Northwestern European Regional Contest 2016

NWERC 2016

Bath, November 20

Problems

A Arranging Hat
B British Menu
C Careful Ascent
D Driving in Optimistan
E Exam Redistribution
F Free Weights
G Gotta Nudge 'Em All
H Hamiltonian Hypercube
I Iron and Coal
J Jupiter Orbiter
K Kiwi Trees

Do not open before the contest has started.

IBM J.P.Morgan

This page is intentionally left (almost) blank.

Problem A
Arranging Hat

Arranging Hat is a cushy job indeed; high impact work, absolute authority, and 364 days of holiday every year. However, the hat has decided that it can do even better—it would like very much to become a tenured professor.

The Arranging Hat. Image by Lisa Abose

Recently the hat has been reading computer science papers in its ample spare time, and of course, being an arranging hat, it is particularly interested in learning more about sorting algorithms.

The hat's new contribution is to a class of algorithms known as *lossy* sorting algorithms. These usually work by removing some of the input elements in order to make it easier to sort the input (e.g., the Dropsort algorithm), instead of sorting all the input.

The hat is going to go one better—it is going to invent a lossy sorting algorithm for numbers that does not remove any input numbers and even keeps them in their original place, but instead changes some of the digits in the numbers to make the list sorted.

The lossiness of the sorting operation depends on how many digits are changed. What is the smallest number of digits that need to be changed in one such list of numbers, to ensure that it is sorted?

Input

The input consists of:

- one line containing the integers n and m ($1 \leq n \leq 40$, $1 \leq m \leq 400$), the number of numbers and the number of digits in each number, respectively.
- n lines each containing an integer v ($0 \leq v < 10^m$). The numbers are zero-padded to exactly m digits.

Output

Write a sorted version of the array, after making a minimum number of digit changes to make the numbers sorted (the numbers must remain zero-padded to m digits). If there are multiple optimal solutions, you may give any of them.

Sample Input 1	Sample Output 1
5 3 111 001 000 111 000	001 001 001 111 200

Sample Input 2	Sample Output 2
15 3 999 888 777 666 555 444 333 222 111 222 333 444 555 666 999	199 288 377 466 555 644 733 822 911 922 933 944 955 966 999

Problem B
British Menu

Hevva Cake by Caitlin on Flickr, cc by

Since you are in Britain, you definitely want to try British food. Unfortunately you will only have a single free evening, so you decided to try all the food you can get in one run. You plan a gigantic meal where you eat one British dish after the other. Clearly not every order of dishes is appropriate. For example, it is not acceptable to eat Blood Pudding directly after Cornish Hevva Cake, but it would be perfectly fine if you chose to eat Baked Beans in between.

You have compiled a comprehensive list of British dishes. For each dish you have also decided which other dishes are fit to be eaten directly afterwards. A menu is a sequence of dishes such that each dish (except the first) is fit to be eaten directly after the previous dish.

After some time studying the list of dishes, you noticed something odd: Whenever it is possible to find a menu in which a dish occurs twice (for example dishes A, then B, then C, then dish A again), there can be at most four different types of dishes between the dish that occurred twice – excluding that dish itself. For example, it is impossible to find a menu like A, B, C, D, E, F, A, but it may be possible to find menus like $A, B, C, B, C, B, C, B, C, B, A$ or $A, B, C, D, E, A, B, C, D, E, A$.

But who wants to eat the same dish twice anyway? Clearly, you want to know how many dishes there can be in a menu without repeating any dish!

Input

The input consists of:

- One line with two integers n, m ($1 \leq n \leq 10^5$, $1 \leq m \leq 10^6$), the number of dishes and compatibilities.
- m lines, each containing two integers a and b ($1 \leq a, b \leq n$), indicating that you can eat dish b immediately after dish a.

Dishes are numbered from 1 to n in no particular order, and the compatibilities satisfy the constraint described above.

Output

A single integer indicating the maximum number of courses in a menu without repeated dishes.

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

Sample Input 2	Sample Output 2
7 7 1 2 2 3 3 4 4 5 5 2 4 6 5 7	6

Problem C
Careful Ascent

That went well! As police sirens rang out around the palace, Mal Reynolds had already reached his lifting device outside of the city.

No spaceship can escape Planet Zarzos without permission from the High Priest. However, Mal's spaceship, Firefly, is in geostationary orbit well above the controlled zone and his small lifting device can avoid being recognised as an intruder if its vertical velocity is exactly 1 km/min.

There are still two problems. First, Mal will not be able to control the vehicle from his space suit, so he must set up the autopilot while on the ground. The vertical velocity must be exactly 1 km/min and the horizontal velocity must be set in such a way that Mal will hit the Firefly on the resulting trajectory. Second, the energy shields of the planet disturb the autopilot: They will decrease or increase the horizontal velocity by a given factor. The original horizontal velocity is restored as soon as there is no interference. For this problem we consider Firefly to be a single point – the shape shown in Figure C.1 is merely for decorative purposes.

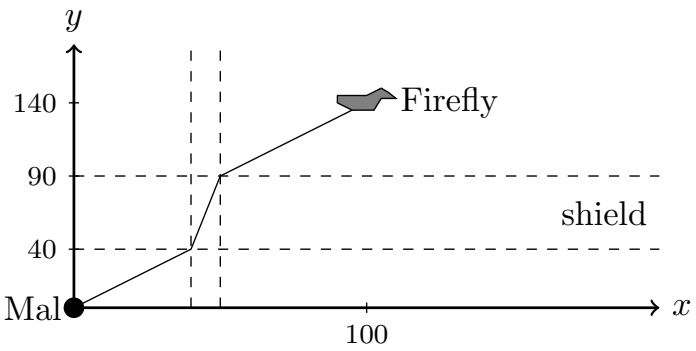


Figure C.1: Illustration of Sample Input 1.

Luckily, Mal recorded the positions of the shields and their influence on the autopilot during his descent. What he needs now is a program telling him the right horizontal velocity setting.

Input

The input consists of:

- one line with two integers x, y ($-10^7 \leq x \leq 10^7$, $|x| \leq y \leq 10^8$ and $1 \leq y$), Firefly's coordinates relatively to Mal's current position (in kilometres).
- one line with an integer n ($0 \leq n \leq 100$), the number of shields.
- n lines describing the n shields, the ith line containing three numbers:
  - an integer l_i ($0 \leq l_i < y$), the lower boundary of shield i (in kilometres).
  - an integer u_i ($l_i < u_i \leq y$), the upper boundary of shield i (in kilometres).
  - a real value f_i ($0.1 \leq f_i \leq 10.0$), the factor with which the horizontal velocity is multiplied during the traversal of shield i.

  It is guaranteed that shield ranges do not intersect, i.e., for every pair of shields $i \neq j$ either $u_i \leq l_j$ or $u_j \leq l_i$ must hold.

  All real numbers will have at most 10 digits after the decimal point.

Output

Output the horizontal velocity in km/min which Mal must choose in order to reach Firefly. The output must be accurate to an absolute or relative error of at most 10^{-6}.

Sample Input 1	Sample Output 1
100 140 1 40 90 0.2000000000	1.0

Sample Input 2	Sample Output 2
100 100 3 0 20 2.0000000000 50 100 0.1000000000 20 50 0.2000000000	1.96078431373

Problem D
Driving in Optimistan

Multi-directional distance sign. Picture licensed CC0 Public Domain.

Optimistan is a strange country. It is situated on an island with a huge desert in the middle, so most people live in port towns along the coast. As the name suggests, people of Optimistan (also called Optimists) like to optimise everything, so they only built roads necessary to connect all port towns together and not a single extra road. That means that there is only one way to get from one port town to another without visiting the same place twice.

The government installed multi-directional distance signs in 1-kilometre intervals on one side of the road, to provide important information to drivers. Thus whenever you go from one port town to another, you pass the first sign at the port town and then one each kilometre. Every distance sign contains the shortest distances to all port towns, each written on a separate small sign directed towards the goal town.

The signs also serve another important function: to guide drivers on intersections. This means that distance of each intersection from every port town is an integer number of kilometres.

You bought a tourist guide of Optimistan which does not have a map of the country, but it contains a huge table with the shortest distances between all pairs of port towns. You quickly calculated the average shortest distance between all pairs of port towns, but then you started wondering: if the signs also contained shortest distances to all other signs, what would be the average number written on a sign? Could this be calculated just from the distance table in the tourist guide?

Input

The input consists of:

- one line with an integer n ($2 \leq n \leq 500$), the number of ports;
- $n-1$ lines, the ith of which contains $n-i$ integers. The jth integer on the ith line denotes the distance between port i and port $i+j$ in kilometres. Each distance is between 1 and 10^6 (inclusive).

You can assume that the distances correspond to a road network in which there is exactly one path between two port towns that does not visit the same place twice. All roads can be used in both directions.

Output

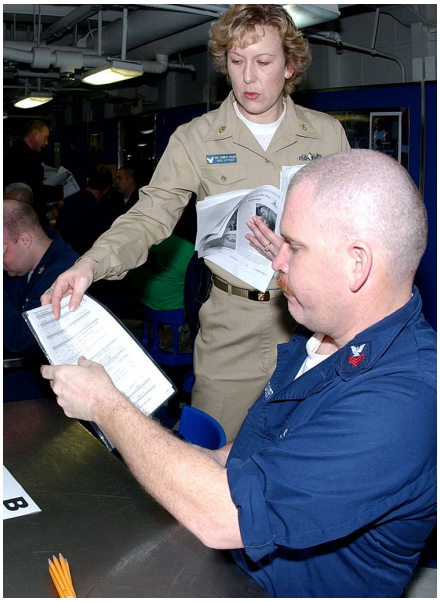
Output one line with the average distances in kilometres between all pairs of distance signs in Optimistan. Your answer should have an absolute or relative error of at most 10^{-9}.

If it is impossible to determine the exact average of distances between all pairs of distance signs in Optimistan, output "`impossible`".

Sample Input 1	Sample Output 1
3 4 4 2	2.13333333333333

Sample Input 2	Sample Output 2
4 2 2 2 2 2 2	1.6

Problem E
Exam Redistribution



Picture from US Navy, public domain

Linda is giving an exam. When the exam is over, Linda will redistribute the exams among the students for a peer review, so they may grade each other's answers and assign preliminary scores.

The students are split up in several rooms of varying sizes. Linda has devised the following scheme for redistributing the exams:

1. Linda visits the first room, picks up all exams written there, and places them in a pile.
2. In each subsequent room Linda takes exams from the top of her pile and randomly distributes them to the students in the room. She then picks up all exams written in that room and adds them to the bottom of her pile.
3. After having visited each room exactly once, Linda returns to the first room, and distributes the remaining exams from her pile there.

Naturally, it is imperative that no student receives their own exam to review, and that Linda does not run out of exams in her pile while doing the redistribution (i.e., that when entering a room after the first one, Linda's pile contains at least as many exams as there are students in the room). Whether or not this is the case depends on the order in which the rooms are visited. We say that an ordering of the rooms is *safe* if Linda will not run out of exams in her pile when visiting rooms in that order, and that there is no chance that any student receives their own exam to review.

Can you find a safe order in which to visit the rooms (or determine that no safe order exists)?

Input

The input consists of:

- one line containing an integer n ($2 \leq n \leq 30$), the number of rooms.
- one line containing n integers $s_1, \ldots, s_n$ ($1 \leq s_i \leq 100$ for each i), where s_i is the number of students in room i.

Output

If it is impossible to redistribute the exams safely, output "`impossible`". Otherwise, output a safe order in which to visit the rooms. If there are multiple safe orders, you may give any of them.

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

Sample Input 2	Sample Output 2
2 10 20	impossible

Problem F
Free Weights

A well-organised rack of weights.

The city of Bath is a noted olympic training ground—bringing local, national, and even international teams to practice. However, even the finest gymnasium falls victim to the cardinal sin... Weights put back in the wrong spots.

All of the pairs of dumbbells sit in no particular order on the two racks, possibly even with some of them split between rows. Initially each row has an equal number of dumbbells, however, this being a well-funded professional gym, there is infinite space at either end of each to hold any additional weights.

To move a dumbbell, you may either roll it to a free neighbouring space on the same row with almost no effort, or you may pick up and lift it to another free spot; this takes strength proportional to its weight. For each pair of dumbbells, both have the same unique weight.

What is the heaviest of the weights that you need to be able to lift in order to put identical weights next to each other? Note that you may end up with different numbers of weights on each row after rearranging; this is fine.

Input

The input consists of:

- one line containing the integer n ($1 \leq n \leq 10^6$), the number of pairs;
- two lines, each containing n integers $w_1 \ldots w_n$ ($1 \leq w_i \leq 10^9$ for each i), where w_i is the mass of the weight i-th from the left along this row.

Every weight in the input appears exactly twice.

Output

Output the weight of the heaviest dumbbell that must be moved, in order that all items can be paired up while lifting the smallest possible maximum weight.

Sample Input 1	Sample Output 1
5 2 1 8 2 8 9 9 4 1 4	2

Sample Input 2	Sample Output 2
8 7 7 15 15 2 2 4 4 5 5 3 3 9 9 1 1	0

Problem G
Gotta Nudge 'Em All

Nudgémon GO is a game in which players should earn as much experience points (*XP*) as possible, by catching and evolving Nudgémon. You gain 100 XP for catching a Nudgémon and 500 XP for evolving a Nudgémon. Your friend has been playing this game a lot recently, but you believe that his strategy is not optimal.

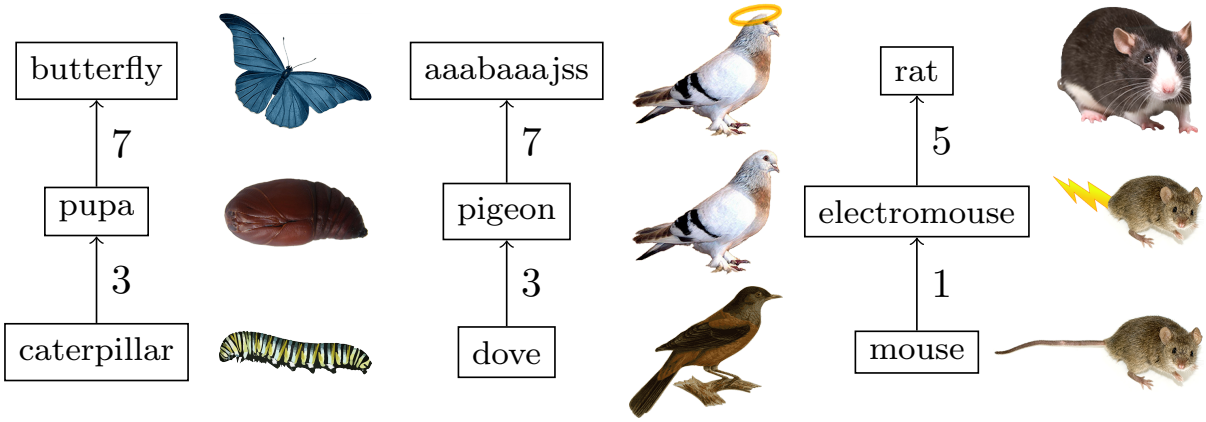
All Nudgémon are split into families, each of which has its own unique type of *candy*. The Nudgémon in a family are ranked from weakest to strongest and hence form a chain. Any Nudgémon that is not the strongest from its family can be evolved to the next ranked Nudgémon from the same family.

Candies are a fundamental currency in the Nudgémon universe:

- When you catch a Nudgémon you earn 3 candies, all associated with the Nudgémon's family.
- When you irreversibly transfer a Nudgémon away from your possession, you earn 1 candy associated with the Nudgémon's family.

Every evolution of a Nudgémon consumes a specific amount of its family's kind of candy. Furthermore, the costs of evolutions along the family chain are non-decreasing, meaning that higher-ranked evolutions in the family will cost the same or more as lower ones.

Here is an example of possible Nudgémon evolutions:



Apart from making the developers money and nudging 'em all, the goal of this game is to earn as much XP as possible to level up the player's character and be able to encounter stronger Nudgémon in the wild. As such, coinciding with the first goal, you can buy a *Blessed Egg* with real money in the game. This item allows you to double your earned XP for the next 30 minutes since activation, i.e. when the Egg is activated at time e (in seconds since the start of the game), for any action taken on time t, you will earn double XP if and only if $e \leq t < e + 1800$.

At the start of the game your friend received a single Blessed Egg. Unfortunately, he completely wasted it. You believe that it is better to only evolve Nudgémon while the Blessed Egg is active, otherwise it is a huge waste of resources! To prove your point to your friend, you took a log of all Nudgémon he caught with timestamps and decided to calculate the maximum amount of XP he could have had right now if he was strategic about when to activate his Blessed Egg and only evolved Nudgémon during the time it was active.

Input

The input consists of:

- one line containing an integer f ($0 \leq f \leq 10^5$), the number of Nudgémon families;
- f lines describing a family of Nudgémon, where each line consists of the following elements:
  - an integer s_i ($1 \leq s_i \leq 10^5$), the number of Nudgémon in this family;
  - $s_i - 1$ times the name of a Nudgémon, followed by an integer c_j ($1 \leq c_j \leq 10^5$), the amount of candies (of appropriate type) consumed by evolving this Nudgémon;
  - the name of the strongest Nudgémon in this family;
- one line containing an integer n ($0 \leq n \leq 4 \cdot 10^5$), the number of Nudgémon your friend caught;
- n lines containing an integer t_i ($0 \leq t_i \leq 10^9$) and a string p_i, the time at which the Nudgémon was caught and the name of the caught Nudgémon.

It is guaranteed that there are at most 10^5 Nudgémon kinds ($\sum_i s_i \leq 10^5$). The Nudgémon in each family are given in order of increasing rank, and thus the values of c in one family are non-decreasing. Every Nudgémon name is a string of between 1 and 20 lowercase letters. The times t_i are non-decreasing (your friend is so quick he can catch multiple Nudgémon in a single second). No Nudgémon name appears more than once within a family or within more than one family, and all n Nudgémon that are caught belong to one of the families.

Output

Output the maximum amount of XP your friend could have had at the current time had he activated his Blessed Egg at the optimal time and only evolved Nudgémon during the time it was active.

Sample Input 1

```
3
3 caterpillar 3 pupa 7 butterfly
3 dove 3 pigeon 7 aaabaaajss
3 mouse 1 electromouse 5 rat
7
0 electromouse
500 electromouse
1000 electromouse
1500 rat
2000 aaabaaajss
2500 pigeon
3000 butterfly
```

Sample Output 1

```
5100
```

Sample Input 2

```
1
1 slownudge
2
0 slownudge
1800 slownudge
```

Sample Output 2

```
300
```

Problem H
Hamiltonian Hypercube

Hypercube graphs are fascinatingly regular, hence you have devoted a lot of time studying the mathematics related to them. The vertices of a hypercube graph of dimension n are all binary strings of length n, and two vertices are connected if they differ in a single position. There are many interesting relationships between hypercube graphs and error-correcting code.

One such relationship concerns the n-bit Gray Code, which is an ordering of the binary strings of length n, defined recursively as follows. The sequence of words in the n-bit code first consists of the words of the $(n-1)$-bit code, each prepended by a 0, followed by the same words in reverse order, each prepended by a 1. The 1-bit Gray Code just consists of a 0 and a 1. For example the 3-bit Gray Code is the following sequence:

$$000, 001, 011, 010, 110, 111, 101, 100$$

Now, the n-bit Gray Code forms a Hamiltonian path in the n-dimensional hypercube, i.e., a path that visits every vertex exactly once (see Figure H.1).

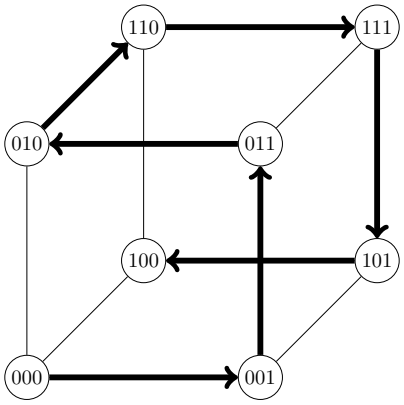


Figure H.1: The 3-dimensional hypercube and the Hamiltonian path corresponding to the 3-bit Gray Code.

You wonder how many vertices there are between the vertices 0^n (n zeros) and 1^n (n ones) on that path. Obviously it will be somewhere between $2^{n-1}-1$ and 2^n-2, since in general 0^n is the first vertex, and 1^n is somewhere in the second half of the path. After finding an elegant answer to this question you ask yourself whether you can generalise the answer by writing a program that can determine the number of vertices between two arbitrary vertices of the hypercube, in the path corresponding to the Gray Code.

Input

The input consists of a single line, containing:

- one integer n ($1 \leq n \leq 60$), the dimension of the hypercube
- two binary strings a and b, both of length n, where a appears before b in the n-bit Gray Code.

Output

Output the number of code words between a and b in the n-bit Gray Code.

Sample Input 1	Sample Output 1
3 001 111	3

Sample Input 2	Sample Output 2
3 110 100	2

Problem I
Iron and Coal

There are many excellent strategy board games, and your favourite among them is called "Steel Age". It offers many different paths to victory but you prefer the blood-and-fire-strategy: build as many soldiers as possible and club your opposition into submission. To be able to build soldiers you need two resources: iron ore and coal.

Coal. Photo by US Federal Government

The board consists of different cells numbered from 1 to n which can contain resources. The rules for moving from one cell to another are rather complicated: if you can move from cell A to cell B, it does not always mean that you can also move from B to A. For example, if two cells are connected by a river, then you may be able to move downstream, but not upstream, so long as you didn't invent a steam engine; however, it still could be possible to reach the upstream cell by using roads and taking a detour over other cells.

At the beginning of the game you own only one such cell, where all your settlers are located. At every move you are allowed to move an arbitrary number of settlers from a cell to one of its accessible neighbours. By moving your settlers into a cell for the first time, you "claim" it. Every claimed cell will bind one settler, which has to stay in this cell until the end of the game. However, there is no need to leave a settler in your initial cell because it is where your palace is located and thus the cell stays claimed for all time.

Your goal is to claim at least one cell containing the resource "iron ore" and at least one cell with resource "coal" in order to be able to build soldiers. What is the minimal number of settlers you need to reach this goal?

Input

The input consists of:

- One line with three integers n ($2 \leq n \leq 10^5$), the number of cells on the playing field, m ($1 \leq m < n$), the number of cells containing iron ore, and k ($1 \leq k < n$), the number of cells containing coal.
- One line with m distinct integers $o_1, \ldots, o_m$ ($1 \leq o_i \leq n$ for all $1 \leq i \leq m$), where $o_1, \ldots, o_m$ are the IDs of cells with iron ore.
- One line with k distinct integers $c_1, \ldots, c_k$ ($1 \leq c_i \leq n$ for all $1 \leq i \leq k$), where $c_1, \ldots, c_k$ are the IDs of cells with coal.
- n lines describing the topology of the board. The j-th line of this block specifies the accessible neighbours of the j-th cell and consists of the following integers:
  - One integer $0 \leq a \leq 10$, the number of cells accessible from cell j.
  - a distinct integers $b_1, \ldots, b_a$ ($1 \leq b_i \leq n$, $b_i \neq j$ for all $1 \leq i \leq a$), the IDs of the cells accessible from cell j.

It is guaranteed, that no cell contains both resources, iron ore and coal. At the beginning of the game you own only the cell with ID 1.

Output

Output the minimum number of settlers needed to claim at least one cell with coal and at least one cell with iron ore. Output "`impossible`" if it is impossible to own both, coal and iron ore.

Sample Input 1

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

Sample Output 1

```
2
```

Sample Input 2

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

Sample Output 2

```
impossible
```

Problem J
Jupiter Orbiter

Although we imagine interplanetary probes to be very sophisticated pieces of technology, their information systems are quite archaic. You might assume that they have a certain amount of contiguous main memory and can store their data wherever is convenient, but unfortunately that is not the case. The probe's main memory is organised in a number of FIFO (first-in-first-out) queues. In such a queue, data has to be taken out of the queue in the same order as it has been added to it.

The Juno spacecraft in front of Jupiter. Graphic by NASA

A probe has multiple sensors and each sensor is linked to one of the queues. Whenever a sensor finishes recording, it appends the generated data to its queue. A sensor can write data to the queue only if the queue has enough space left to take all the data; if not, the data is lost.

In order to transfer data from the probe to Earth (in a process called *downlinking*), the path between the satellite and Earth must not be blocked by anything (e.g. a planet like Jupiter) and the antenna must be correctly positioned. During each downlink opportunity, data can be taken from multiple queues and transmitted back to Earth. The total amount of data that can be transmitted during a downlink opportunity depends on the length of the downlink opportunity and distance to Earth. Sensors do not collect data during downlink opportunities, since all electricity available is devoted to the transmitter.

The most important thing for scientists is not to lose any data recorded by sensors. In particular, all queues have to be empty after the last downlink opportunity. The scientists have asked you to write a program to determine whether all data can be transferred to Earth in a given time frame.

Input

- one line containing three positive integers n, q, s ($1 \leq n, q \leq 30$, $1 \leq s \leq 100$), the number of downlink windows, FIFO queues, and sensors, respectively.
- one line with s integers $q_1 \ldots q_s$ ($1 \leq q_i \leq q$ for each i), determining for each sensor the queue it feeds its data into.
- one line with q integers $c_1 \ldots c_q$ ($1 \leq c_i \leq 10^6$ for each i), determining for each queue the size of the queue in megabytes.
- n lines, each describing one downlink window. Each contains $s + 1$ non-negative integers.
  - The first integer d ($1 \leq d \leq 10^6$) states the number of megabytes that can be transferred to earth during the window.
  - The following s numbers $a_1 \ldots a_s$ ($0 \leq a_i \leq 10^6$ for each i) describing the amount of data (in megabytes) generated by each of the sensors after the last but before this downlink window.

There will never be new data during a downlink window.

Output

Output "`possible`" if it is possible to transfer all data to Earth, and "`impossible`" otherwise.

Sample Input 1

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

Sample Output 1

```
possible
```

Sample Input 2

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

Sample Output 2

```
impossible
```

Problem K
Kiwi Trees

Photo by Nathaniel Hayag, cc by-nd

You are about to plant a pair of fine kiwi trees inside your big property. You are worried that tree branches would grow beyond the boundaries of your property, making your neighbours complain. You must also avoid planting the two trees too close to each other so that the branches of the trees grow into each other, because that could lead to a loss of fruit.

The seller of the trees guaranteed that no branch or leaf will be farther than 4 meters from the trunk of its tree, thus we will model the trees as circles of radius 4 meters. The trunks are perfectly vertical.

Local government regulations forbid certain shapes of properties. In particular, in order for government employees to be able to draw and handle maps of the area, each property must satisfy the following properties:

1. The boundary is a simple polygon, i.e. the sides are non-intersecting and form a closed path.
2. Each side length of the property is at least 30 meters long.
3. The angle between any two consecutive sides of the property must be at least 18 degrees (10% of a straight angle), and at most 144 degrees (80% of a straight angle, or if you will, the angle of a regular decagon).

Non-convex properties are allowed as long as the angles of consecutive sides follow rule 3 above, so the inner angle at a vertex can also be between $360 - 144 = 216$ and $360 - 18 = 342$ degrees. See Figure K.1 for an example.

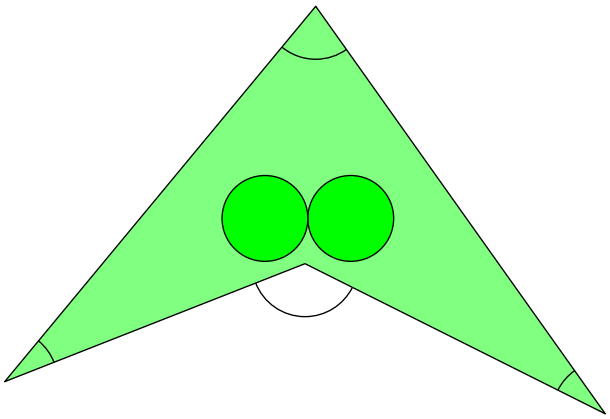


Figure K.1: Illustration of Sample Input 1 and the solution given in Sample Output 1. All the marked angles are at least 18 and at most 144 degrees.

Your property follows these rules. Can you plant two trees inside the property such that their branches and leaves do not grow beyond its boundary, and that the branches and leaves of each tree do not grow into the other tree?

Input

The input consists of:

- One line containing an integer n, where $3 \leq n \leq 2\,000$ is the number of vertices of the polygon describing your property.
- n lines describing one vertex each. Each such line contains two integers x and y, where $0 \leq x, y \leq 10^7$. These two numbers denote the x- and y-coordinates of a vertex in *millimeters*, in a clockwise order as they appear in the polygon.

Also, each polygon side is at least 30 meters ($30\,000$ millimeters) long and the angle of two segments at a vertex is at least 18 degrees and at most 144 degrees. The polygon is non-intersecting and closed, i.e. the last vertex is connected to the first vertex.

Output

If it is possible to plant two trees without their branches growing beyond the boundary of your property or into each other, output two lines, each containing two real numbers x and y giving the coordinates in millimeters of two points where the trees can be planted.

Otherwise, print "`impossible`".

You may assume that increasing or decreasing the radius by 1 mm will not change whether or not it is possible to plant the trees. Your answer will be accepted if the tree locations are at least 3999 mm away from the boundary and at least $2 \cdot 3999$ mm away from each other.

Sample Input 1	Sample Output 1
4 0 3000 29000 38000 56000 0 28000 14000	32266.13633130 18219.22050526 24266.13633130 18219.22050526

Sample Input 2	Sample Output 2
3 50000 50000 69500 73000 99000 80000	impossible