ANZAC Round 5

August 27th 2022

Problems

A Adolescent Architecture
B Bookshelf Building
C Confined Catching
D Decorative Dominoes
E Exhausting Errands
F Flip Flow
G Gravity Grid
H Hectic Harbour
I Impressive Integers
J Joining Jargon
K Knightly Knowledge
L Lexicographical Lecturing
M Mixtape Management

The time limit for all problems is 1 second, except problems B and H (2 seconds).
The memory limit for all problems is 1024MB, except problem H (2048MB).
Beginners are recommended to start with problems F and J.

This page is intentionally left (almost) blank.

Problem A: Adolescent Architecture

Little Peter is building a stack out of his toy blocks. He is using two kinds of blocks – cubes and cylinders – and wants to stack all of them into a single tower, where each block other than the topmost block has a single block standing on it. In order for the tower to be stable, the outline of each block needs to be fully contained within the outline of the block below when looking at the tower from above (the outlines are allowed to touch). Is it possible to construct such a tower, and if so, in which order do the blocks need to be stacked?

Building blocks by Thaliesin on Pixabay

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 100$), the number of blocks.
- n lines, each with the description of a block. The description consists of a string giving the type of block (`cube` or `cylinder`) and an integer a ($1 \leq a \leq 1\,000$) giving the size of the block – if the block is a cube then a is its side length, and if it is a cylinder then a is the radius of its base (note that the height of the cylinder does not matter).

Output

If there is no way to construct the tower, output `impossible`. Otherwise output n lines, giving the order in which to stack the blocks from top to bottom.

Sample Input 1	Sample Output 1
3 cube 7 cube 11 cylinder 5	cube 7 cylinder 5 cube 11

Sample Input 2	Sample Output 2
2 cube 5 cylinder 3	impossible

Sample Input 3	Sample Output 3
3 cube 4 cylinder 2 cube 4	cylinder 2 cube 4 cube 4

This page is intentionally left (almost) blank.

Problem B: Bookshelf Building

Bookshelf by Free-Photos on Pixabay

Anna just came home from her favourite furniture store *Ikey Yeah!* where she bought a new bookshelf designed by her favourite artist *Bill Lee*. The shelf has a rectangular shape with a width of x and a height of y. Included in the box of the shelf is one board that Anna may put horizontally inside the shelf. For this purpose the frame of the shelf already contains pre-drilled holes at heights $1, 2, \ldots, y-1$. Anna may attach the board to the frame at any of these heights. She can also decide not to install it at all.

Anna owns n books, which all have the same depth. The books' depth matches that of the shelf exactly. However, due to differences in the formats and the numbers of pages, her books may have different widths and heights. Anna does not want to flip her books or stack several of them on top of each other. Instead, she wants to store all of them in an upright fashion inside her bookshelf, such that the width and height of each book are aligned with the width and height of the bookshelf. Help Anna to find the perfect position for the board (or tell her not to use it at all) so that she can store all her books in her new bookshelf.

For this problem you may assume that the frame and the board of the shelf are infinitely thin.

Input

The input consists of:

- One line with three integers n, x and y, where
  - n ($1 \leq n \leq 10^4$) is the number of books that Anna owns;
  - x ($1 \leq x \leq 10^4$) is the width of the bookshelf frame;
  - y ($1 \leq y \leq 10^4$) is the height of the bookshelf frame.
- n lines, the ith of which contains two integers w_i and h_i, where
  - w_i ($1 \leq w_i \leq 10^4$) is the width of the ith book;
  - h_i ($1 \leq h_i \leq 10^4$) is the height of the ith book.

Output

If there is no possibility to store Anna's books in the shelf, print out `impossible`. In case Anna decides not to install the board, output `-1`. Otherwise, output the height at which the board should be installed. If there are multiple possible heights, you may output any one of them.

Sample Input 1	Sample Output 1
4 4 4 3 1 1 1 1 3 2 2	3

Sample Input 2

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

Sample Output 2

```
-1
```

Sample Input 3

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

Sample Output 3

```
impossible
```

Problem C: Confined Catching

You are playing a board game against an AI on a square grid consisting of $n \times n$ cells. You have two game pieces and the AI has one, and each piece is placed in one of the grid cells. Your goal is to "catch" the AI's piece, that is, one (or both) of your pieces has to lie in the same cell as the AI's piece after one of your turns. When this happens, you win and the game ends. You lose if you have not won after 600 turns.

Each turn, you have up to five movement options per piece: You can move a piece up, down, left, or right to an adjacent cell (if there is one) or let the piece remain in its current cell. The AI has the same options for its piece in each of its turns. Of course, you can move your pieces completely independently from one another and even have them occupy the same cell.

Your goal is simple: Win the game! You can safely assume that this is always possible.

Initial Input

Before it is your first turn, your program will receive:

- One line with an integer n ($3 \leq n \leq 100$), giving the size of the grid.
- One line with four integers x_1, y_1, x_2, y_2 ($1 \leq x_1, y_1, x_2, y_2 \leq n$), giving your pieces' initial positions.
- One line with two integers x, y ($1 \leq x, y \leq n$), giving the AI's piece's initial position.

You can safely assume that your pieces do not lie in the same cell as the AI's piece (but they may lie in the same cell as each other).

Interaction Protocol

Your submission will be interacting with a special program called the *grader*. This means that the output of your submission is sent to the grader and the output of the grader is sent to the standard input of your submission. This interaction must follow a specific protocol:

Your submission and the grader alternate writing turns, with you going first. Your turns consist of a single line with four integers x_1, y_1, x_2, y_2 to specify the locations of your pieces after your turn (in the same order as in the initial input). Similarly, the grader will send lines with two integers x, y to indicate the AI's turn. You can safely assume that the grader will only send you valid moves, and the AI will never choose to place its piece in a cell occupied by one or both of your pieces. Your submission may take at most 600 turns.

After each of your turns you should *flush* the standard output to ensure that the request is sent to the grader. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java and `sys.stdout.flush()` in Python.

After you send the grader a valid turn catching the AI's piece, the game ends. Your submission should then terminate with exit code `0` as usual. For convenience, the grader will send a single line containing `0 0` to signal the end of the game, which your submission may or may not read.

Your submission will be accepted if it follows the rules and protocol above and wins the game. If it sends any invalid turn, it will be judged as "Wrong Answer".

For your convenience, we have provided you with a testing tool that lets your solution interact with a simple version of the AI opponent. It is included in the ZIP archive with sample data that you can download from the DOMjudge problem overview page.

Read **Sample Interaction 1** **Write**

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

```
1 2 3 2
```

```
3 3
```

```
1 2 3 3
```

```
0 0
```

Problem D: Decorative Dominoes

Marie likes Dominoes. She is too young to fully understand the game, so she just creates arrangements based on the following simple rule: Each of the two ends of a domino must be adjacent to an end of another domino with the same number on it.

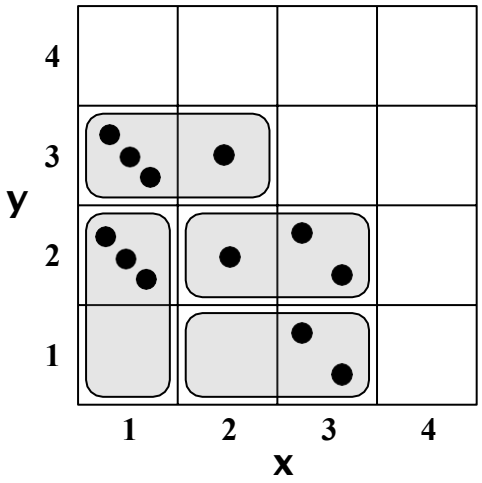


Figure D.1: Visualization of the first sample test case.

Today, Marie found a large box of blank dominoes. This is very exciting for her because now she can show her full creativity by first creating an unrestricted arrangement and then, in a second step, painting numbers on both ends of all dominoes so that her simple rule is fulfilled.

She already decided that putting the same number on each end of every domino is not satisfying enough for her. She only wants to use each number at most twice. However, she does not restrict herself to numbers between 0 and 6, and she also does not care if two dominoes have the same pair of numbers on them.

Marie positions the dominoes along an integer grid, so that each domino occupies exactly two neighbouring grid squares. Note that Marie's arrangement does not necessarily have to be connected.

After Marie decided on an arrangement, she notices that choosing suitable numbers is harder than initially anticipated. Help her to find a valid numbering for her given arrangement or state that this is impossible.

Input

The input consists of:

- One line with an integer n ($2 \leq n \leq 5\,000$), the number of dominoes in Marie's arrangement.
- n lines, each with four integers x_1, y_1, x_2, y_2 ($1 \leq x_1, y_1, x_2, y_2 \leq 10\,000$), where (x_1, y_1) and (x_2, y_2) are the grid positions of the two ends of one of the dominoes.

It is guaranteed that all dominoes occupy two neighbouring positions in the integer grid and no two dominoes overlap.

Output

If a valid numbering exists, print n lines, the ith of which contains two numbers, the integers that Marie should write on the two ends of the ith domino, respectively. Output the numbers in

the same order as the dominoes (including their two ends) appear in the input. All numbers in the output should be integers between 0 and 10^6 inclusive. In case multiple valid numberings exist, you may output any one of them. If there does not exist a valid numbering, output `impossible` instead.

Sample Input 1

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

Sample Output 1

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

Sample Input 2

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

Sample Output 2

```
impossible
```

Problem E: Exhausting Errands

Dolly the delivery drone is out for a busy working day. It has to complete n errands in a street where ℓ houses are lined up in a row, numbered in ascending order as 1, ..., ℓ. The distance between adjacent houses is 1. Each errand consists of picking up a package at some house a and delivering it to another house b. Dolly can start with any errand, complete the errands in any order, and is able to carry an unlimited number of packages at the same time. Your job is to find the minimal total distance Dolly has to cover to complete all errands. The delivery route can start and finish at arbitrary locations along the street.

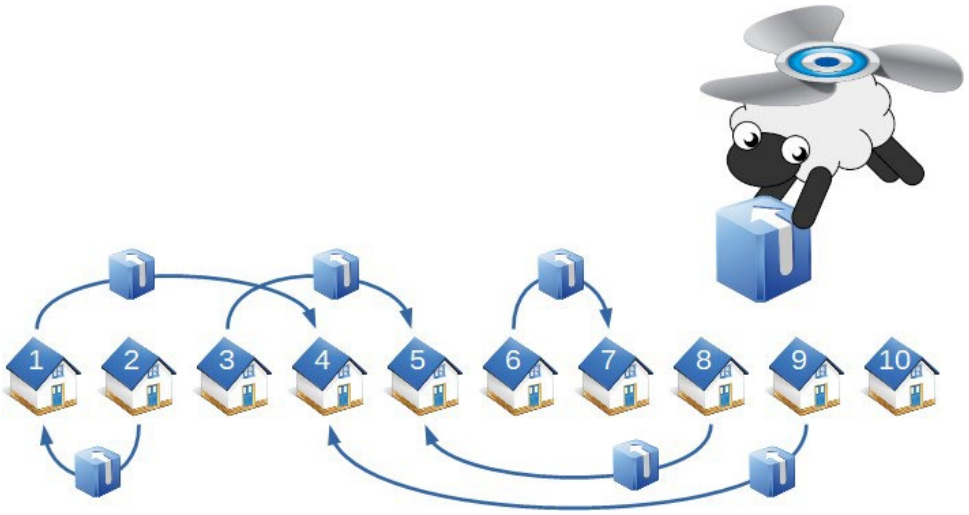


Figure E.1: Illustration of Sample Input 1. The shortest route is 2 → 1 → 9 → 4 with length 14.

Input

The input consists of:

- One line with two integers ℓ and n ($1 \leq \ell \leq 10^9$, $1 \leq n \leq 10^5$), where ℓ is the number of houses in the street, and n is the number of errands.
- n lines, each with two integers a and b ($1 \leq a, b \leq \ell$, with $a \neq b$) describing an errand where a package must be picked up at house a and be delivered to house b.

Output

Output a single line with the minimal distance Dolly has to cover from picking up the first package until delivering the last package.

Sample Input 1

```
10 6
1 4
3 5
6 7
2 1
9 4
8 5
```

Sample Output 1

```
14
```

Sample Input 2

```
100 3
11 50
50 49
36 35
```

Sample Output 2

```
42
```

Problem F: Flip Flow

Hourglass by nile on Pixabay

For over 600 years, the hourglass has been a well-known timekeeping instrument. An hourglass consists of two glass flasks arranged one above the other which are connected by a narrow channel. Inside the hourglass there is sand which slowly flows from the upper to the lower flask. Hourglasses are typically symmetrical so that the sand always takes the same amount of time to run through, regardless of orientation. For the purposes of this problem, we also assume that the flow rate of the sand is a known constant and does not depend on the amount or distribution of sand in the upper half.

Your friend Helen was bored and has been playing around with her hourglass. At time 0, all the sand was in the lower half. Helen flipped the hourglass over several times and recorded all the moments at which she did so. How many seconds does she need to wait from the current time until all the sand is back in the lower half?

Input

The input consists of:

- One line with three integers t, s and n, where
  - t ($1 \leq t \leq 10^6$) is the current time;
  - s ($1 \leq s \leq 10^6$) is the amount of sand in the hourglass, in grams;
  - n ($1 \leq n \leq 1\,000$) is the number of times the hourglass was flipped.
- One line with n integers $a_1, \ldots, a_n$ ($0 < a_1 < \cdots < a_n < t$), the times at which the hourglass was flipped.

All times are given in seconds. You may assume that the sand flows from the top to the bottom at a constant rate of 1 gram per second.

Output

Output the time in seconds needed for the hourglass to run out starting from time t.

Sample Input 1	Sample Output 1
10 7 2 4 9	4

Sample Input 2	Sample Output 2
2000 333 3 1000 1250 1500	0

Sample Input 3	Sample Output 3
100 10 5 15 20 93 96 97	5

This page is intentionally left (almost) blank.

Problem G: Gravity Grid

Alice and Bob are playing a generalized version of *Connect Four*. In their game, the board consists of w columns of height h and the goal is to be the first player to complete a row of k tiles of equal colour, either vertically, horizontally or diagonally. The two players alternate dropping their tiles into one of the columns, with Alice using red tiles and going first and Bob using yellow tiles and going second. Once a tile is dropped, it falls down to the bottommost available position, making that position no longer available. Once a column has h tiles in it, it becomes full and the players can no longer drop their tiles there.

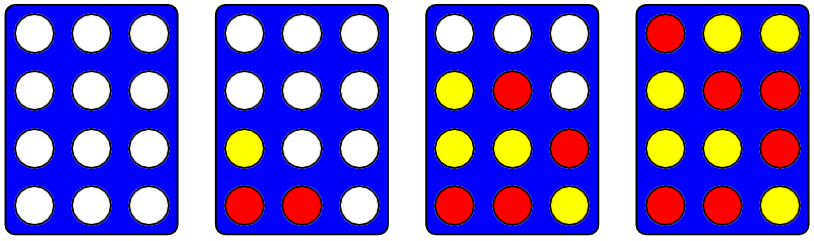


Figure G.1: Visualisation of the sample cases, showing the state of the game after 0, 3, 8 and 12 moves, respectively. Alice's tiles are shown in red, Bob's tiles in yellow.

As Alice and Bob found it quite challenging to keep track of the winning condition, they just kept playing until the board was completely filled with tiles. They recorded a log of the moves they made and asked you to tell them who won the game, and on what turn they did so. If neither player managed to complete a row, the game ends in a draw, so report that instead.

Input

The input consists of:

- One line with three integers h, w and k ($h, w \geq 1$, $h \cdot w \leq 250\,000$, $1 \leq k \leq \max(h, w)$). The columns are numbered from 1 to w.
- One line with $h \cdot w$ integers $a_1, \ldots, a_{h \cdot w}$ ($1 \leq a_i \leq w$ for each i), where a_i is the index of the column that the ith tile was dropped in. Odd indices correspond to Alice's moves and even indices correspond to Bob's moves. Each column appears exactly h times in this list.

Output

Output the winner of the game (`A` for Alice or `B` for Bob), followed by the number of moves needed to decide the winner. If the game ends in a draw, output `D` instead.

Sample Input 1

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

Sample Output 1

```
A 3
```

Sample Input 2

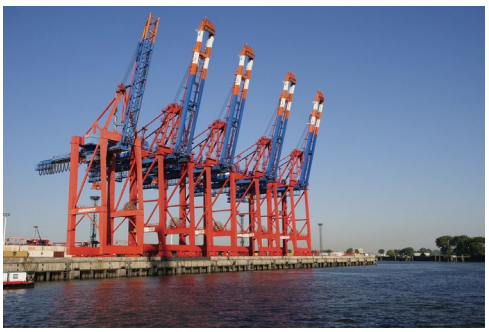
```
4 3 3
1 1 2 3 3 2 2 1 1 2 3 3
```

Sample Output 2

```
B 8
```

Sample Input 3	Sample Output 3
4 3 4 1 1 2 3 3 2 2 1 1 2 3 3	D

Problem H: Hectic Harbour



Gantry cranes by wasi1370 on Pixabay

There are two gantry cranes operating on the same gantry of length n. The gantry has some fixed integral positions, labelled from 1 to n, at which the cranes must perform loading/unloading operations. In the beginning the first gantry crane is located on the very left of the gantry at position 1, while the second one is located on the very right of the gantry at position n. In each time step a gantry crane can either move to a neighbouring integral position or stay at its current position (and potentially perform a loading/unloading operation). To prevent the gantry cranes from hitting each other, the first crane needs to stay strictly to the left of the second crane at all times. For both cranes you are given a task list consisting of gantry positions at which the cranes must perform loading/unloading operations. Both cranes must perform their assigned operations in the given order. What is the minimal amount of time necessary for both gantry cranes to finish their tasks? It is guaranteed that the first gantry crane never has to operate at position n of the gantry while the second gantry crane never has to operate at position 1. For both gantry cranes the first and last loading/unloading operation in the task list is their initial position.

Input

The input consists of:

- One line with integers n, a and b where
  - n ($2 \leq n \leq 2\,000$) is the length of the gantry;
  - a ($2 \leq a \leq 50$) is the number of operations in the task list of the first gantry crane;
  - b ($2 \leq b \leq 50$) is the number of operations in the task list of the second gantry crane.
- One line with a integers $k_1, \ldots, k_a$ ($1 \leq k_i \leq n - 1$ for all i), the tasks of the first gantry crane.
- One line with b integers $\ell_1, \ldots, \ell_b$ ($2 \leq \ell_i \leq n$ for all i), the tasks of the second gantry crane.

The first and last task of both gantry cranes are at their initial position, i.e., $k_1 = k_a = 1$ and $\ell_1 = \ell_b = n$.

Output

Output the minimum number of time steps necessary for both gantry cranes to finish their assigned tasks.

Sample Explanation

In the first sample test case the gantry is of length 3, the first gantry crane has 2 operations in its task list while the second gantry crane has 4 operations in its task list. At least 6 time steps are necessary for both gantry cranes to finish their assigned tasks.

Time	Gantry Crane 1	Gantry Crane 2
1	Operate at 1	Operate at 3
2	Operate at 1	Operate at 3
3	Idle at 1	Move from 3 to 2
4	Idle at 1	Operate at 2
5	Idle at 1	Move from 2 to 3
6	Idle at 1	Operate at 3

In the second sample test case the gantry is of length 4 and both gantry cranes have to perform 4 operations. At least 9 time steps are necessary for both gantry cranes to finish their assigned tasks.

Time	Gantry Crane 1	Gantry Crane 2
1	Operate at 1	Operate at 4
2	Move from 1 to 2	Move from 4 to 3
3	Operate at 2	Operate at 3
4	Move from 2 to 3	Move from 3 to 4
5	Operate at 3	Idle at 4
6	Move from 3 to 2	Move from 4 to 3
7	Move from 2 to 1	Operate at 3
8	Operate at 1	Move from 3 to 4
9	Idle at 1	Operate at 4

Sample Input 1

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

Sample Output 1

```
6
```

Sample Input 2

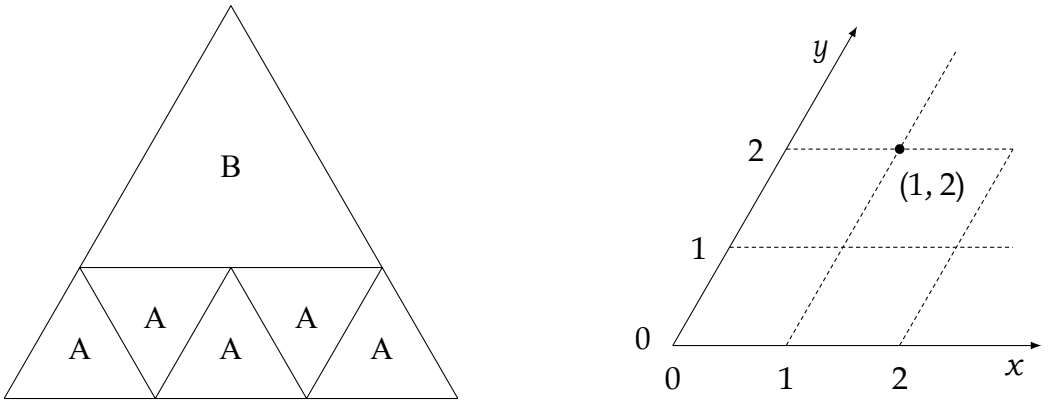
```
4 4 4
1 2 3 1
4 3 3 4
```

Sample Output 2

```
9
```

Problem I: Impressive Integers

Recently, Alice and her sister Clara learned about *adorable* numbers: A positive integer n is called adorable if there exist some integers a, b, and c so that an equilateral triangle with side length c can be tiled with n smaller equilateral triangles, each having a side length of either a or b. For instance, 6 is an adorable number as shown in Figure I.1a below.



(a) Visualization of the second sample output. (b) Coordinate system used for the output.

Figure I.1

They decided to see who can come up with more adorable numbers, but it turns out that checking all the numbers manually is too cumbersome. Therefore, Alice asked you to help her check whether the numbers Clara listed are adorable or not. As she wants to be sure that every number claimed to be adorable really is adorable, she asked you to write a program that, given an integer n, determines if it is adorable and if so, outputs a valid tiling as shown in Figure I.1a.

Input

The input consists of:

- A single positive integer n ($1 \leq n \leq 1\,000$).

Output

If the given integer n is adorable, output a valid tiling using the following format:

- Three integers a, b, c ($1 \leq a, b \leq c \leq 10^9$), such that an equilateral triangle with side length c can be tiled with n equilateral triangles with side lengths a and b.
- n lines, each describing one of the triangles and consisting of:
  - one of `A` and `B`, specifying if the side length of the triangle is a or b;
  - two integers x, y, giving the leftmost corner of the triangle;
  - one of `U` and `D`, specifying whether the triangle is pointing upwards or downwards.

  These triangles must form a valid tiling of the equilateral triangle with corners at $(0, 0)$, $(0, c)$, and $(c, 0)$, where all coordinates are given using the coordinate system in Figure I.1b.

Otherwise, if n is not adorable, output `impossible`.

If your tiling consists of triangles that all have the same size, you may either use `A` or `B` exclusively for all of your triangles, or set $a = b$ and arbitrarily label each triangle with `A` or `B`.

It can be shown that for every adorable number in the input range it is possible to construct a tiling according to the above set of rules. You may output any valid tiling.

Sample Input 1	Sample Output 1
2	impossible

Sample Input 2	Sample Output 2
6	1 2 3 B 0 1 U A 0 0 U A 0 1 D A 1 0 U A 1 1 D A 2 0 U

Problem J: Joining Jargon

This is not a problem about Brexit. Or at least not about the social or economic implications of Brexit. Instead, we – the Grammatical Correctness Policing Committee (GCPC) – want to focus exclusively on the linguistic challenges posed by this combination of the words "Britain" and the noun "exit". We are very concerned about the ambiguity of this construction. When using the term Brexit, it is not clear at all whether it is supposed to refer to to Great Britain or Brazil. Or perhaps Bremen leaving the Bundesliga.

And it's not just Brexit, you might also have heard of "Megxit" (having to do something with the British royal family) and similar constructions.

Looking ahead, we want to avoid a similar disaster in the future. For that purpose we, as a European agency with German roots, would like to implement some standardisation. We have focused on the act of entering. If a subject (a person, an organization, a plant, etc.) enters (or possibly reenters) something, our leading linguistic scientists suggest we look for the last vowel in the subjects name (`a`, `e`, `i`, `o` and `u` are considered vowels). We cut off any letters after this last vowel and add the ending `ntry` instead. Here are some examples:

- If Britain were to reenter the European Union, we would call it "Britaintry"[1].
- If Canada were to enter, say, the NWERC region, that would be a "Canadantry".
- And whenever a person named "Paul" enters someplace, we clearly have a "Pauntry".

Given a subject's name, determine how the act of entering should be called for this subject.

Input

The input consists of:

- One line with a string s ($1 \leq |s| \leq 50$), the name of the subject. This name can refer to any person, animal, country, organization, etc.

All characters in s are uppercase letters `A-Z` or lowercase letters `a-z`. The first letter may be uppercase or lowercase, all other letters are lowercase. s will contain at least one lowercase vowel.

Output

Output one single word, the term for the act of the subject entering.

Sample Input 1	Sample Output 1
`Britain`	`Britaintry`

Sample Input 2	Sample Output 2
`Canada`	`Canadantry`

Sample Input 3	Sample Output 3
`Paul`	`Pauntry`

[1] We are convinced this is going to happen, because Brexit must have been such a joy for the people of Britain (otherwise, their political leaders certainly would not have taken this step). When something brings so much joy to everybody, you want to repeat it. But to repeat Brexit, there has to be Britaintry first!

This page is intentionally left (almost) blank

Problem K: Knightly Knowledge

Ancient cultures had an enormous and mostly unexpected knowledge of the earth and geometry. When they were building monuments, they did not place them at arbitrary locations. Instead, each building spot was carefully selected and calculated. Nowadays we can observe this in the form of so-called *ley lines*. A ley line is a horizontal or vertical straight line of infinite length that runs through at least two ancient monuments. Ley lines are expected to be a source of some kind of magic energy and every church built along such a line is a *mighty church*.

There are already several monuments and churches. An ancient culture is planning to construct a new monument, but the location is yet to be determined. They are looking for the spot that will turn the most ordinary churches into mighty churches. The monument can be co-located with a church or an existing monument – in that case, the new monument will simply be built around the church or monument.

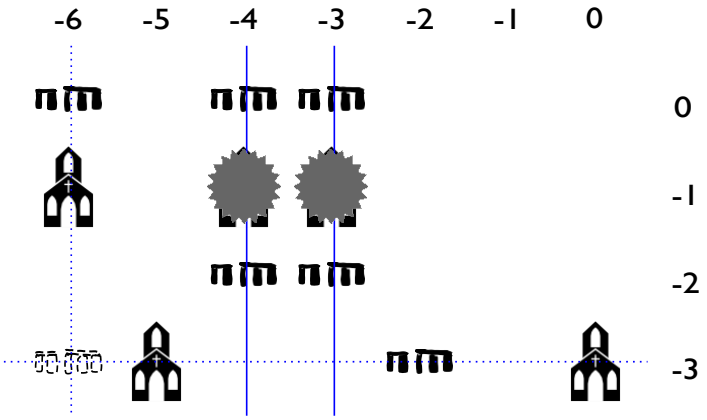


Figure K.1: Illustration of Sample 2 with 2 ley lines (|), 6 monuments ( ), 2 mighty churches ( ) and 3 ordinary churches ( ) that are turned into mighty churches when building the dashed monument.

Input

The input consists of:

- One line with two integers m and c ($0 \leq m, c \leq 1\,000$), the number of already built monuments and churches.
- m lines with the coordinates of the monuments.
- c lines with the coordinates of the churches.

All coordinates are given as two integers x and y ($-10^6 \leq x, y \leq 10^6$). None of the given coordinate pairs coincide with one another, but any may coincide with the location of the new monument.

Output

Output three integers: the coordinates for where to build the new monument and the number of ordinary churches that will be turned into mighty churches with this new monument. The coordinates should be in the range ($-10^6 \leq x, y \leq 10^6$). If there are multiple optimal solutions, you may output any one of them.

Sample Input 1

```
2 3
0 5
5 0
0 1
0 3
3 0
```

Sample Output 1

```
0 0
3
```

Sample Input 2

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

Sample Output 2

```
-6 -3
3
```

Sample Input 3

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

Sample Output 3

```
0 0
2
```

Problem L: Lexicographical Lecturing

Unsorted strings by bethrosengard on Pixabay

The OUG ("Ordered University of Germany") is a renowned German university. Since it has a lot of students, student IDs are quite long strings of equal length ℓ, where each student ID only contains lowercase letters of the English alphabet. Unfortunately for the students, the university's president hates chaos and expects students to always enter lecture halls in ascending lexicographic order of their IDs. As you can imagine, the process of sorting themselves in front of the lecture hall takes quite a lot of time for the students. Georgina, a computer science student, has the following idea to accelerate this process: She plans to fix two integers i, j with $1 \leq i \leq j \leq \ell$ denoting a substring of the student ID starting at the ith letter and ending in the jth letter. Students then sort themselves lexicographically with respect to this substring of their student ID. Of course, i and j must be chosen in a way such that this new ordering is equal to the lexicographic ordering with respect to their complete IDs. In order to make the process as fast as possible, the length of the substring should be minimal. Can you help Georgina to solve this problem?

Input

The input consists of:

- One line with two integers n and ℓ, where
  - n ($2 \leq n \leq 500$) is the number of student IDs;
  - ℓ ($1 \leq \ell \leq 2 \cdot 10^4$) is the length of each student ID.
- n lines, the ith of which contains the student ID of the ith student.

All student IDs only contain lowercase letters of the English alphabet, they are pairwise distinct and appear in ascending lexicographic order.

Output

Output two integers denoting the indices of the first and last letter of the shortest substring so that when students sort themselves lexicographically with respect to this substring of their student ID, the same order establishes as if students sorted themselves lexicographically with respect to their complete student ID.

If there are multiple shortest substrings, you may output any one of them.

Sample Input 1	Sample Output 1
4 6 aaaaaa aaabbb aaacaa aaacac	4 6

Sample Input 2	Sample Output 2
3 5 ccccca ccgda ccgia	4 4

Problem M: Mixtape Management

Mary has created a mixtape with her favourite reggae tracks. The mixtape consists of a list of MP3 files on her computer that she wants to share with her friends Wendy and Larry. However she knows that her friends have different musical tastes and will therefore also have different preferences for the order in which the tracks are played.

Mary knows that Wendy is a Windows user and Larry is a Linux user and realised that she can use this to her advantage. This is because Windows and Linux use different methods to sort files within a directory in case their names contain numerical data. In Windows, numbers in file names are read as actual numbers, causing the files to be sorted by increasing values of these numbers. In Linux, there is no special handling for numbers, so file names are sorted lexicographically. See Figure M.1 for an example of file sorting on the two operating systems.

Linux	Windows
`337.mp3`	`7.mp3`
`34.mp3`	`34.mp3`
`3401.mp3`	`79.mp3`
`7.mp3`	`337.mp3`
`780.mp3`	`780.mp3`
`7803.mp3`	`3401.mp3`
`79.mp3`	`7803.mp3`

Figure M.1: Illustration of the first sample case. Note that the file extensions `.mp3` do not influence the ordering and are purely for illustration.

After deciding on the order in which she wants Wendy and Larry to listen to the tracks, Mary has already sorted the files according to Larry's taste. Now she wants to rename the files such that the filenames are distinct positive integers without leading zeroes, they are sorted in increasing lexicographic order, and when sorting them by value the new order will match Wendy's taste. For this purpose, she has come up with a permutation $p_1, \ldots, p_n$ that describes how to rearrange Larry's list into Wendy's list: for every i, the ith number in lexicographic order needs to be the p_ith smallest by value. Help Mary find a suitable sequence of filenames.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 100$), the number of tracks.
- One line with n distinct integers $p_1, \ldots, p_n$ ($1 \leq p_i \leq n$ for each i), the given permutation.

Output

Output n distinct integers in lexicographically increasing order, your sequence of filenames. All numbers must be positive integers less than 10^{1000} and may not contain leading zeroes. Any valid sequence of filenames will be accepted.

Sample Input 1	Sample Output 1
`7` `4 2 6 1 5 7 3`	`337 34 3401 7 780 7803 79`

Sample Input 2	Sample Output 2
4 4 1 3 2	234 6 87 9