FAU Wintercontest 2019
January 26th

Problems
A  A Race Against Time  (easy)
B  Bongo Bongo Chess
C  Cryptanalysis
D  Drawing Numbers  (easy)
E  Elementary
F  Four-Letter Words
G  Game Historian
H  Height Differences
I  Infiltration
J  Journey Witnesses
K  Kings’ Meeting
L  Local Etiquette  (very easy)
M  Maharajah

Sponsored by
This page is intentionally left (almost) blank.
Every time traveller knows the Time Pirates, vicious lawbreakers changing key points in time for their own personal gain, and now it seems like your previous encounters with them have given you a cosy top spot on their hit list. They have launched a series of coordinated strikes against you by travelling through time and causing problems both in your past and in your future. However, since time is not a linear “cause and effect” kind of thing, but more like a big ball of wibbly wobbly timey wimey stuff, these attacks do not take full effect immediately. Instead, they will appear gradually, and you have the luxury of being able to choose in what order you want to fix the problems – but you do have to fix them all!

For each problem, you have already determined the base time of how many hours it will take to solve if you start working on it immediately. Additionally, you estimate that the time needed to resolve an untouched issue increases at a rate of 1% of the base time per hour. After solving any issue, you will need exactly 1 hour to tie up loose ends and get back into your time machine. (For instance, if you start fixing an issue with a base time of 5 hours, and you have already spent 123.4 hours fixing other issues, it will take $5 + 0.01 \cdot 5 \cdot 123.4 + 1 = 12.17$ hours to solve this issue and return to your time machine.)

Naturally, solving a problem partially just to leave and return later would just confuse the locals, so you will always completely fix an issue once you have started working on it before moving on to the next one. Now you just need to determine in what order you should solve the problems to minimise the time it will take you in total.

Assuming it takes no time to travel in your time machine, how long will it take you to fix all problems and return to your time machine?

**Input**

The input consists of:
- One line with an integer $n$ ($1 \leq n \leq 100$), the number of problems caused by the Time Pirates.
- One line with $n$ integers $t_1, \ldots, t_n$ ($1 \leq t_i \leq 12$ for each $i$), the base times (in hours) required to solve the individual problems.

**Output**

Output a single number, the minimum number of hours it will take you to solve all the problems and return to your time machine. Your output will be accepted if the absolute or relative error does not exceed $10^{-6}$.

**Sample Input 1**

```
2
2 1
```

**Sample Output 1**

```
5.03
```

**Sample Input 2**

```
3
3 5 2
```

**Sample Output 2**

```
13.3836
```

**Sample Input 3**

```
6
12 12 11 12 12 12
```

**Sample Output 3**

```
103.6716
```
This page is intentionally left (almost) blank.
You travel to the distant year 3019. All animals have learned to talk in human languages and live together in harmony. One of the most popular pastimes in Bongo Bongo Jungle, where you are staying over the weekend, is to play *Friendship Chess*: a chess variant played in teams by passing the board around.

The players are all sitting in a circle, with each player sitting between two players of the opposing team. After every move, the chess board is kindly handed to the next player in the circle, so that turns alternate between the two teams.

The odd number of animals you want to play with are already arranged in a circle, waiting for you to join. When you pick a place to sit, teams will be determined as described above, so you have some control over who will be in your team.

You’d really like to win, so you decide to check all the animals’ social media to look up their *Animelo Rating*, a number that indicates how good someone is at chess. While you don’t know the exact details of how this rating system works, you strongly believe that it’s probably best to sit such that the sum of your teammates’ Animelo Ratings is as high as possible. What’s the highest that sum can be?

**Input**

The input consists of:

- One line with an odd integer $n$ ($3 \leq n \leq 456789$), the number of animals playing.
- One line with $n$ integers $s_1, \ldots, s_n$ ($1 \leq s_i \leq 1000$ for each $i$), giving the Animelo Ratings of the animals in the order they’re sitting in.

**Output**

Output the maximum possible sum of your teammates’ Animelo Ratings.

**Sample Input 1**

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

**Sample Output 1**

```
9
```

**Sample Input 2**

```
3
1 1000 1
```

**Sample Output 2**

```
1000
```
This page is intentionally left (almost) blank.
Problem C: Cryptanalysis

AVTCCMIPTJEOOTNKNI – this is the first thing you read on a small note during a visit to the 16th century. There are plenty of similar notes on the table, all with unreadable texts written on them. Very soon you realise that you are standing inside an office for cryptanalysis.

One guy explains to you what they are currently working on:

“We intercepted hundreds of these military notes from our enemy, the Bellaso syndicate. A command given by their leader will eventually be spread around to all members, but for different recipients it will be encrypted with different keys. We may be lucky and own several notes that contain the same message but are encrypted with different keys. This would help break the code. Our plan is to separate the notes onto different piles, with each pile consisting of notes that might contain the same hidden message.”

How do you know which messages belong to the same group? And how were they encrypted?

```
BLIZZARD
+253 253 25
= DQL BEDIT

BLIZZARD
+977 977 97
= KSPIGHAK
```

Figure C.1: Encrypting the word BLIZZARD with two different keys (2, 5, 3) and (9, 7, 7).

“As the Vigenère cipher is quite new and unbreakable, we are confident that all notes are encrypted using this cipher.

As shown in Figure C.1, a message is encrypted by first repeating the secret key and aligning it with the message, and then shifting each letter the appropriate amount, wrapping around from Z to A if necessary.

We already managed to get the key length $k$ they use, so we can use this information to find out which messages belong together. For instance, both words DQLBEDIT and KSPIGHAK can result from the same original message and therefore go on the same pile. There is still the chance that, for example, KSPIGHAK is the result of a different message with a different key, but we do not yet care about that.”

You realise that cryptanalysis was a really frustrating job back in this time with no computers that could solve this problem in seconds...

**Input**

The input consists of:

- One line with two integers $n$ and $k$ ($1 \leq n \leq 5 \cdot 10^4$, $1 \leq k \leq 50$).
- $n$ lines, each with one word $w$ ($1 \leq |w| \leq 50$) consisting of uppercase letters A-Z. All words are distinct and have the same length $|w| \geq k$.  

FAU ICPC 2019 – Problem C: Cryptanalysis 5
Output

Output one line for each pile, containing all messages that go on that pile. The order of piles does not matter, nor does the order within a pile.

<table>
<thead>
<tr>
<th>Sample Input 1</th>
<th>Sample Output 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 3</td>
<td>ANBEJDSI</td>
</tr>
<tr>
<td>DQLBEDTI</td>
<td>DQLBEDTI KSPIGHAK</td>
</tr>
<tr>
<td>KSPIGHAK</td>
<td></td>
</tr>
<tr>
<td>ANBEJDSI</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample Input 2</th>
<th>Sample Output 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2</td>
<td>AVTCCMIPTJEOOTNKNI BAUHDRJUUOFTPYPON</td>
</tr>
<tr>
<td>AVTCCMIPTJEOOTNKNI</td>
<td>BAGVVYNLFAJUUOFIBY</td>
</tr>
<tr>
<td>BAUHDRJUUOFTPYPON</td>
<td></td>
</tr>
<tr>
<td>BAGVVYNLFAJUUOFIBY</td>
<td></td>
</tr>
</tbody>
</table>
You decide to travel to the future and check up on your hundred-year-old self. Future-you is playing bingo, and since they have misplaced their glasses, you decide to help them out with the game.

They have a card with 25 distinct numbers on it, arranged in a $5 \times 5$ square. The host of the game calls out random numbers, and you mark each called number that future-you has on their card. When a player has all five numbers of a row, column, or diagonal marked, they get to yell “Bingo!”

Future-you is really looking forward to that, so they ask you how many more numbers are required in the best-case scenario, and which numbers those are.

**Input**

The input consists of five lines, each with five integers $b$ ($0 \leq b \leq 100$ for each $b$), representing the bingo card. A 0 represents a number already drawn and marked. The positive integers (numbers yet to be drawn) are distinct.

**Output**

Output an integer $a$ – the minimum amount of numbers that need to be drawn to complete a row, column, or diagonal – followed by the $a$ numbers required (in any order). If multiple combinations of $a$ numbers are applicable, any of them will be accepted.

**Sample Input 1**

```
65 50 80 20 15
30 0 35 0 0
0 10 70 55 45
95 85 0 25 0
0 60 40 90 75
```

**Sample Output 1**

```
2
35 30
```

**Sample Input 2**

```
0 3 2 0 0
4 0 0 0 12
5 0 1 0 11
6 0 0 0 10
0 7 8 9 0
```

**Sample Output 2**

```
1
1
```

Figure D.1: Illustrations of the first three samples and their valid solutions.
### Sample Input 3

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Sample Output 3

0

### Sample Input 4

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

### Sample Output 4

5

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem E: Elementary

Even time travellers need money for food, so unfortunately you have to have a day job. Since using time travel for personal gain is prohibited and you want to be a lawful time traveller, you found a start-up instead, in the year 3018. Why 3018? Well, it was around that time that mankind discovered how to safely create and contain all the elements of the periodic table\(^1\) in pure form. And you need that for your business idea: selling words created from elements.

Now the only thing left for you to do is to check if and how any given word can be spelled out using element symbols.

### Input

The input consists of:

- One line with one string \(s\) (\(1 \leq |s| \leq 1\,000\)) consisting of lowercase letters a-z.

### Output

If there is no solution, output `impossible`. Otherwise, output a sequence of element symbols that spell out the given string \(s\). If there is more than one solution, any one of them will be accepted.

**Sample Input 1**

```
fauerlangen
```

**Sample Output 1**

```
F Au Er La N Ge N
```

**Sample Input 2**

```
wintercontest
```

**Sample Output 2**

```
impossible
```

**Sample Input 3**

```
prognosis
```

**Sample Output 3**

```
Pr Og N O Si S
```

---

\(^1\)You may find the following code snippet useful, which is also available in the samples.zip archive that can be downloaded via the DOMjudge web interface. These are all 118 element symbols:

```
```
This page is intentionally left (almost) blank.
Problem F: Four-Letter Words

Writing emails to friends can be tricky for a time traveller. Not only do you have to use the correct email address but the correct date as well. And you have to make sure that you don’t send a follow-up email to an earlier date. The Improved Chronological Private Chat is a far superior tool for having a conversation. It connects to chat rooms of different eras and ensures that the chronological order of the conversation stays intact even when travelling through time.

After trying to send a message to your newest friend Morgan, you are prompted with the following error: MESSAGE CONTAINS THE BANNED WORD; MESSAGE REJECTED. Poking around a bit reveals that the chat room you are connected to has a poorly implemented word filter. It filters out whole messages if they contain a certain blacklisted word as a substring. Poking around a bit more reveals that this forbidden word is a string consisting of four letters, but you do not know the exact string yet.

Certainly there must be a way to find out the banned word while adhering to the size limits of chat room messages and without accidentally spamming your new friend too much.

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 must send requests of the following two types:

- One line of the form “? q”, where q is a string consisting of lowercase letters a-z (1 ≤ |q| ≤ 10^4). The grader will respond yes if the banned word is a substring of q, and no otherwise.
- One line of the form “! w”, where w is a string consisting of exactly four lowercase letters, the forbidden word determined by your submission.

After every request 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, sys.stdout.flush() in Python, hFlush stdout in Haskell, and CALL FLUSH() in Fortran.

Your submission may send at most 100 requests of the first type.

Your submission must send exactly one request of the second type. After sending this request, it must terminate with exit code 0 as usual.

Your submission will be accepted if it follows the protocol above and it guesses the forbidden word correctly. If it sends any invalid request or guesses the forbidden word incorrectly, it will be judged as “Wrong Answer”. You may safely assume that there is always a correct solution.

Note that for testing purposes, the forbidden word is not necessarily fixed at the start of the interaction, but may be set by the grader at any point (but always in a way that is consistent with its answers to previous requests).
<table>
<thead>
<tr>
<th>Sample Input 1</th>
<th>Sample Output 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>? checkmate</td>
</tr>
<tr>
<td>no</td>
<td>? mate</td>
</tr>
<tr>
<td>no</td>
<td>? icpc</td>
</tr>
<tr>
<td>yes</td>
<td>? check</td>
</tr>
<tr>
<td>no</td>
<td>? chec</td>
</tr>
<tr>
<td>! heck</td>
<td></td>
</tr>
</tbody>
</table>

In the example interaction above, the output of the grader is on the left and one possible correct output of a submission is on the right. The empty lines on both sides only serve to emphasise the chronological order of requests and answers. The grader will never output any empty lines.
Problem G: Game Historian

You are travelling through ancient Rome in the first century BC. You notice a lot of chalk markings on the walls that you quickly identify as tic-tac-toe games. As you can recall from a history lesson, the Romans actually invented the popular game. You get really excited about the possibility to interact with people from this time (and to beat them in a game to show your intellectual superiority).

It is common knowledge that, if played properly, it is always possible to end a game of tic-tac-toe in at least a draw. However, on your way to your first opponent, you get nervous. What if you make a wrong move and suddenly lose? That would be extremely embarrassing. So instead of trusting your own abilities, you decide to create a program that will never lose (ties are fine). Fortunately, you have already convinced your waiting opponents to let you make the first move in all games, which should give you an advantage.

As a reminder, here are the rules of tic-tac-toe:

- Two players take turns marking the spaces in a $3 \times 3$ grid (with the first turn being yours).
- Your mark is an \(\times\), your opponent’s is an \(\circ\).
- A player chooses exactly one (as of yet unmarked) space during their turn.
- As soon as a row, column, or diagonal contains three \(\times\) marks, the game ends and you win.
- As soon as a row, column, or diagonal contains three \(\circ\) marks, the game ends and you lose.
- If all nine spaces are marked but neither player’s win condition applies, the game ends in a tie.

![Figure G.1: Visualisation of the given sample.](image)

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 moves, with your submission going first. A move is a line with a single integer \(s\) (\(1 \leq s \leq 9\)) specifying the space marked by the player, corresponding to the numbering in Figure G.1. You can safely assume that the grader will only perform valid moves.

After every move you should flush the standard output to ensure that the move is sent to the grader. For example, you can use `fflush(stdout)` in C++, `System.out.flush()` in Java, `sys.stdout.flush()` in Python, `hFlush stdout` in Haskell, and `CALL FLUSH()` in Fortran.

The game ends as soon as one of the two players manages to complete a row/column/diagonal or all cells have been marked. After the game ends, your submission should terminate with exit code 0 as usual. For convenience, the grader will send a single \(-1\) to signal the end of the game, which your submission may or may not read.
Your submission will be accepted if it follows the protocol above and achieves a win or a tie. If it makes an illegal move (i.e. writes anything other than a cell number or tries to move to a cell that has already been marked), it will be judged as “Wrong Answer”.

Note that for testing purposes, your submission will still be run multiple times, and the grader will not make the same moves each time.

<table>
<thead>
<tr>
<th>Sample Input 1</th>
<th>Sample Output 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>−1</td>
<td>1</td>
</tr>
</tbody>
</table>

In the example interaction above, one possible output of the grader is on the left and one possible correct output of a submission is on the right. The empty lines on both sides only serve to emphasise the chronological order of requests and answers. The grader will never output any empty lines.
Problem H: Height Differences

By the 23rd century, the general public has taken note of your time travelling ways and problem solving proficiency. As a result, you were asked to be in Skyscraper City on the 26th of January, 2219.

While Skyscraper City is a huge metropolis, there is just one street you need to concern yourself with. Viewed from above, its buildings form a straight line, and the distance between adjacent buildings’ doors is always exactly 1 skyscraper unit. What’s weird is that each building has one inhabitant who spends their days overlooking the cityscape from the rooftop. The other residents of Skyscraper City have decided to call them “rooftop people” and to analyse their behaviour further.

As it turns out, rooftop people occasionally want to see the city from a rooftop of a different height. When this happens, they walk along their street to the nearest building whose height differs enough from the height of their own building (regardless of whether the new roof will be higher or lower than their home roof, and they don’t care about the time spent riding elevators).

On that fateful 26th of January, the unthinkable is happening: all of the rooftop people are leaving their respective rooftops to see the city from a different height. To estimate the additional traffic, the city council has asked you to determine how many skyscraper units the rooftop people will walk in total.

Input

The input consists of:

- One line with two integers \( n \) and \( k \) (\( 2 \leq n \leq 500\,000 \), \( 1 \leq k \leq 10^9 \)), the number of buildings and the minimum height difference the rooftop people are looking for.
- One line with \( n \) integers \( a_1, \ldots, a_n \) (\( 1 \leq a_i \leq 2 \cdot 10^9 \) for each \( i \)), the heights of the buildings from left to right.

You may safely assume that for each building, there is another building such that the height difference between the two is at least \( k \).

Output

Output the sum of the minimum number of skyscraper units that each rooftop person has to walk in order to reach a building whose height differs from their home roof by at least \( k \).

Sample Input 1

```
6 5
1 3 7 6 6 8
```

Sample Output 1

```
19
```

Sample Input 2

```
2 1
100 10
```

Sample Output 2

```
2
```
This page is intentionally left (almost) blank.
The last adventures took a toll on you, so you decide to visit a hotel in the year 2634 that was advertised to you at the last time traveller convention. A week of wellness and comfort is ahead of you. Well, at least that’s what you were promised.

As it turns out, the hotel is managed by a rich collector who decided to collect time travellers by trapping them in a computer simulation forever. Luckily, you saw through the facade and managed to avoid being captured. Now, after crawling through vents for a while, you find yourself in the hotel’s server room where you inspect what you are up against.

The simulation is upheld by a network of servers. Every server in the network is directly or indirectly connected to every other server. You can also see that there is at most one direct connection between any two servers, and no server is directly connected to itself. In other words, the network is a connected undirected graph without multi-edges or self-loops.

To free all the remaining poor souls, each server must be manually switched off using a lever. Since it is physically impossible for you to pull all levers at once, you have to deactivate one server at a time. But pulling the levers in any arbitrary order will not be enough! The remaining servers still have to form a network. If at any point two or more separated networks coexist, their respective simulation states will desynchronize, which could seriously harm the still imprisoned. Find a valid order that lets you shut down the servers without harming the trapped time travellers!

Input
The input consists of:

- One line with two integers \( n \) and \( m \) (\( 1 \leq n \leq 10^5, 0 \leq m \leq 2 \cdot 10^5 \)), the number of servers and the number of connections. The servers are numbered from 1 to \( n \).
- \( m \) lines, each with two integers \( a \) and \( b \) (\( 1 \leq a, b \leq n \)), describing a connection between servers \( a \) and \( b \).

The given graph is connected and does not have any self-loops or multi-edges.

Output
Output one line with \( n \) integers, specifying a safe order in which to shut down the servers. If there is more than one solution, any of them will be accepted.

Sample Input 1
4 3
3 2
1 3
3 4

Sample Output 1
1 4 2 3

Sample Input 2
5 8
1 2
1 3
1 4
2 3
2 4
3 4
3 5
4 5

Sample Output 2
3 5 2 1 4
This page is intentionally left (almost) blank.
Problem J: Journey Witnesses

It is June 23rd, 2073. Your friend Riley just returned from a journey through time. Normally, time travelling is very time consuming (that’s only fair). However, Riley has modified her time machine slightly, hard-wiring certain locations and directional paths between them. Now, whenever there is a so-called time wire going from her current location to another, she can time travel to that location in just one minute (measured in accurate absolute time – AAT). The downside to mods using time wires is that they prevent all other location changes – Riley always has to travel along the wires now.

This last trip of hers was a flash visit trip: Each location she visited she jumped out of the machine, took a picture and went back in to continue to the next location. She now proudly presents you with all her photographs, showing various famous cities, boasting about how amazing her time wire mod is and that she changed location every minute. Knowing her, you have a hunch that she may be making some stuff up, and decide to investigate.

![Figure J.1: All photos that you consider in Sample 1. From left to right (year, AAT): Paris (2073, 0), Rome (34 BC, 2), Tokyo (2015, 5), Toyko (2142, 7), Lima (1993, 8).](image)

A small trick and now you are in possession of all the photos. For some of them, it is impossible to determine where they were taken. You discard those. The remaining ones are useful not just because of the cities shown, but also because each one contains a unique timestamp placed there by the camera. These timestamps indicate when the photos were actually taken in AAT, because the time chip in the camera is not affected by time travelling. This might suffice to prove Riley a liar!

Compare the photos’ timestamps with the time machine’s time wires to detect how many photos she went without lying (it’s nice to phrase things positively, after all).

Input

The input consists of:

- One line with two integers \( l \) and \( w \) (\( 2 \leq l \leq 250, 1 \leq w \leq 10^5 \)), giving the number of hard-wired locations and the number of time wires, respectively.
- \( w \) lines, each with two strings \( a \) and \( b \) (\( 1 \leq |a|, |b| \leq 20, a \neq b \)), indicating that a time wire exists to rapidly travel from \( a \) to \( b \). All wires are unique.
- One line with a single integer \( p \) (\( 1 \leq p \leq 10^3 \)), giving the number of usable photos. They are numbered from 1 to \( p \) in increasing AAT order.
- One line with \( p \) strings \( s_1, \ldots, s_p \), indicating that the \( i \)th photo was taken in city \( s_i \) (for each \( i \)). It is guaranteed that all \( s_i \) appear in the list of time wires.
- One line with \( p \) integers \( t_1, \ldots, t_p \) (\( 0 \leq t_1 < t_2 < \ldots < t_p \leq 10^{12} \)), indicating that photo \( i \) has an AAT timestamp of minute \( t_i \) (for each \( i \)).

All location names consist of lowercase letters \( a-z \).
Output

Assuming Riley started her flash visit trip in the city shown in the first photo (at the given time),
print the maximum $i$ such that all photos between the first and $i$th are consistent with the given
information.

Explanation Sample 1

AAT 0: Riley started her flash visit trip in Paris.
AAT 1: She must have been in Tokyo, but the photo just shows a beautiful tree. This could be
everywhere on earth, so you discarded this photo.
AAT 2: She visited Rome, which can be proved by the second photo in the list.
After jumping between Rome and Tokyo, she managed to be in Tokyo at AAT 5 and again at 7.
But the next photo clearly shows Lima at AAT 8, and there is no way to get there from Tokyo
that fast.
Perhaps there is another time wire that goes from Tokyo to Lima – you confront her with the
fact that only the first 4 photos are consistent with her story.

Sample Input 1   Sample Output 1
4 4
paris tokyo
tokyo rome
rome tokyo
rome lima
5
paris rome tokyo tokyo lima
0 2 5 7 8

4
Problem K: Kings’ Meeting

You have heard a lot about the majestic castles of the Middle Ages. Sadly, most of the castles were destroyed during the many wars of the time. Instead of visiting old ruins, you decide to travel to the Middle Ages for an authentic sightseeing tour.

Upon arrival you find yourself in the middle of a heated argument between three kings.

The first king explains: “Each of us owns a castle, but we are unable to agree on the sizes of our kingdoms. A kingdom is defined by a number \( l \) and the location of the corresponding castle. Everything within a distance of less than \( l \) from the castle will be part of that kingdom. Our kingdoms do not necessarily have to be of the same size, but none of them should be empty. Also, calculating square roots is really hard for us. So instead of using the distance \( \sqrt{d_x^2 + d_y^2} \) between two points whose \( x \) and \( y \) coordinates differ by \( d_x \) and \( d_y \), we simply build the sum of the horizontal and the vertical difference, i.e. \( d = |d_x| + |d_y| \).”

The second king goes on to say that “all points with a distance of exactly \( l \) from a castle form the border of the corresponding kingdom. It does not matter if the kingdoms’ borders touch, so long as the kingdoms themselves do not overlap one another!”

Finally, the last king adds: “If we want to maintain peace, we must agree on a location \( s \) where we can meet from time to time to discuss important matters. This location must lie on the border of all three kingdoms, so that each of us can travel there safely. Please help us find such a location – otherwise, we will have no choice but to go to war again and destroy our beautiful castles in the process.”

Input

The input consists of three lines specifying the location of the three castles. Each line contains two integers \( x \) and \( y \) \((-10^{10} \leq x, y \leq 10^{10})\), giving the location of one castle. No two castles share the same location.

Output

If it is possible to form kingdoms that satisfy all conditions laid out above, output a single line containing the \( x \) and \( y \) coordinates of \( s \). Otherwise, print \textit{impossible}.

Sample Input 1
0 0
3 3
5 1

Sample Output 1
3 1

Sample Input 2
0 0
3 0
4 2

Sample Output 2
impossible
This page is intentionally left (almost) blank.
Problem L: Local Etiquette

Time travelling is exhausting, so regular breaks are essential for a good time travel experience. To combine your break with a new adventure, you decide to grab your next bite in medieval England.

You visit the best pub in the historic city of Oxford and sit together with some knights. Being unfamiliar with the local etiquette is very embarrassing, so you tell them that you are a merchant from the South and ask if they can show you how to eat appropriately. A bit confused, but always helpful, the knights agree and show you the basics of the regional Oxford etiquette. The biggest difference between the local etiquette and the one in the South – at least according to the knights – is that after eating the meat from the bones, you don’t put the bones back on the table, but just throw them on the floor backwards. You think that this is a bit disgusting, but don’t want to question it any further, so you behave as required.

After some great dining with the knights, they tell you of another custom of the English knights of Oxford. Instead of everybody paying for themselves, they play a betting game to decide who will have to pay the bill tonight. In this game, everyone has to guess the number of bones on the floor that are left over from the meal. The one with the worst bet – meaning the one which has the highest absolute difference to the correct number – has to pay. To prevent draws as much as possible, all guesses must be distinct integers. If a draw occurs nonetheless, the involved bidders split the bill. Sometimes the knights are very hungry and sometimes they do not eat anything, so the actual number of bones on the floor may be any positive integer or even 0. As you are the guest and the knights are really honourable, everyone else gives their guess before you give yours.

You have read some horrifying stories about English knights and suspect that the knights are trying to trick you into paying. Is there a safe guess you can place that ensures you will not have to pay at all, regardless of the number of bones on the floor?

Input

The input consists of:

- One line with an integer \( n \) (1 \( \leq \) \( n \) \( \leq \) 1 000), the number of knights.
- One line with \( n \) integers \( a_1, \ldots, a_n \) (0 \( \leq a_i \leq 10^9 \) for each \( i \)), the knights’ guesses.

Output

Output any number that constitutes a safe guess, or \text{impossible} if there is no such number. Only integers between 0 and 10^9 (both inclusive) are valid guesses.

Sample Input 1

\[
\begin{align*}
3 \\
1 & 4 & 2 
\end{align*}
\]

Sample Output 1

\[
3 
\]

Sample Input 2

\[
\begin{align*}
1 \\
4 
\end{align*}
\]

Sample Output 2

\[
\text{impossible} 
\]

Sample Input 3

\[
\begin{align*}
2 \\
0 & 10 
\end{align*}
\]

Sample Output 3

\[
7 
\]
This page is intentionally left (almost) blank.
You have travelled to the 18th century to visit the Maharajah of Confusistan, who has been a dear friend of yours ever since you saved him from a tiger attack when he was young.

The Maharajah owns a huge treasure which is safely kept inside his mighty palace. The palace is surrounded by a circular wall that several security guards have been assigned to. To ensure better protection of the treasure, the guards change their positions every night.

The guards change their positions according to the following scheme: there are $m$ guards and $m$ positions, numbered from 1 to $m$, as well as a permutation $p$. Each night, the guard who took position $i$ the previous night moves to position $p_i$, for each $i$.

His advisors tell you that the Maharajah has grown increasingly paranoid since your last visit – he ordered the construction of a second circular wall, to be built around the existing wall. $n$ guards are to be assigned to this second wall, who will also rotate their positions according to a fixed permutation.

It is not hard to see that for each permutation, there must be a night in which all $m + n$ guards are back in the same position that they were in the first night. This is a weak point in the security concept, as it makes it easier for thieves and looters to look for inattentiveness in the guards’ behaviour.

The Maharajah, too, is aware of this, so he has ordered that the permutation for the $n$ new guards be chosen so that it will take as long as possible for this to happen. Since he knows you are quite proficient at problem solving, he asked you to find such a permutation.

**Input**

The input consists of:

- One line with two integers $m$ and $n$ ($1 \leq m, n \leq 3000$), the number of guards on the inner and outer wall, respectively.
- One line with $m$ integers $p_1, \ldots, p_m$, ($1 \leq p_i \leq m$ for each $i$, the $p_i$ are distinct), the permutation for the guards on the inner wall.

**Output**

Output $n$ integers $p_{m+1}, \ldots, p_{m+n}$ ($m + 1 \leq p_i \leq m + n$ for each $i$, the $p_i$ are distinct), a permutation for the guards on the outer wall such that the time it takes for all $m + n$ guards to return to their original positions is maximised. If there is more than one solution, any one of them will be accepted.

**Sample Input 1**

```
3 7
2 3 1
```

**Sample Output 1**

```
10 7 9 8 4 6 5
```
This page is intentionally left (almost) blank.