New Zealand Programming Contest 2023

PROBLEM SET

Problem	Points	Title
A	3	Girl Swaps
B	3	Get Out Of Jail
C	3	Penalty Shootout
D	3	Bulls and Cows
E	10	Beads
F	10	Election
G	10	Set Game
H	10	Traffic Lights
I	30	Packing Cases
J	30	Subway
K	30	Ferry Loading
L	30	Molecules 1
M	70	Molecules 2
N	100	Faucet Flow
O	100	Lambda Lifting
P	100	Losing Lakes

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin`, `Console.ReadLine` or equivalent. Input will be redirected from a file to form the input to your submission.

- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.

- Write all output to the screen, i.e. use `stdout`, `System.out`, `cout`, `Console.WriteLine` or equivalent. Do not write to `stderr`.

- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by a space.

- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.

- Unless otherwise stated, a *name* is a continuous sequence of from 2 to 30 characters.

- If it is stated that 'a line contains no more than *N* characters', this does not include the character(s) specifying the end of line.

- Input files are sometimes terminated by a 'sentinel' line. This line should not be processed.

Please also note that:

- The filenames of your submitted programs may need to follow a particular naming convention, for example the name of a Java file containing a public class needs to be the name of the class followed by the '.java' extension.

- DOMjudge will reject a submitted file which has any spaces in its file name.

- Unless otherwise specified, all problems have a time limit of 1 second. A *TIMELIMIT* error will be issued for submissions that exceed that limit on a single test run.

- Each problem description takes up at least 2 pages, one of which may be empty.

PROBLEM A GIRL SWAPS 3 POINTS

Part of an exercise at the Whangerei School for Young Ladies involves girls standing in a line. Their position in a line is given by a number; 1 means the first girl, 3 the third girl etc. The exercise involves girls swapping places and the participants having to remember the new order.

You are to write a program which accepts as input pairs of numbers which mean you should swap the positions of the girls at those two positions. The program will then report the new line positions which the participants are trying to remember.

Input

Input will consist of the first names of the girls in the line from first to last at the start of the exercise. Names will be unique, and will be as defined in the preamble. The names will be entered on one line, each name separated by a space. There will be at least 2 names and a maximum of 26 names.

The number of pairs (swaps) will be *n* (a non-negative integer no greater than 1000) on the second line. There will follow *n* lines of pairs of numbers which will be separated by a single space. The two numbers in a pair will always be different to each other and will both represent the position of a girl in the line. These are the girls who must swap places.

Output

Output should be of the names of the girls in their new order in the line.

Sample Input

```
Ruby Jo Fiona Phoebe Angela Sophia
3
1 3
2 6
4 1
```

Output for Sample Input

```
Phoebe Sophia Ruby Fiona Angela Jo
```

Explanation:

Swapping 1 and 3 leads to Fiona Jo Ruby Phoebe Angela Sophia

Swapping 2 and 6 leads to Fiona Sophia Ruby Phoebe Angela Jo

Swapping 4 and 1 leads to Phoebe Sophia Ruby Fiona Angela Jo

PROBLEM B GET OUT OF JAIL! 3 POINTS

According to Wikipedia, "Monopoly is a multi-player economics-themed board game". It goes on to say: "In the game, players roll two dice to move around the game board, buying and trading properties and developing them with houses and hotels. Players collect rent from their opponents, aiming to drive them into bankruptcy."

There is a jail on the board where players may find themselves sent for a number of reasons, most obviously for landing on the "Go to Jail" square. There are also 2 packs of cards which players have to obey when they land on appropriate squares. Both packs contain a "Go to Jail" card, but also a "Get out of jail free" card. A player may keep a "Get out of jail free" card until it is needed.

This problem assumes a player is in jail, and has to get out. There are 3 ways:

1. Throw doubles, that is where both dice show the same number.
2. After 3 turns use a "Get out of jail free" card.
3. After 3 turns, pay a $50 fine.

This problem assumes that the player will wait until they have played 3 turns before using their card or paying the fine.

Once out, the player moves forward the number of squares indicated by adding the dice from the last turn.

Input

The first line of input will be the dice scores for the first throw, on a single line and separated by a space.

If doubles are not thrown, a second similar line will appear, and similarly a third line if doubles are still not thrown.

If all 3 lines do not show doubles, a fourth line of input will show if the player has a "Get out of jail free" card (lower case y or n).

Output

Output will consist of a single line with the following format:

```
<Reason>.  Move forwards n squares.
```

<Reason> is one of

```
Doubles    Use card   $50 fine
```

`n` is the sum of the last 2 dice thrown.

Sample Input 1

```
2 6
4 5
1 2
n
```

Output for Sample Input 1

```
$50 fine. Move forwards 3 squares.
```

Sample Input 2

```
1 3
4 4
```

Output for Sample Input 2

```
Doubles. Move forwards 8 squares.
```

Sample Input 3

```
4 2
3 1
6 3
y
```

Output for Sample Input 3

```
Use card. Move forwards 9 squares.
```

PROBLEM C PENALTY SHOOTOUT 3 POINTS

In most major football tournaments, in knock out games a penalty shootout is used to determine the winners of a drawn match. The most recent FIFA World Cup (men) was decided by such means. For the uninitiated, 5 players from each team take alternating penalties, with the team scoring most goals winning.

For those taking the penalties, what is the best part of the goal to aim at for maximum chance of success? In this problem you will attempt to supply the answer.

The goal in a football match is approximately 7.5 metres wide and 2.5 metres high. To record the part of the goal the ball hit, the goal is divided into ½ metre square grids. Grids are numbered horizontally from the left of the goal from 0 to 14, and vertically from the ground upwards from 0 to 4. So, the bottom left hand corner would be grid 0,0 while the top right corner would be grid 14, 4.

Input

Input will consist of data from a number of penalty kicks. On the first line will be a single positive integer, *n*, being the number of kicks recorded, at least 5, no greater than 500. The next *n* lines will each contain data on 1 kick.

The first character of the kick data will be G for a goal, S for a save or M for a miss. The grid coordinates follow, horizontal first, with a space between each item. For a miss, the coordinates will be given as -1 -1 as no part of the goal was hit.

You may assume that there is at least one square with a score greater than 0.

Output

Output will consist of a line giving the best target spot or spots. This is found by scoring each grid square with +1 for each goal and -1 for each save. The square or squares with the highest score will be presented. Misses are ignored.

Where more than one square has the same best score, they must be presented in numerical order of horizontal coordinates then numerical order of vertical coordinates.

See the sample outputs for the required wording. Note the use of “place” and “places” as appropriate.

Turn over for sample input and output

Sample Input 1

```
15
G 0 1
G 14 1
S 7 1
G 10 1
G 0 1
S 2 1
S 11 2
M -1 -1
G 7 1
G 7 1
S 10 1
S 5 2
G 0 1
G 6 3
G 7 1
```

Output for Sample Input 1

```
Best place 0 1.
```

Explanation

3 goals were scored at location 0, 1 with no saves made.

Location 7, 1 was also scored from 3 times, but there was also 1 save.

Sample Input 2

```
12
M -1 -1
G 14 1
S 8 2
S 6 3
G 0 1
G 0 2
G 14 1
M -1 -1
G 1 3
S 9 2
G 0 2
G 2 4
```

Output for Sample Input 2

```
Best places 0 2, 14 1.
```

Explanation

Both locations recorded 2 goals with no saves. The location with the 0 horizontal value is shown first.

PROBLEM D **BULLS AND COWS** **3 POINTS**

When James Wardle invented his popular word game, Wordle, he was actually producing a modern version of a much older game. So old that William Shakespeare and Jane Austen may well have played it with their friends.

The game requires 2 players. The chooser sets a word of 4 letters and the guesser tries to guess it. After each guess, the chooser tells the guesser how many bulls and how many cows their guess has scored.

A bull is scored for every correct letter in the guess that is in the correct position in the word. A cow is scored for any other correct letter that is in the wrong place. If either word has repeated letters the rule is that each letter can only count towards the score once, and bulls are counted before cows. For example, if the set word is NEED, and the guess was EVEN, the score would be 1 bull (for the E in position 3) and 2 cows (for the N and the other E). The V would not score. If the set word is LOPE and the guess is EVEN, the score would be just 1 cow, for the first E.

Input

Each line of input consists of a single 4 letter word, all letters being in upper case.

The first line will be the secret word selected by the chooser.

Each other line will be a word put forward by the guesser. Input will end with a correct guess, or with the guesser entering GIVE UP.

Output

For each guess in the input, apart from GIVE UP, a single line will be output. The line will consist of the guess, followed by a space, followed by the number of bulls awarded, followed by a space, followed by the number of cows awarded. As shown in the sample, bulls and cows must be bull and cow if there is only 1.

The final line of output will be either

```
The word was guessed in N.
```

or

```
The word was not guessed. Answer: <word>.
```

N in the first version is the number of guesses made, an integer.

<word> in the second version is the secret word.

Turn over for sample input and output.

Sample Input 1

```
LOVE
FISH
VAIN
LANE
VILE
LOVE
```

Output for Sample Input 1

```
FISH Score 0 bulls and 0 cows.
VAIN Score 0 bulls and 1 cow.
LANE Score 2 bulls and 0 cows.
VILE Score 1 bull and 2 cows.
LOVE Score 4 bulls and 0 cows.
The word was guessed in 5.
```

Sample Input 2

```
QUAG
HOST
PINS
MELD
BARK
BLUB
DRAB
BARE
GIVE UP
```

Output for Sample Input 2

```
HOST Score 0 bulls and 0 cows.
PINS Score 0 bulls and 0 cows.
MELD Score 0 bulls and 0 cows.
BARK Score 0 bulls and 1 cow.
BLUB Score 0 bulls and 1 cow.
DRAB Score 1 bull and 0 cows.
BARE Score 0 bulls and 1 cow.
The word was not guessed. Answer: QUAG.
```

Problem E — Beads — 10 Points

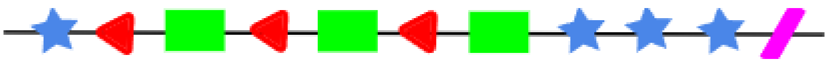
Your friend Allie designs necklaces from beads with various shapes and she wishes to share her design with another friend, Bella, who makes the necklaces. Allie uses a compact representation.

Each shape is described with a single bead letter (S for star, T for triangle, R for rectangle and L for line).

Instead of just writing down the sequence of beads in the necklace, Allie uses the following rules:

- if there are several identical beads following each other, write the number of beads, followed by the bead letter
- if there is a repeating sequence of beads, write the number of repetitions followed by the repeated sequence in parentheses
- otherwise, just write the bead letter

For example, the following necklace:



may be described as: S3(TR)3SL

You receive such a compact description from Allie and need to unpack it for Bella. She also needs to know how many of each type of bead is required.

Input

A bead sequence following the rules as described above. There will be at least 1 bead in the sequence.

The number of beads or the number of repetitions (in front of parentheses) in the bead sequence is guaranteed to be less than 100. There will not be any nested repeating sequences such as 3(S2(TR)L).

Output

Two output lines are required. The first contains the unpacked sequence of beads in the order given in the input.

The second line has 4 integers, space separated, representing the total number of Stars, Triangles, Rectangles and Lines needed, in that order.

Turn over for sample input and output

Acknowledgement to Sébastien Combéfis, Belgium, whose 2021 Bebras problem inspired this problem.

Sample Input

```
S3(TR)3SL
```

Output for Sample Input

```
STRTRTRSSSL
4 3 3 1
```

Explanation
The sequence, unpacked, contains 4 Stars, 3 Triangles, 3 Rectangles and 1 Line

PROBLEM F **ELECTION** **10 POINTS**

2023 is an election year in New Zealand, so here is an election problem for you.

In this problem you are simply given a list of votes from which to determine the winner. The vote list will contain one vote on each line, "vote" being the name of the candidate for whom the vote was cast. Your task is to find the name of the winning candidate (the one who received most votes) and their majority (the number of votes by which they won). If the top two or more candidates received the same number of votes, a tie must be declared with the tied candidates listed in alphabetical order. If only one candidate receives votes, then their majority is the number of votes they received.

Input

The first line of input contains a single positive integer, *n*, no greater than 1,000. The next *n* lines each contain a single name (as defined in the preamble) being the name of the person for whom a vote was cast. There will be at least one valid vote.

Output

Output should be the name of the winning candidate, followed by their majority in the format

```
<name> won by <majority> votes.
```

where <name> is the winner's name and `<majority>` is their majority.

NOTE If the majority is 1 you must use "vote" not "votes".

In the case of a tie, the output should be

```
Tie between <names list>.
```

where <names list> is a comma separated list of tied candidates in alphabetical order.

Turn over for sample input and output.

Sample Input 1

```
10
Ngata
Li
Brown
Li
Li
Ngata
Brown
Li
Li
Brown
```

Output for Sample Input 1

```
Li won by 2 votes.
```

Explanation:

Brown	3 votes
Li	5 votes
Ngata	2 votes

Sample Input 2

```
9
Ngata
Li
Brown
Li
Li
Ngata
Brown
Ngata
Brown
```

Output for Sample Input 2

```
Tie between Brown, Li, Ngata.
```

Explanation:

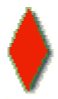
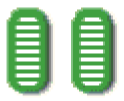
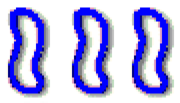
Brown	3 votes
Li	3 votes
Ngata	3 votes

PROBLEM G **SET GAME** **10 POINTS**

Set is a card game played with a pack of 81 unique cards. Each card in the pack has 4 properties, having one of the three possible values for each property.

Colour – Red (R), Green (G), Blue (B)	**Number** 1, 2, 3,
Shape – Diamond (D), Oval (O), Squiggle (S)	**Fill** – Filled (F), Shaded (S), Empty (E)

This sample contains 3 cards and illustrates all the variations. If it is not in colour, note that the left card is red, the middle card is green and the right card is blue.

		
RD1F	GO2S	BS3E

Below each card is its representation, being the value for its colour, shape, number and fill in that order. In this problem, cards will always be represented in this manner.

The 3 cards illustrated form a set. A set is a collection of 3 cards such that for each property all 3 cards are either the same or different. The 3 cards above are different for all 4 properties.

Because each card is unique, at least one property must be different.

Here are some other examples.

RD1F	RD1S	RD1E	SET. Same colour, shape and number, different fill.
RD1F	RD1S	RD2E	NOT a set. Two are 1s but one is not.

Your task in this problem is to solve the daily set puzzle on the Play Monster website[1] which shows 12 set cards and contains 6 sets. The idea is to identify those 6 sets as quickly as possible.

Input

Input consists of 12 lines, each line representing a single card. Each card will be represented by 4 characters as described above. The cards have an implied numbering, the first card listed being card 1, the last card 12.

[1] The website may be found on http://www.setgame.com/set/puzzle

Output

Six lines of output are required, one line to identify each of the sets presented in the puzzle. Each set is identified by giving the numbers of its 3 cards, in numerical order. The sets are listed in numerical order.

Sample Input

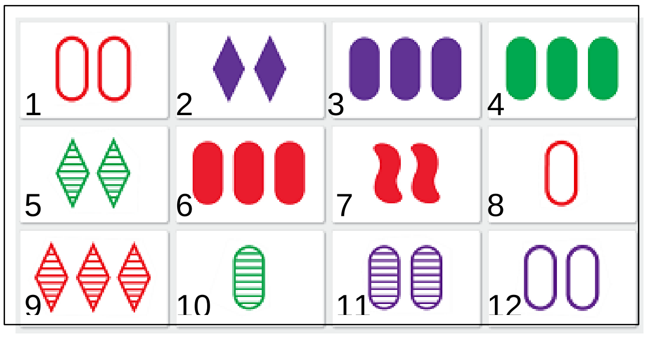
```
RO2E
BD2F
BO3F
GO3F
GD2S
RO3F
RS2F
RO1E
RD3S
GO1S
BO2S
BO2E
```

Output for Sample Input

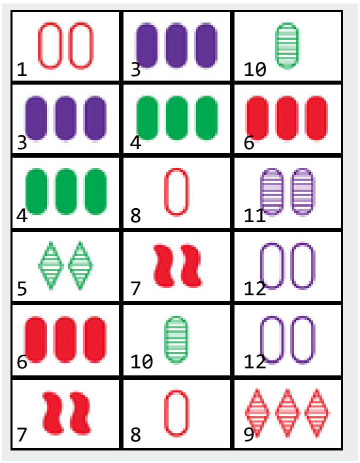
```
1 3 10
3 4 6
4 8 11
5 7 12
6 10 12
7 8 9
```

Explanation

The cards as listed:



The 6 sets:



Explanation:

Same shape
Diff colour, number, fill
Same shape, number, fill
Diff colour
Same shape
Diff colour, number, fill
Same number
Diff colour, shape, fill
Same shape
Diff colour, number, fill
Same colour
Diff shape, number, fill

PROBLEM H **TRAFFIC LIGHTS** **10 POINTS**

Inspired by hold ups at traffic lights near his home, Dr Rush decided to collect data on several sets of lights and report his findings to the relevant transport authorities.

Your task here is to process the data for Dr Rush and pass on your observations. What he particularly wants to know is what is the longest queue that formed during his observations, and what was the longest time it took a vehicle to pass through the lights.

Input

The first line will contain 2 positive integers each no less than 10 and no greater than 300. They will represent the observed length in seconds of the green and red phases respectively during the observations.

The next line will also contain 2 non-negative integers representing the number of cars in the queue at the start of observations, and how many seconds they had been waiting.

The next line contains `S`, a positive integer no greater than 50 giving the number of sequences that follow. The next `S` lines will each contain data for 1 sequence. Each sequence will begin with G for green or R for red, and be followed by a space then a non-negative number. Lines will alternate between red and green sequences.

For green sequences the number will be the number of cars who cleared the lights while the green light was on. This will always be no greater than the number of cars in the queue. For red sequences, it will be the number of cars that joined the queue while the red light was on.

The cars clearing the green light will be assumed to take 0 seconds to do so. Cars joining the queue at the red light will be assumed to have waited the full duration of the red light.

Continued

Output

The observations are to be recorded on 2 lines. The first line will give the longest queue length, the second line the slowest through time. The format will be:

```
Longest queue was V vehicles.
Longest through time was M minutes and S seconds.
```

V is the number of vehicles in the longest queue.

M and S are the minute and second values for the slowest passage.

Note that the text must be contextually correct so “1 vehicle”, “1 minute” and “1 second.” must be used where appropriate.

Sample Input	**Explanation**
`20 100`	26 cars cleared the lights. Call the cars A to Z for explanation.
`6 100`	Cars A to F in the queue for 100s so far.
`11`	
`G 4`	Cars A to D go after 100s.
`R 6`	Cars G to L join, 8 in queue.
`G 3`	Cars E and F leave after 220s, G after 100.
`R 8`	Cars M to T join, 13 in queue.
`G 5`	Cars H to L leave after 220s.
`R 6`	Cars U to Z join, 14 in queue.
`G 4`	Cars M to P leave after 220s.
`R 5`	5 more cars join, **15 in queue**.
`G 5`	Cars Q to T **leave after 340s**, U after 220.
`R 3`	3 more cars join, 13 in queue.
`G 5`	Cars V to Z leave after 340s. The extra cars are now ignored.

Output for Sample Input

```
Longest queue was 15 vehicles.
Longest through time was 5 minutes and 40 seconds.
```

PROBLEM I **PACKING CASES** **30 POINTS**

Angus has recently moved country. He had shipped all his belongings in wooden cubical boxes supplied by a packing company in his home country and he now wishes to return the empty boxes to the company. He can fit smaller boxes inside larger boxes, and a set of boxes nested in this way is called a "unit". To minimise shipping costs he wants as few units as possible. One box will fit inside another if the side length of the inner box is strictly less than the side length of the outer box.

Given that the total number of units has been minimised, Angus further wishes to minimise the maximum number of boxes in any one unit.

You are to write a program to help Angus determine a suitable nesting of boxes.

Input

The first line of input is the number of boxes, $n \leqq 10000$. The second line is a list of n space-separated integers, which are the side-lengths of all the boxes.

Output

The first line of output is an integer k, the minimum number of units to be shipped back. This is followed by k lines, each giving the side-lengths of the boxes comprising one unit, separated by spaces. Each side-length in the input should appear exactly once in the output, and the boxes in each unit must fit nested one within another. The order of the units in the output and the order of side-lengths in each unit does not matter. If there is more than one solution, any one will do.

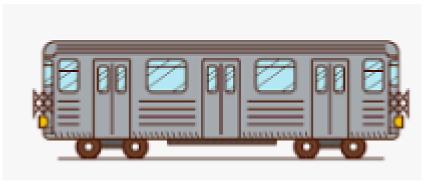
Sample Input

```
6
1 1 2 2 2 3
```

Output for Sample Input

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

PROBLEM J **SUBWAY** **30 POINTS**



You have just moved from Christchurch to study at a University in a big overseas city. Instead of cycling everywhere, you now get to walk and take the subway. Because you don't want to be late for class, you want to know how long it will take you to get to the University.

You walk at a speed of 10 km/h. The subway travels at 40 km/h. Assume that you are lucky, and whenever you arrive at a subway station, a train is there that you can board immediately. You may get on and off the subway any number of times, and you may switch between different subway lines if you wish. All subway lines go in both directions.

Input

The first line of input contains five space-separated non-negative integers: x_h, y_h, x_u, y_u and n, which are the x, y coordinates of your home, the x, y coordinates of your University and the number of subway lines, respectively.

The following n lines each contain a sequence of 6 or more integers x_1, y_1, x_2, y_2, ..., which are the x, y coordinates of each stop on the line, in order. The last 2 coordinates are -1, -1 to mark the end of the line; these should be ignored. You may assume the subway runs in a straight line between adjacent stops, and the coordinates represent an integral number of metres.

In total there are at most 200 subway stops in the city and all coordinates are in the range 0 – 12,000 inclusive.

Output

Output is the number of minutes it will take you to get to University, rounded to the nearest minute, taking the fastest route.

Sample Input

```
0 0 10000 1000 2
0 200 5000 200 7000 200 -1 -1
2000 600 5000 600 10000 600 -1 -1
```

Output for Sample Input

```
21
```

PROBLEM K **FERRY LOADING** **30 POINTS**

Recent flooding has damaged a bridge over a major river. The transport agency, Waka Kotahi, has taken the unusual action of reducing the bridge to one-way traffic, and have arranged for a car ferry to take cars in the other direction.

The ferry can take *n* cars at a time across the river in *t* minutes, and returns again after a further *t* minutes. The ferry operator is pondering what the optimal strategy would be if she knew all the car arrival times for the day in advance. Her goal is to finish for the day as early as possible with the secondary goal of making the minimum number of trips for that completion time. She can choose to leave on each crossing at any time she chooses, but obviously she can only take cars that have arrived up to that time. The times that individual cars have to wait is of no concern to her in this little algorithmic challenge.

Input

The first line of input contains three space-separate integers *n*, *t* and *m* where *n* and *t* are as above and *m* is the total number of cars arriving for the day. The following m lines each contain a single integer, the arrival time of a car in minutes from the start of the day. You may assume 0 < n, t, m < 1440. Car arrival times are in non-decreasing order.

Output

The output is two space-separated integers: the minimum time in minutes from the start of the day at which she can drop off the last car of the day and the minimum number of trips to achieve that time. Each departure with a load of cars counts as one trip.

Sample Input 1

```
2 10 10
0
10
20
30
40
50
60
70
80
90
```

Output for Sample Input 1

```
100 5
```

Sample Input 2

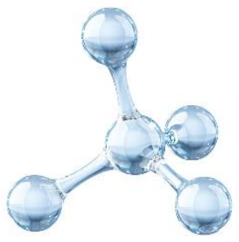
```
2 10 3
10
30
40
```

Output for Sample Input 2

```
50 2
```

PROBLEM L MOLECULES (PART 1) 30 POINTS

Note: Problems L and M are the same except for the limits on *r* and *c*. If you submit a correct solution for Problem L then you will score 30 points.



Organic molecules can be amazingly complex and need a great variety of shapes and conventions to represent them, particularly if we wish to depict details of their 3-dimensional structures. For this problem, we will restrict ourselves to reasonably simple compounds with only single bonds between atoms, that can be represented on a simple rectangular grid with bonds aligned horizontally or vertically. In such molecules, carbon is bonded to 4 adjacent atoms, nitrogen to 3, oxygen to 2 and hydrogen to 1. These numbers are called the *valences* of the atoms.

Of course, not all such grids represent valid molecules. Your task is to write a program that will determine whether a given grid could represent one or more valid molecules, satisfying the valences of all the atoms. Note that the molecule(s) in a valid grid do not need to exist in the real world.

Input

The first line of the input consists of a pair of integers (*r* and *c*, $1 \leq r, c \leq 5$) representing the number of rows and columns in the rectangle to follow. The next *r* lines contain *c* characters each, where the characters are chosen from the set {'H', 'O', 'N', 'C', '.'} representing hydrogen, oxygen, nitrogen, carbon and 'empty', respectively.

Output

Output consists of a single line containing a single word:

- *Valid* if it possible to draw horizontal and/or vertical bonds between neighbouring atoms so as to satisfy the valences of all the atoms in the grid, or
- *Invalid* if no such set of bonds exists.

Turn over for Sample Input and Sample Output

Sample Input #1

```
3 4
HOH.
NCOH
OO..
```

Output for Sample Input #1

```
Valid
```

Explanation

A valid bonding is:

```
 H O–H
  | |
  N–C–O–H
  | |
  O–O
```

Sample Input #2

```
3 4
HOH.
NCOH
OCNH
```

Output for Sample Input #2

```
Invalid
```

Sample Input #3

```
2 3
HOH
HOH
```

Output for Sample Input #3

```
Valid
```

Explanation

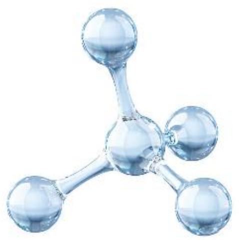
This could represent two water molecules:

```
H–O–H
H–O–H
```

PROBLEM M MOLECULES PART 2 +70 POINTS

Note: This problem is the same as Problems L except that the limits are *1 ≤ r, c ≤ 20*. To score the full points, you must submit to *both* problems. You may submit the same program to both (but you might find that Problem M requires a more refined solution).

Sample Input 1



```
4 10
0000000000
0000000000
0000N00000
0000000000
```

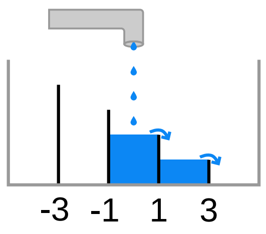
Output for Sample Input 1

```
Invalid
```

The sample inputs and outputs from Problem L also apply to this problem.

PROBLEM N **FAUCET FLOW** **100 POINTS**

A faucet is pouring water into a long, thin aquarium with various vertical dividers (walls) in it. The aquarium is initially empty, and its bottom is perfectly level. How long will it take for water to spill over its left- or right-most divider?



Sample Input 2

The faucet is above location *x* = *0*, and the dividers are located at *x* = *-1, -3, -5, ..., L* and *1, 3, 5, ..., R*. The dividers are attached perpendicular to the floor and sides of the aquarium, and have various heights. The aquarium's length is greater than *R* - *L*, its walls are higher than the highest divider, and its width is 1 unit everywhere. Water pours from the faucet at a rate of 1 cubic unit per second.

For the purpose of this problem, assume that water is an ideal liquid: it always flows downhill if it can, and if it cannot it spreads at an equal rate in all horizontal directions.

Input

The first line will consist of two integers: *L* (an odd negative integer) and *R* (an odd positive integer). The second line will contain the heights of the dividers (positive integers, at most 10^9) from left to right. There will be at most 1000 dividers.

Output

Output an integer on a single line, indicating how long it will take, in seconds, before water starts spilling over either the left or right divider.

Sample Input 1

```
-1 1
3 5
```

Output for Sample Input 1

```
6
```

Sample Input 2

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

Output for Sample Input 2

```
6
```

Sample Input 3

```
-3 5
1 2 2 1 1
```

Output for Sample Input 3

```
8
```

PROBLEM O — LAMBDA LIFTING — 100 POINTS

Thomas is developing a compiler for a functional programming language which supports *local function definitions*. Your task is to help him transform a program with local function definitions into a program with only global function definitions, by "lifting" the local functions to the global scope.

As an example, here is a program for computing the nth Fibonacci number:

```
func fibonacci ( n )
    func helper ( i , a , b )
        if i == n then a else helper ( i + 1 , b , a + b )
    end
    helper ( 0 , 0 , 1 )
end
```

Figure 1: An example program with a local function definition

In this example, `fibonacci` is a global function and `helper` is a local function defined inside `fibonacci`. The details are unimportant, but for the curious: The language does not have an explicit `return` keyword; instead, the value of the expression at the end of the function is returned implicitly. The **`if then else`** construct is an expression (like C's ternary operator, `cond ? val1 : val2`). In the example program, the parameter `i` of `helper` takes on the values `0` to `n`; the parameter `a` is the `i`th Fibonacci number; and `b` is the next Fibonacci number. The first two Fibonacci numbers are 0 and 1, so the `fibonacci` function kicks off the computation by passing `i = 0, a = 0, b = 1` to `helper`.

One useful property of local functions is that they may refer to identifiers declared in the enclosing scope. For instance, the `helper` function in Figure 1 refers to `n`, which is a parameter of the enclosing `fibonacci` function. This makes it a little tricky to lift local functions to global scope, because it might break references. The solution is to introduce *additional parameters* to the lifted functions. For example, the program in Figure 1 can be converted to the following program:

```
func fibonacci ( n )
    helper ( n , 0 , 0 , 1 )
end
func helper ( n , i , a , b )
    if i == n then a else helper ( n , i + 1 , b , a + b )
end
```

Figure 2: A transformed version of the program from Figure 1

Note how the parameter `n` was added to the signature of `helper` and to all the locations where `helper` is called.

This technique requires that function names only appear in call expressions (functions cannot be passed unevaluated to other functions). Functions and parameters can safely retain their names assuming all identifiers are unique. The program in the input is guaranteed to satisfy these two requirements.

To improve the efficiency of the transformed program, only the *minimal set* of required parameters should be introduced. For example, if the `helper` function in Figure 1 did *not* refer to `n`, then the transformation should *not* introduce `n` as an additional parameter of `helper`.

Input

The input will be a valid *program* according to the following (heavily reduced) syntax:

- A *program* consists of one or more *function definitions*
- A *function definition* consists of:
  - a *signature* line with the function name and zero or more parameters: **`func`** *funcname* `(` *paramname* **`,`** *paramname* **`,`** *...* `)`
  - followed by zero or more *function definitions*
  - followed by a line with an *expression*
  - followed by a line with the keyword **`end`**
- An *expression* is one of the following (from highest to lowest precedence):
  - an *integer literal*
  - a *paramname*
  - a *call expression* of the form: *funcname* `(` *expression* **`,`** *expression* **`,`** *...* `)`
  - an *add expression* of the form: *expression* + *expression*
  - an *equality expression* of the form: *expression* == *expression*
  - an *if-then-else expression* of the form: **`if`** *expression* **`then`** *expression* **`else`** *expression*
- An *integer literal* consists of one or more digits, without a leading zero (unless it is the literal `0`), and with at most nine digits in total
- *funcname* and *paramname* consist of one to ten lowercase letters, and will not be a keyword (**`func end if then else`**)

Notes

- All tokens are separated by a single space. The program in the input will *not* be indented, unlike Figures 1 and 2 (which are only indented for readability). Likewise, the output program should not be indented.
- All identifier references in the program will be within scope according to the following lexical scoping rules:
  - The scope of a parameter name is the entire declaring function.
  - The scope of a global function is the entire program.
  - The scope of a locally defined function is the entire enclosing function.

  (Note that this means functions can be used before they are defined; see Sample Input 2.)
- Function names will only ever appear in call expressions (as the function being called). Parameter names will never be used as a function in a call expression.
- All identifiers (both function names and parameter names) will be unique across the entire program. (This is ensured by a preprocessing step in the compiler.)

Limits

The input program will have at most 1,000 characters, and it is guaranteed that the output program will have at most 1,000 characters. There is no limit on the line length (other than the limit implied by the program length).

Output

Output the transformed program, with all local function declarations converted to global functions, with the minimal set of introduced parameters.

For judging purposes, the following additional constraints are placed on the transformation to ensure the output is unique:

- The order of the functions in the transformed program must be the same as the order in the original program, based on the location of their **`func`** keyword.
- The introduced parameters must be sorted alphabetically, and must be added at the start of the parameter list (before any original parameters).

Sample Input 1

(This program is identical to Figure 1, but without indentation.)

```
func fibonacci ( n )
func helper ( i , a , b )
if i == n then a else helper ( i + 1 , b , a + b )
end
helper ( 0 , 0 , 1 )
end
```

Output for Sample Input 1

(This program is identical to Figure 2, but without indentation.)

```
func fibonacci ( n )
helper ( n , 0 , 0 , 1 )
end
func helper ( n , i , a , b )
if i == n then a else helper ( n , i + 1 , b , a + b )
end
```

Sample Input 2

```
func main ( a , b )
func f ( )
a + g ( )
end
func g ( )
b + 1
end
f ( )
end
```

Output for Sample Input 2

```
func main ( a , b )
f ( a , b )
end
func f ( a , b )
a + g ( b )
end
func g ( b )
b + 1
end
```

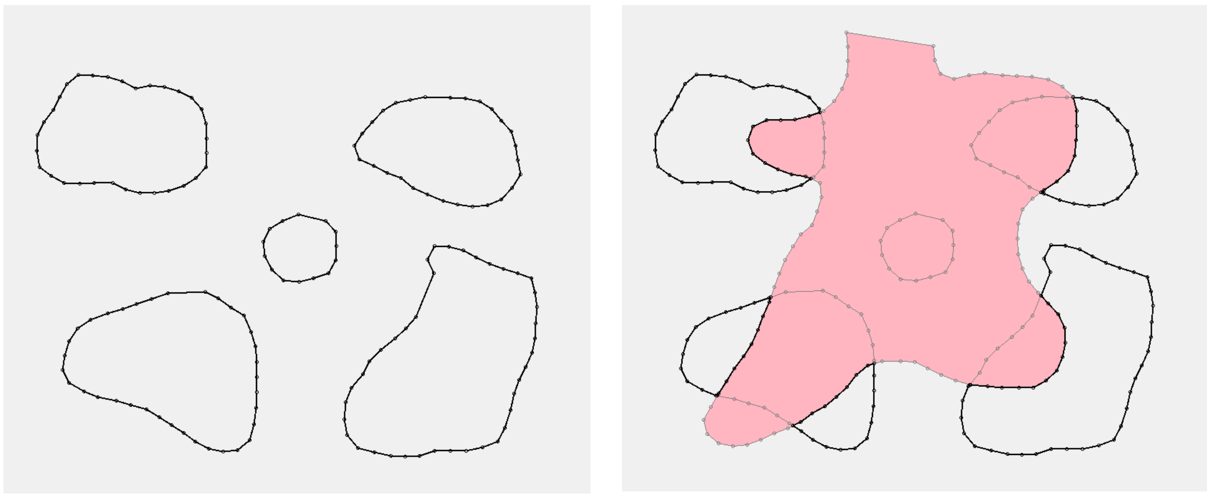
Explanation

In the original program, `f` and `g` are local functions defined inside `main`. When those functions are lifted, the parameters `a` and `b` from `main` need to be introduced as parameters of `f`, and `b` needs to be introduced as a parameter of `g`.

Problem P **Losing Lakes** **100 points**

The country of Lakalia has many lovely lakes. The people of Lakalia have a government department dedicated to the care and maintenance of lakes; one important function of the lake department is counting lakes and keeping track of any creation or loss. For example when there is a volcanic eruption, lava may fill in lakes or parts of lakes. The Lakalia lake department has asked you to write a program which, given information on existing lakes and volcanic lava coverage, will calculate the number of lakes remaining after a lava flow.

Lakes and lava areas are defined in 2D by polygonal outlines. For example, the image on the left below is a map of an area of Lakalia with five lakes. The image on the right shows the same area partly covered by a large lava flow (shaded/pink area). The heavy black outlines show the new lake shapes – again five lakes, but one of the originals has been lost (centre) and another divided in two (bottom left).



Input

The first line of input has the original number of lakes: *N*. Then for each lake there is a series of lines, the first of which has the number of vertices *V* for the lake's polygon, followed by *V* lines describing the coordinates of each vertex, with each line having the *x* and *y* coordinates (in km) as two space-separated integers. After the lake data there is a further polygon description (number and coordinate pairs) for the lava flow. The bounds are $1 \le N \le 10;\ 3 \le V \le 500;\ 0 \le x, y \le 2000$.

Output

One line holding a single integer – being the number of distinct lakes remaining after the lava flow.

Notes

You can assume that the input satisfies the following constraints.

- All polygons are closed shapes (closed by an edge between their first and last coordinates).
- The polygons are *simple polygons*, meaning they are not self-intersecting and do not have 'spikes': i.e. no edge directly folds back on the last.
- Lake polygons are distinct (non-overlapping) and non-touching.
- All polygon edges have length at least 1 km.

- All intersections between lava and lake polygons are crossings – i.e. lava and lake edges never run along together (no collinear overlap) and polygons never just touch at a point.
- The remaining lakes after the lava flow are never very small – always at least one square kilometre.

Submission restrictions

You are not permitted to use graphics or geometry libraries (even if such libraries are available in the judging environment). The judges will manually check submissions and remove submissions that do so.

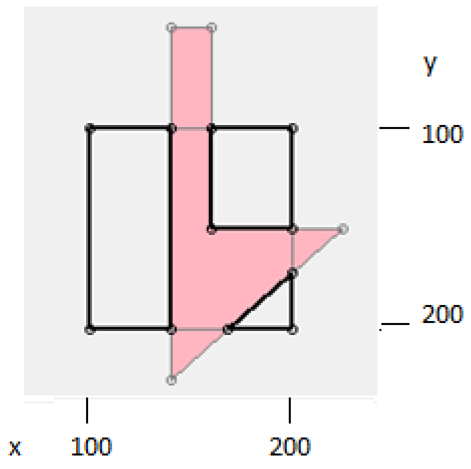
Sample Input

```
1
4
100 100
100 200
200 200
200 100
5
160 50
160 150
225 150
140 225
140 50
```

Output for Sample Input

```
3
```

Illustration of sample


y
100
200
x
100
200