Problem A. Airport Logistics

Source file name:	airport.c, airport.cpp, airport.java, airport.py
Input:	`Standard`
Output:	`Standard`

Many airports have moving conveyor belts in the corridors between halls and terminals. Instead of walking on the floor, passengers can choose to stand on a conveyor or, even better, walk on a conveyor to get to the end of the corridor much faster.

The brand new Delft City Airport uses a similar system. However, in line with the latest fashion in airport architecture, there are no corridors: the entire airport is one big hall with a bunch of conveyor lines laid out on the floor arbitrarily.

To get from a certain point A to a certain point B, a passenger can use any combination of walking on the floor and walking on conveyors. Passengers can hop on or off a conveyor at any point along the conveyor. It is also possible to cross a conveyor without actually standing on it.

Walking on the floor goes at a speed of 1 meter/second. Walking forward on a conveyor goes at a total speed of 2 meter/second.

Walking in reverse direction on a conveyor is useless and illegal, but you may walk on the floor immediately next to the conveyor. (Conveyors are infinitely thin.)

How fast can you get from A to B?

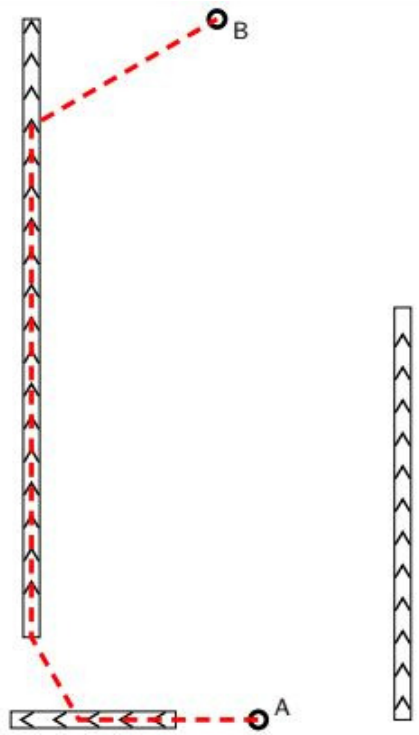


Figure 1: Fastest route for first example input.

Input

The first line contains four floating point numbers, X_A, Y_A, X_B, and Y_B. They describe the coordinates of your initial location $A = (X_A, Y_A)$ and your final location $B = (X_B, Y_B)$.

The second line contains an integer N, the number of conveyors in the hall ($0 \leq N \leq 100$). The following N lines each contain four floating point numbers, X_1, Y_1, X_2, and Y_2, describing a conveyor which starts at the point (X_1, Y_1) and ends at the point (X_2, Y_2), running in a straight line from start to end.

All coordinates are floating point numbers in the range $(0 \leq X, Y \leq 10^3)$, expressed in units of meters.

Conveyors are at least 1 meter long. Conveyors do not intersect or touch. Your start and destination are not on any conveyor.

Output

Write one line with a floating point number, the minimum time (in seconds) needed to get from A to B in seconds.

Your answer may have an absolute error of at most 10^{-4}.

Example

Input	Output
60.0 0.0 50.0 170.0 3 40.0 0.0 0.0 0.0 5.0 20.0 5.0 170.0 95.0 0.0 95.0 80.0	168.791651
60.0 0.0 50.0 170.0 3 40.0 0.0 0.0 0.0 5.0 20.0 5.0 170.0 95.0 0.0 95.0 100.0	163.527474
0.0 1.0 4.0 1.0 1 0.0 0.0 4.0 0.0	3.732051
0.0 1.0 10.0 1.0 2 1.0 0.0 2.0 3.0 6.0 1.0 4.0 1.0	10.000000

Problem B. Battle Simulation

Source file name:	battle.c, battle.cpp, battle.java, battle.py
Input:	`Standard`
Output:	`Standard`

A terrible monster is rampaging through Neo Tokyo 5! The Earth Defense Force (EDF) has sent a mech unit[1] to defeat the monster. Because there is only a single mech unit available after previous monster rampages, the EDF has decided to simulate the upcoming battle between the mech and the monster before launching an assault. The EDF noted that the monster's attack pattern can be simulated by a series of moves that it performs in succession. When denoting each of its moves with a single letter, the attack pattern can be simulated as a single string, which should be read from left to right. The monster has the following moves:

Picture by Bandai Namco via Wikimedia Commons

- Rake, denoted by the letter '`R`';
- Bite, denoted by the letter '`B`';
- Laser breath, denoted by the letter '`L`'

In order to defeat the monster, the mech must perform a counter move per move that the monster makes:

- Slice, denoted by the letter '`S`', counters the monster's rake;
- Kick, denoted by the letter '`K`', counters the monster's bite;
- Shield, denoted by the letter '`H`', counters the monster's laser breath;

However, there is one catch. When the monster performs a subsequent combination of the three moves *Rake*, *Bite* and *Laser breath*, in any order, it becomes a very powerful attack for which the mech must perform a single counter move called *Combo breaker*, denoted by the letter '`C`'. A single *Combo breaker* absorbs the entire combination of three moves. Any following moves from the monster will have to be countered separately or as part of a new combination. A move of the monster can never be part of more than one combination.

Through extensive analysis of the monster's past behaviour, the EDF is now able to reliably predict the actions of the monster ahead of time. You are given a string representing the moves that the monster will use when battling the mech. The EDF needs you to write a program that outputs the sequence of moves that the mech must perform in order to defeat the monster.

Input

A single line containing a string of at least 1 and at most 10^6 characters, consisting of the letters '`R`', '`B`' and '`L`'.

Output

Output a single string consisting of the letters denoting the moves that are to be made in succession by the mech in order to defeat the monster.

[1] huge bipedal robot, piloted by Japanese teenagers.

Example

Input	Output
RRBBBLLR	SSKKKHHS
RBLLLBRR	CHCS
RBLBR	CKS

Problem C. Brexit

Source file name:	brexit.c, brexit.cpp, brexit.java, brexit.py
Input:	`Standard`
Output:	`Standard`

A long time ago in a galaxy far, far away, there was a large interstellar trading union, consisting of many countries from all across the galaxy. Recently, one of the countries decided to leave the union. As a result, other countries are thinking about leaving too, as their participation in the union is no longer beneficial when their main trading partners are gone.

Picture by NASA

You are a concerned citizen of country X, and you want to find out whether your country will remain in the union or not. You have crafted a list of all pairs of countries that are trading partners of one another. If at least half of the trading partners of any given country Y leave the union, country Y will soon follow. Given this information, you now intend to determine whether your home country will leave the union.

Input

The input starts with one line containing four space separated integers C, P, X, and L. These denote the total number of countries ($2 \leq C \leq 2 * 10^5$), the number of trading partnerships ($1 \leq P \leq 3 * 10^5$), the number of your home country ($1 \leq X \leq C$) and finally the number of the first country to leave, setting in motion a chain reaction with potentially disastrous consequences ($1 \leq L \leq C$).

This is followed by P lines, each containing two space separated integers A_i and B_i satisfying $1 \leq A_i < B_i \leq C$. Such a line denotes a trade partnership between countries A_i and B_i. No pair of countries is listed more than once.

Initially, every country has at least one trading partner in the union.

Output

For each test case, output one line containing either `leave` or `stay`, denoting whether you home country leaves or stays in the union.

Example

Input	Output
4 3 4 1 2 3 2 4 1 2	stay
5 5 1 1 3 4 1 2 2 3 1 3 2 5	leave
4 5 3 1 1 2 1 3 2 3 2 4 3 4	stay
10 14 1 10 1 2 1 3 1 4 2 5 3 5 4 5 5 6 5 7 5 8 5 9 6 10 7 10 8 10 9 10	leave

Problem D. Bridge Automation

Source file name:	bridge.c, bridge.cpp, bridge.java, bridge.py
Input:	`Standard`
Output:	`Standard`

In Delft there are a number of bridges that are still being operated by a human, known as the bridge operator. One such bridge operator will soon retire, hence there is the need for a replacement. The Bridge And Poker Committee has decided to use a computer program to automatically open and close the bridge, eliminating the need for human interaction.

However, the computer program still needs to be written. The requirements for this project are as follows:

1. No boat may be forced to wait for more than 30 minutes.
2. The amount of time during which the bridge is unavailable to road traffic must be as small as possible while still satisfying requirement 1.

It takes 60 seconds to raise or lower the bridge. During this time the bridge is not available to either road traffic or water traffic.

Boats arrive at the bridge at predictable times. It takes 20 seconds for a boat to sail through the bridge, assuming the bridge is already fully raised.

If the bridge is not fully raised when a boat arrives, the boat must wait. If there are boats waiting when the bridge becomes fully raised, these boats pass through the bridge one-by-one, which takes 20 seconds per boat. The bridge must remain fully raised as long as there are still boats sailing through! As soon as all boats have passed, the bridge may be lowered. But it might be more efficient to keep the bridge raised for a little while longer if the next boat is soon to arrive.

Given the arrival times of all boats, operate the bridge such that all boats can pass through without any boat waiting longer than 30 minutes. What is the total amount of time during which the bridge is unavailable to road traffic?

Input

The first line contains an integer N, the number of boats that must pass the bridge ($1 \leq N \leq 4 * 10^3$).

Then follow N lines, each containing an integer T_i, the time at which boat i will arrive at the bridge in seconds ($60 \leq T_i \leq 10^5$).

Boats are sorted by increasing time of arrival, and never arrive within 20 seconds of each other ($i < j$ implies $T_i + 20 \leq T_j$).

Output

Write one line with an integer, the total number of seconds during which the bridge must be unavailable for road traffic in order for all boats to pass the bridge.

Example

Input	Output
2 100 200	160
3 100 200 2010	250
3 100 200 2100	300

Problem E. Charles in Charge

Source file name:	charles.c, charles.cpp, charles.java, charles.py
Input:	`Standard`
Output:	`Standard`

Every day, Charles drives from his home to work and back. He uses the highways of the country that run from one city to another. Charles has decided that he wants to help the environment by buying an electrical car. Electrical cars, however, are not very common in his country yet. They can only be charged inside a city; there are no charging stations along the highways in between the cities. Moreover, all electrical cars are identical except for one thing: the size of the battery. As batteries are very expensive, Charles would like to buy a car with battery that is as small as possible.

Picture by Frank Hebbert via Flickr

However, this greatly increases the time it takes for him to get home, much to the distaste of his wife, Charlotte. This has spawned an argument, and after much discussion they have decided to compromise: Charlotte is fine with Charles taking a longer route, as long as it its length is at most $X\%$ longer than the length of shortest route that Charles could have taken to get home from work by using a regular car. Charles has agreed with this, and he now wants to find a route that minimizes the size of the car battery that he needs, i.e. the route that minimizes the maximum distance that Charles has to drive on a highway without passing through a city.

The amount of time Charles spends to charge his car can be neglected.

Input

The input starts with integers $2 \leq N \leq 10^4$, $1 \leq M \leq 10^5$ and $0 \leq X \leq 10^4$, the number of cities, the number of highways connecting the cities and the aforementioned percentage X. City 1 is the place where Charles lives and city N is where he works.

Then follow M lines with on each line three integers: $1 \leq C_1 \leq N$, $1 \leq C_2 \leq N$, $1 \leq T \leq 10^9$. This means that there is a highway of length T connecting cities C_1 and C_2 (Charles can traverse the highway in both directions) *without* passing through any other cities. You may assume that there exists a path from city 1 to city N.

Output

The output is a single integer: the smallest maximum distance that Charles has to travel on a highway without passing through a city, such that the route he takes is at most $X\%$ longer than the shortest route.

Example

Input	Output
2 1 100 1 2 5	5
9 8 15 1 9 16 1 4 4 4 5 4 5 6 4 6 8 4 4 7 5 7 8 5 8 9 4	5
9 8 30 1 9 16 1 4 4 4 5 4 5 6 4 6 8 4 4 7 5 7 8 5 8 9 4	4

Explanation

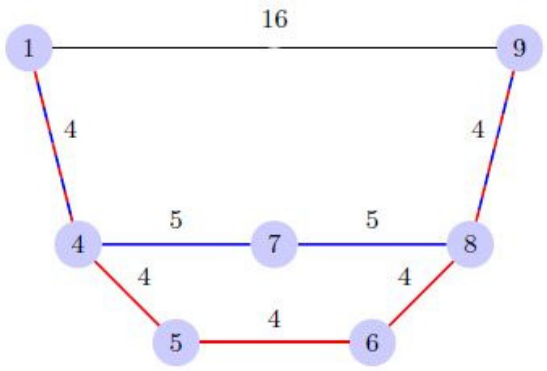


Figure 2: The graph of the second and third test case. The shortest path has length 16. In the second testcase, Charles's travel distance may not exceed $16 \cdot 1.15 = 18.4$ distance units. As a result, he cannot use a battery with which he can travel 4 distance units, as the red path 1–4–5–6–8–9 has length 20. Therefore, he uses the blue path 1–4–7–8–9, which has length 18, and the longest edge has length 5. In the third test case, he is allowed to travel $16 \cdot 1.30 = 20.8$ distance units, so he can follow the red path, where the longest edge has length 4.

Problem F. Endless Turning

Source file name:	endless.c, endless.cpp, endless.java, endless.py
Input:	`Standard`
Output:	`Standard`

Last week your little sister celebrated her birthday, and she was very pleased with her birthday present: a brand new scooter! Since then several times a day she goes for a drive, without telling anyone. She will leave the house, and go right on the pavement along the road. Fortunately she knows that she is not allowed to cross the road, and she is not sufficiently skilful with her new toy to turn around on the pavement. This means that, in order to continue her way, at each intersection she must turn right.

Picture by Cha già José via Flickr

Every time your sister sets out, you are sent after her, of which you grow tired. Thus, knowing your little sister's ways with her scooter, you decide to write a program to find her.

The city consists of R roads. Each road has a name which does not contain any spaces, and is an infinite line in the Euclidean plane. No three roads go through one point, i.e. at every intersection exactly two roads intersect. You figured out that your sister by this time should have completed N turns on the intersections of the city, unless of course she managed to leave the city by travelling on a road in a direction in which there is no other intersection, in which case she might not be able to even complete N turns. Your task is to write a program that tells you on which road your sister is right now.

Input

The first line contains four integers: the number of roads R, the number of turns N and the $X-$ and $Y-$coordinate of your parents' home, satisfying $R \leq 100$, $N \leq 10^{10}$ and $|X|, |Y| \leq 10^7$.

The next R lines each describe a street. Each line contains one string S (without spaces, containing only alphanumeric characters, of length at most 20), the name of the street, and four integers X_1, Y_1, X_2, and Y_2, satisfying $|X_1|, |X_2|, |Y_1|, |Y_2| \leq 10^7$ and $(X_1, Y_1) \neq (X_2, Y_2)$, indicating that road S goes through points (X_1, Y_1) and (X_2, Y_2).

The location of your parents' home (X, Y) is guaranteed to lie on a unique non-vertical street (i.e. not on an intersection), on the south (negative $Y-$direction) side of the street, meaning that your little sister will depart in the east (positive X) direction. Moreover, each intersection is guaranteed to have coordinates of absolute value at most 10^7, and is guaranteed to lie at least 10^{-4} from each other intersection in the same street.

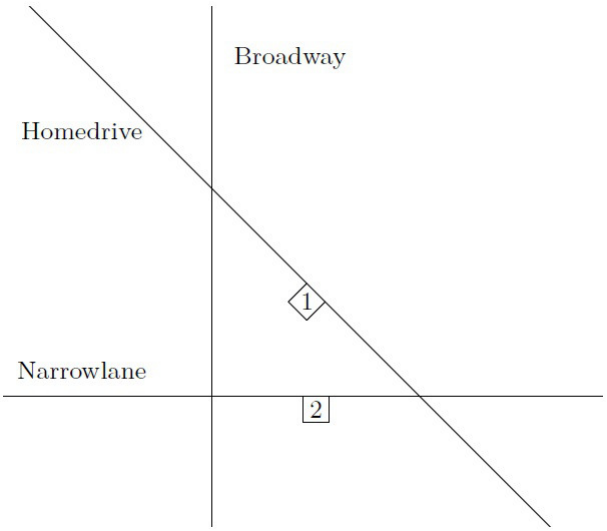
Output

Output a single line containing the name of the road where your little sister can be found.

Example

Input	Output
3 4 1 1 Broadway 0 0 0 1 Narrowlane 0 0 1 0 Homedrive 1 1 2 0	Narrowlane
3 4 1 0 Broadway 0 0 0 1 Narrowlane 0 0 1 0 Homedrive 1 1 2 0	Homedrive

Explanation



In the first case your sister starts on Homedrive heading east, turns right on Narrowlane, Broadway, Homedrive and Narrowlane, which means that after four turns she is in Narrowlane.

In the second case your sister starts on Narrowlane heading east and turns right on Homedrive, leaving the city behind her. She will never finish four turns and ends on Homedrive.

Problem G. Manhattan Positioning System

Source file name:	manhattan.c, manhattan.cpp, manhattan.java, manhattan.py
Input:	`Standard`
Output:	`Standard`

The Manhattan Positioning System (MPS) is a modern variant of GPS, optimized for use in large cities. MPS assumes all positions are discrete points on a regular two-dimensional grid. Within MPS, a position is represented by a pair of integers (X, Y).

To determine its position, an MPS receiver first measures its distance to a number of beacons. Beacons have known, fixed locations. MPS signals propagate only along the X and Y axes through the streets of the city, not diagonally through building blocks. When an MPS receiver at (X_R, Y_R) measures its distance to a beacon at (X_B, Y_B), it thus obtains the Manhattan distance: $|X_R - X_B| + |Y_R - Y_B|$.

Given the positions of a number of beacons and the Manhattan-distances between the receiver and each beacon, determine the position of the receiver. Note that the receiver must be at an integer grid position (MPS does not yet support fractional coordinates).

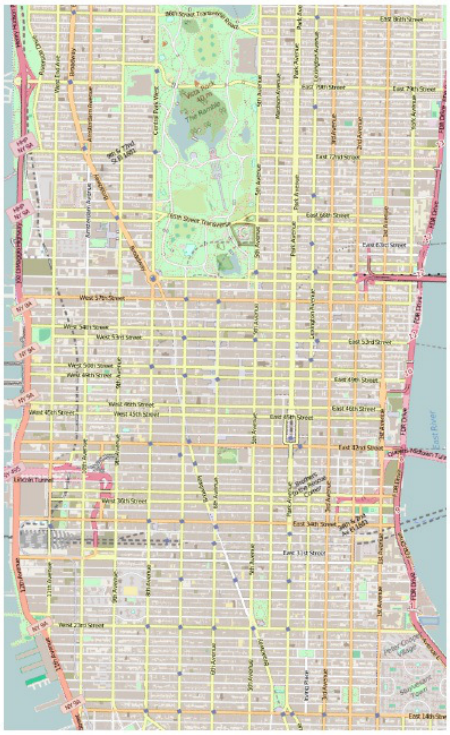


Figure 3: MPS is ideal for this city
© OpenStreetMap contributors

Input

The first line contains an integer N, the number of beacons ($1 \leq N \leq 10^3$). Then follow N lines, each containing three integers, X_i, Y_i, and D_i, such that $-10^6 \leq X_i, Y_i \leq 10^6$ and $0 \leq D_i \leq 4*10^6$. The pair (X_i, Y_i) denotes the position of beacon i, while D_i is the Manhattan distance between receiver and beacon i.

No two beacons have the same position.

Output

If there is exactly one receiver position consistent with the input, write one line with two integers, X_R and Y_R, the position of the receiver.

If multiple receiver positions are consistent with the input, write one line with the word `uncertain`.

If no receiver position is consistent with the input, write one line with the word `impossible`.

Example

Input	Output
3 999999 0 1000 999900 950 451 987654 123 13222	1000200 799
2 100 0 101 0 200 199	uncertain
2 100 0 100 0 200 199	impossible
2 0 0 5 10 0 6	impossible

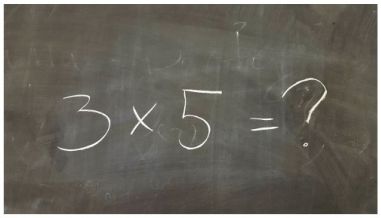
Problem H. Multiplying Digits

Source file name:	multiplying.c, multiplying.cpp, multiplying.java, multiplying.py
Input:	`Standard`
Output:	`Standard`

For every positive integer we may obtain a non-negative integer by multiplying its digits. This defines a function f, e.g. $f(38) = 24$.

This function gets more interesting if we allow for other bases. In base 3, the number 80 is written as 2222, so: $f_3(80) = 16$.

We want you to solve the reverse problem: given a base B and a number N, what is the smallest positive integer X such that $f_B(X) = N$?



Picture by Mees de Vries

Input

The input consists of a single line containing two integers B and N, satisfying $2 < B \leq 10^4$ and $0 < N < 2^{63}$.

Output

Output the smallest positive integer solution X of the equation $f_B(X) = N$. If no such X exists, output the word "`impossible`". The input is carefully chosen such that $X < 2^{63}$ holds (if X exists).

Example

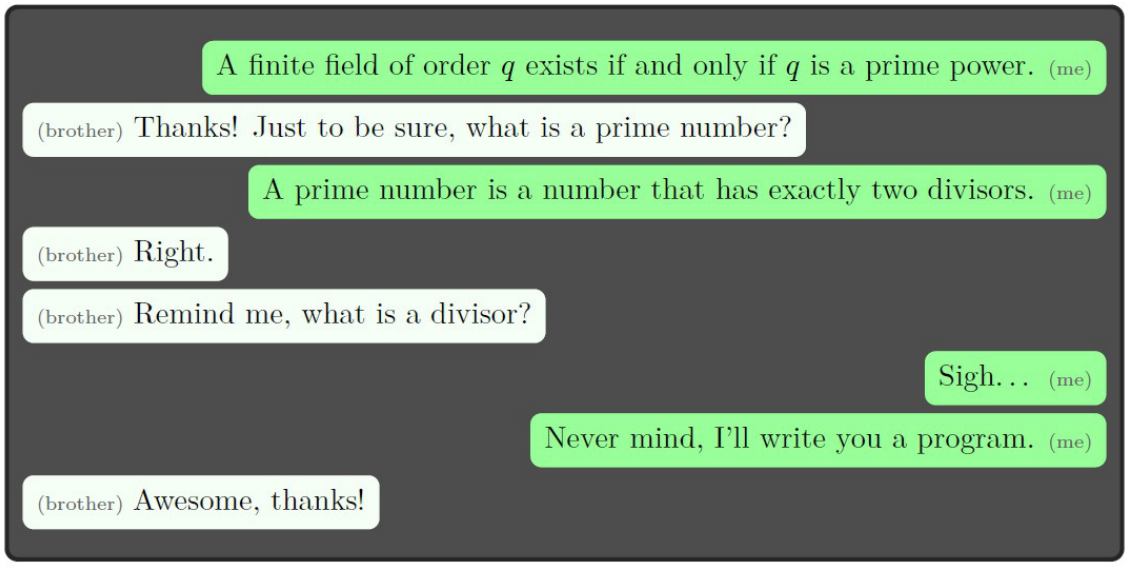
Input	Output
10 24	38
10 11	impossible
9 216	546
10000 5810859769934419200	5989840988999909996

Problem I. Older Brother

Source file name:	brother.c, brother.cpp, brother.java, brother.py
Input:	`Standard`
Output:	`Standard`

Your older brother is an amateur mathematician with lots of experience. However, his memory is very bad. He recently got interested in linear algebra over finite fields, but he does not remember exactly which finite fields exist. For you, this is an easy question: a finite field of order q exists if and only if q is a prime power, that is, $q = p^k$ holds for some prime number p and some integer $k \geq 1$. Furthermore, in that case the field is unique (up to isomorphism).

The conversation with your brother went something like this:



Input

The input consists of one integer q, satisfying $1 \leq q \leq 10^9$.

Output

Output `yes` if there exists a finite field of order q. Otherwise, output `no`.

Example

Input	Output
1	no
37	yes
65536	yes

Problem J. Programming Tutors

Source file name:	tutors.c, tutors.cpp, tutors.java, tutors.py
Input:	`Standard`
Output:	`Standard`

You are the founder of the Bruce Arden Programming Collective, which is a tutoring programme that matches experienced programmers with newbies to teach them. You have N students and N tutors, but now you have to match them up. Since the students will have to travel to their tutors' houses from their own (or vice versa) you decide to do your matching based on travel distance.

Picture by Damien Pollet via Flickr

Minimising overall distance doesn't seem fair; it might happen that one student has to travel a huge distance while all the other students get a tutor very close by, even though the tutors could have been split up so that each gets a tutor that is at least somewhat close.

Thus, you opt to minimise the distance travelled by the student who is worst off; one pairing of students to tutors is better than another if the student who has to travel farthest in the first pairing has to travel less far than the student who has to travel farthest in the second pairing.

Because the students live in a city, the distance that a student needs to travel is not the literal distance between them and their tutor. Instead, the distance between points (X, Y) and (X', Y') in the city is

$$|X - X'| + |Y - Y'|$$

Input

The first line of the input contains an integer N, with $1 \leq N \leq 100$, the number of students and the number of tutors to pair up.

Then, there are N lines, each with two space separated integers with absolute value at most 10^8, which give the locations of the N students.

These are followed by N lines, each with two space separated integers with absolute value at most 10^8, which give the locations of the N tutors.

Note that it is possible for students and/or tutors to have identical locations (they may share a house).

Output

Output a single line containing a single integer K, where K is the least integer such that there exists a pairing of students to tutors so that no pair has distance greater than K between them.

Example

Input	Output
2 0 0 0 3 0 2 0 5	2
4 0 1 0 2 0 3 0 4 1 0 1 1 1 2 1 3	2
3 0 5 5 2 4 5 3 3 5 2 5 2	5
2 0 0 0 5 -1 4 8 3	10

Problem K. Safe Racing

Source file name:	racing.c, racing.cpp, racing.java, racing.py
Input:	`Standard`
Output:	`Standard`

Tomorrow is racing day. There will be yet another grand prix in yet another country. Beside the safety car, there are various other security measures in order to make sure that everybody is as safe as possible. Among these safety measures are the track marshals: special race officials standing along the track with an assortment of flags that they can use to signal various messages to the drivers. For instance, the yellow flag warns the drivers of a dangerous situation, and the blue flag is used to order a lapped car to make way for one of the faster cars.

Picture by Martin Pettitt via Flickr

Every marshal should be stationed in a so-called *marshal booth*, a kind of protected cage that is clearly visible from the race track. These booths are located at regular intervals of ten metres (one decametre) along the track. The track is circular and L decametres long and therefore contains exactly L booths.

Not every booth needs to be used. International racing regulations require that the distance between two consecutive marshals should be at most S decametres, meaning that every S consecutive booths should contain at least one marshal. The marshals are not responsible for waving the finish flag, so it is not required (but also not forbidden) to have a marshal at the start/finish.

This leaves you with many ways of assigning marshals to the various booths along the track. Out of sheer curiosity you decide to calculate the total number of valid marshal assignments. Reduce your answer modulo 123456789 in case it gets too large.

Input

The input consists of two integers L, the length of the track, and S, the maximal distance between consecutive marshals along the track, satisfying $1 \leq S \leq L \leq 10^6$.

Output

Output the integer W, the number of ways to put marshals modulo 123456789. (Your answer must satisfy $0 \leq W < 123456789$.)

Example

Input	Output
3 2	4
2500 2000	27511813

Explanation

In the first sample test case, the four solutions are to put marshals at distances 0 and 1, at distances 0 and 2, at distances 1 and 2, or, at distances 0, 1 and 2 (in decametres) from the start.

Problem L. Sticky Situation

Source file name:	situation.c, situation.cpp, situation.java, situation.py
Input:	`Standard`
Output:	`Standard`

While on summer camp, you are playing a game of hide-and-seek in the forest. You need to designate a "safe zone", where, if the players manage to sneak there without being detected, they beat the seeker. It is therefore of utmost importance that this zone is well-chosen.

Picture by Jeanette Irwin via Flickr

You point towards a tree as a suggestion, but your fellow hide-and-seekers are not satisfied. After all, the tree has branches stretching far and wide, and it will be difficult to determine whether a player has reached the safe zone. They want a very specific demarcation for the safe zone. So, you tell them to go and find some sticks, of which you will use three to mark a non-degenerate triangle (i.e. with strictly positive area) next to the tree which will count as the safe zone. After a while they return with a variety of sticks, but you are unsure whether you can actually form a triangle with the available sticks.

Can you write a program that determines whether you can make a triangle with exactly three of the collected sticks?

Input

The first line contains a single integer N , with $3 \leq N \leq 2 * 10^4$, the number of sticks collected.

Then follows one line with N positive integers, each less than 2^{60} , the lengths of the sticks which your fellow campers have collected.

Output

Output a single line containing a single word: `possible` if you can make a non-degenerate triangle with three sticks of the provided lengths, and `impossible` if you can not.

Example

Input	Output
3 1 1 1	possible
5 3 1 10 5 15	impossible