD Guessing Game

| | |
|---|---|
| Input file: | `gcd.in` |
| Output file: | `gcd.out` |

Paul had a birthday yesterday, and they were playing a guessing game there with Andrew: Andrew was trying to guess Paul's age. Andrew knew that Paul's age is an integer between 1 and $n$, inclusive. Andrew can guess any number $x$ between 1 and $n$, and Paul will tell him what is the greatest common divisor of $x$ and his age.

Here's a possible course of the game for $n = 6$. Andrew starts with guessing 3, and Paul replies that the greatest common divisor of 3 and his age is 1. That means that Paul's age can't be 3 or 6, but can still be 1, 2, 4 or 5. Andrew continues with guessing 2, and Paul replies 2. That means that Paul's age can't be 1 or 5, and the only two remaining choices are 2 and 4. Finally, Andrew guesses 4, and Paul replies 2. That means that Paul's age is 2, and the game is over.

Andrew needed three guesses in the above example, but it's possible to always determine Paul's age in at most two guesses for $n = 6$. The optimal strategy for Andrew is: at the first step, guess 6. If Paul says 1, then its 1 or 5 and he can check which one by guessing 5. If Paul says 2, then its 2 or 4, and he can check by guesing 4 as we've seen above. If Paul says 3, then we already know the answer is 3. Finally, if Paul says 6, the answer is 6.

What is the number of guesses required in the worst case if Andrew guesses optimally for the given $n$?

## Input

The input file contains one integer $n$, $2 \le n \le 10\,000$.

## Output

Output one integer — the number of guesses Andrew will need to make in the worst case.

## Sample input and output

| gcd.in | gcd.out |
|---|---|
| 6 | 2 |

## Problem H. Huzita Axiom 6

| | |
|---|---|
| Input file: | `huzita.in` |
| Output file: | `huzita.out` |

The first formal axiom list for origami was published by Humiaki Huzita and Benedetto Scimemi and has come to be known as the Huzita axioms. These axioms describe the ways in which a fold line can be generated by the alignment of points and lines. A version of the six axioms follows.

1. For points $p_1$ and $p_2$, there is a unique fold that passes through both of them.
2. For points $p_1$ and $p_2$, there is a unique fold that places $p_1$ onto $p_2$.
3. For lines $l_1$ and $l_2$, there is a fold that places $l_1$ onto $l_2$.
4. For a point $p_1$ and a line $l_1$, there is a unique fold perpendicular to $l_1$ that passes through point $p_1$.
5. For points $p_1$ and $p_2$ and a line $l_1$, there is a fold that places $p_1$ onto $l_1$ and passes through $p_2$.
6. For points $p_1$ and $p_2$ and lines $l_1$ and $l_2$, there is a fold that places $p_1$ onto $l_1$ and $p_2$ onto $l_2$.

Roman is a good coder, but he is new to origami construction, so he decided to write a program to calculate the necessary folds for him. He already finished coding the cases for the first five axioms, but now he is stuck on the harder case, the axiom number 6. So he decided to hire a team of good coders — your team — to implement this case for his program.

### Input

The input consists of one or more test cases. The total number of test cases $t$ is specified in the first line of the input file. It does not exceed 20 000.

Each test case consists of exactly four lines, describing $l_1$, $p_1$, $l_2$ and $p_2$, in that order. Each line is described by four integers — the coordinates of two different points lying on it: $x_1$, $y_1$, $x_2$, $y_2$. Each point is described just by two integers — its $x$ and $y$ coordinates. All coordinates do not exceed 10 by their absolute values. It is guaranteed that neither $p_1$ lies on $l_1$ nor $p_2$ lies on $l_2$. Lines $l_1$ and $l_2$ are different, but points $p_1$ and $p_2$ may be the same.

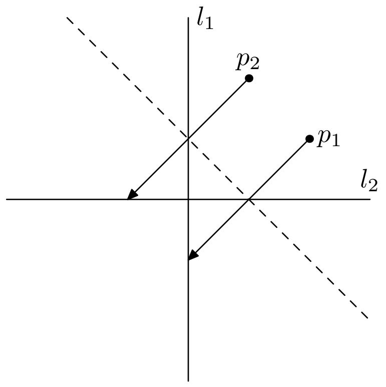
### Output

For each test case write a separate line with the description of the straight line one should use for folding. Use the same format as in the input — specify the coordinates of two points on it. Either $x$ or $y$ coordinates of those two points must differ by at least $10^{-4}$. Coordinates must not exceed $10^9$ by their absolute value. The judging program will check that both the distance between $p_1$ after folding and $l_1$; and the distance between $p_2$ after folding and $l_2$ do not exceed $10^{-4}$. If there are multiple solutions, any one will do. If there are no solutions, output the line of four zeros, separated by spaces.

### Sample input and output

The picture to the right illustrates the first sample. The fold line is dashed.

| huzita.in | huzita.out |
|---|---|
| 2<br>0 0 0 1<br>2 1<br>0 0 1 0<br>1 2<br>0 0 0 1<br>5 0<br>1 0 1 1<br>6 0 | 0.0 1.0 2.0 -1.0<br>0 0 0 0 |



# Problem I. Interactive Permutation Guessing

| | |
|---|---|
| Input file: | `standard input` |
| Output file: | `standard output` |

There is a permutation $a$ of size $n$ that you have to guess interactively.

You are allowed to make queries of the following kind. You output any permutation $b$ of size $n$. The information given back to you is the length of the longest common subsequence of permutations $a$ and $b$.

## Interaction protocol

First, your program must read from the standard input one line with integer $n$, the size of the permutation you have to guess.

Your program must then write to the standard output one line with a permutation and wait for a line in the standard input with a response, then write next query and read next response, and so on until you know $a$.

Once you receive response $n$ (which means you've found $a$), you're done and your program must exit.

## Input

The first line of the standard input contains integer $n$, the size of the permutation ($1 \le n \le 40$).

Each of the next lines of the standard input contains response to your query — the length of the longest common subsequence of the permutation queried by you and the permutation $a$.

## Output

Each line of the standard output should contain a space-separated list of integers that form a permutation you're querying.

Your can make at most $5n^2$ queries.

You must flush the standard output after printing each line. You must not print any lines after you receive the response $n$, just exit.

## Sample input and output

| standard input | standard output |
|---|---|
| 4 | 1 2 3 4 |
| 3 | 1 3 4 2 |
| 2 | 4 1 2 3 |
| 2 | 3 1 2 4 |
| 4 | |

# Problem J. Journey

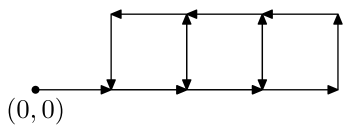
| | |
|---|---|
| Input file: | `journey.in` |
| Output file: | `journey.out` |

There is a robot who lives on a cartesian plane and likes to walk around it. One day he planned a very interesting journey around the plane. To make that journey he developed a program which he is going to follow. The program consists of $n$ functions: $f_1$, $f_2$, …, $f_n$. The $i$-th function $f_i$ is a sequence of $c_i$ commands. Each command is of one of the following types:

- GO: Move forward one meter;
- LEFT: Turn 90 degrees to the left;
- RIGHT: Turn 90 degrees to the right;
- F$k$: Follow the instructions of the function $f_k$, then continue following the instructions of the current function.

The robot starts the journey at his home located at the point with coordinates $(0, 0)$ following the instructions of the function $f_1$.

For example, consider the following set of functions:



$f_1$: GO F2 GO F2 GO F2
$f_2$: F3 F3 F3 F3
$f_3$: GO LEFT

The robot's journey for that case is shown on the picture.

In some cases the journey of the robot might never end. Consider for example the set of instructions consisting of one function $f_1$ that has the following commands: GO F1. In that case the robot keeps going forward and never stops.

The question that puzzles the robot now is how far from the home will he get during his journey. That is, consider the set of all points which the robot will visit. Find the maximum value of $|x| + |y|$ among all those points. If there are points on the path of the robot with arbitrarily large values of $|x| + |y|$, output "`Infinity`".

## Input

The first line of the input file consists of an integer number $n$ $(1 \le n \le 100)$. The following $n$ lines contain a description of the functions. The $i$-th line describes function $f_i$. It consists of the number $c_i$ $(1 \le c_i \le 100)$ — the number of commands for function $f_i$, followed by a description of each command.

## Output

Output the maximum value of $|x| + |y|$ among all points visited during the journey or "`Infinity`".

## Sample input and output

| `journey.in` | `journey.out` |
|---|---|
| `3`<br>`6 GO F2 GO F2 GO F2`<br>`4 F3 F3 F3 F3`<br>`2 GO LEFT` | `5` |
| `1`<br>`2 GO F1` | `Infinity` |
| `4`<br>`2 GO F2`<br>`7 LEFT GO GO GO F3 LEFT F4`<br>`5 GO F4 RIGHT F2 RIGHT`<br>`7 GO GO LEFT LEFT GO LEFT GO` | `13` |

## Problem K. Kingdom Roadmap

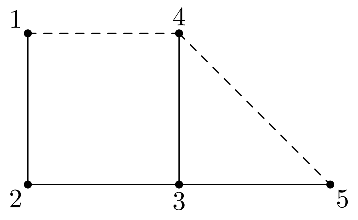
| | |
|---|---|
| Input file: | `kingdom.in` |
| Output file: | `kingdom.out` |

The Kingdom consists of $n$ cities connected by $n - 1$ roads in such a way that there is exactly one way to travel from one city to another.

The King is a busy man and he constantly travels from city to city. Unfortunately, during one of his travels one of the roads got damaged and needed serious repairs. As a result, the King was unable to reach his destination in time.

After the incident the King decided to improve reliability of the road system. It was decided that the improved road system shall be able to withstand one damaged road, i.e. there shall always be a path from one city to another even when one road in the Kingdom is damaged. As always, the budget is limited so you need to minimize the number of new roads.

For example, the picture below shows 5 Kingdom's cities numbered from 1 to 5 and roads between them in solid lines. One of the ways to build new roads is shown in dashed lines.



Your task is to build as few new roads as possible so that there is always a path between any two cities even if one of the roads gets unusable for any reason. There may be multiple optimal solutions. Any one can be used.

## Input

The first line of the input file contains integer $n$ — the number of cities in the kingdom ($2 \le n \le 100\,000$). The following $n - 1$ lines contain two integers $u_i$, $v_i$ each — the cities connected by $i$-th road ($1 \le u_i, v_i \le n$).

## Output

The first line of the output file shall contain one integer $k$ — the number of roads to be built. The following $k$ lines shall contain two integers $a_i$, $b_i$ each — the cities which should be connected by $i$-th new road ($1 \le a_i, b_i \le n$).

## Sample input and output

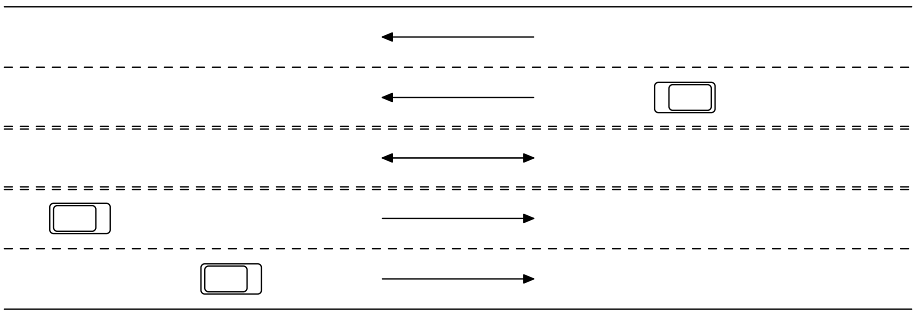
| kingdom.in | kingdom.out |
|---|---|
| 5<br>1 2<br>2 3<br>3 4<br>3 5 | 2<br>1 4<br>4 5 |
| 4<br>1 2<br>1 3<br>1 4 | 2<br>3 2<br>1 4 |

## Problem L. Lanes

| | |
|---|---|
| Input file: | `lanes.in` |
| Output file: | `lanes.out` |

Intercity Council for People Commute (ICPC) is in charge of operating a new bridge that connects two busy communities across the river. The bridge does not have enough throughput to accept the maximal traffic that can arrive on both sides of the bridge. The bridge has just a total of $n$ lanes in both directions, while roads that connect to the bridge on both sides are wider.

However, the traffic on both sides of the bridge is not symmetric. In the morning more people travel from the left side of the bridge to the right side, while in the evening more people travel from the right side to the left side. So it was decided to configure a bridge with $n_1$ lanes for left-to-right traffic, $n_2$ lanes for right-to-left traffic, and leave one lane in the center as a reversible one ($n_1 + 1 + n_2 = n$). In the morning the central lane will be open for left-to-right traffic, while in the evening the central lane will be open for right-to-left traffic.



The challenge is to figure out the optimal time to switch the direction of the central reversible lane.

In order to address this challenge, ICPC had gathered the data on the traffic on the typical day. The whole day from the morning to the evening was split into equal time intervals. The duration of time intervals was conveniently chosen in such a way, that the throughput of one lane is exactly one car per time interval. Time intervals were numbered from 1 in the morning to $m$ in the evening and the data on the number of cars arriving at each of $m$ time intervals at each side of the bridge was gathered.

The traffic is modeled at each time interval in both direction starting from the first time interval in the following way:

1. New cars arrive at the bridge.
2. Cars start crossing the bridge in the corresponding direction according to the number of lanes currently open in the given direction.
3. Remaining cars wait in the queue for the next time interval.

If there are still cars waiting at any side of the bridge after time interval $m$, then additional time intervals are modeled in the same way until all cars start crossing the bridge and no cars are waiting to cross, assuming that no more new cars arrive at the bridge after time interval $m$.

The reversal of the central lane in not instantaneous process. It takes time to let cars safely clear out the central lane before the lane can be open for the traffic in the reverse direction. The central lane will have to be closed for $r$ time intervals. It means, that if decision to reverse the lane on time interval $t$ ($1 \le t \le m$) is made, then the bridge will have $n_1 + 1$ lanes open from left-to-right traffic and $n_2$ lanes for right-to-left traffic before time interval $t$; $n_1$ and $n_2$ lanes correspondingly from time interval $t$ to time interval $t + r - 1$ inclusive; and $n_1$ and $n_2 + 1$ lanes correspondingly on time interval $t + r$ and afterwards.

The problem is to find such time interval $t$ ($1 \le t \le m$) to reverse the central lane, that minimizes the total time that all cars in both directions have to wait in the queue. If the are multiple optimal time intervals, then the earliest one has to be found.

## Input

The first line of the input file contains four integer numbers $n_1$, $n_2$, $m$, and $r$; $n_1$ and $n_2$ ($1 \le n_1, n_2 \le 10$) represent the number of permanently open lanes for left-to-right and right-to-left traffic respectively; $m$ ($1 \le m \le 100\,000$) is the number of time intervals in a day; $r$ ($1 \le r \le m$) is the number of time intervals that the central lane is closed for new traffic on reversal.

The following $m$ lines contains traffic data on the typical day. Each line describes time interval from 1 to $m$ with two integer numbers — the number of cars arriving at the bridge on the left and on the right. There are at most 100 arriving cars at each time interval on each side.

## Output

Write to the output file a single integer $t$ — the earliest optimal time interval to reverse the central lane per the problem statement.

## Sample input and output

| lanes.in | lanes.out |
|---|---|
| 2 2 10 2<br>1 0<br>2 1<br>3 2<br>4 2<br>3 3<br>2 3<br>1 5<br>0 3<br>1 2<br>0 1 | 4 |

The tables below model the traffic for the above sample, showing at each time interval the number of lanes open in the given direction, the number of cars that arrive at each time interval (step 1 in the traffic model as given in the problem statement), the number of cars that start crossing the bridge (step 2), and the number of cars remaining in the queue (step 3). The total wait time in the queue is 20 time intervals (10 for left-to-right cars and 10 for right-to-left). Time interval 11 is explicitly shown in this table to clarify that there is no traffic on this time interval and after it in this particular sample.

| time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| left-to-right lanes | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| left-to-right cars | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | 1 | 0 | 0 |
| left-to-right cross | 1 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 0 | 0 |
| left-to-right queue | 0 | 0 | 0 | 2 | 3 | 3 | 2 | 0 | 0 | 0 | 0 |
| right-to-left lanes | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| right-to-left cars | 0 | 1 | 2 | 2 | 3 | 3 | 5 | 3 | 2 | 1 | 0 |
| right-to-left cross | 0 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 0 |
| right-to-left queue | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 3 | 2 | 0 | 0 |