Northwestern Europe Regional Contest 2021

NWERC 2021

November 21, 2021

Problems

Do not open before the contest has started.

Problem A
Access Denied
Time limit: 2 seconds

Computer passwords have been around for a long time. In fact, 60 years ago CTSS was one of the first computers with a password. The implementation of this was very simple. In CTSS the password was stored in plain text in a file on disk. When logging in, the user would enter a password. The computer would then compare the password to the password on disk. If the comparison failed, it would deny access, if it succeeded, access would be allowed. Researchers at MIT were quick to discover several security flaws in this password system. We will explore one of them, the timing attack.

IBM 7090 console (Public Domain)

In a timing attack, we exploit that we can deduce a computation path from the time it takes to do the computation. In CTSS the password check was done using a simple string matching algorithm, similar to this:

```
bool CheckPassword(string pwd1, string pwd2) {
    if (pwd1.Length != pwd2.Length) {
        return false;
    }
    for (int i = 0; i < pwd1.Length; i++) {
        if (pwd1[i] != pwd2[i]) {
            return false;
        }
    }
    return true;
}
```

For the purpose of this problem, we will use a (very) simplified timing model and the above algorithm. The timing model looks as follows:

- A branching statement (if or for) takes 1 ms.
- An assignment, or update of a memory address takes 1 ms.
- A comparison between two memory addresses takes 3 ms.
- A return statement takes 1 ms.

The password consists of between 1 and 20 English letters, upper or lower case, and digits.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

- Your program first sends a password string, consisting of between 1 and 20 English letters, upper or lower case, and digits.

- Depending on if the password is correct, the interactor then responds with either:
  - If the password is correct; "`ACCESS GRANTED`". Your program should then exit cleanly.
  - If the password is incorrect; "`ACCESS DENIED (`t` ms)`", where t is the time it took to verify the password in ms. Your program can then make another guess.

Make sure you flush the buffer after each write. You can guess at most $2\,500$ times. A testing tool is provided to help you develop your solution.

Read	Sample Interaction 1	Write
		`A`
`ACCESS DENIED (5 ms)`		
		`HunFhun`
`ACCESS DENIED (41 ms)`		
		`Hunter1`
`ACCESS DENIED (68 ms)`		
		`Hunter2`
`ACCESS GRANTED`		

Problem B
Boredom Buster
Time limit: 8 seconds

A log cabin

You are stuck alone in a cabin and it is raining outside. You are bored out of your mind and the only available boredom buster is a deck of Memory cards. Playing Memory by yourself is not very fun, but you developed a single player variant that requires not only good memory but also strategy.

It goes like this: You have a shuffled deck of n cards, where n is even and each card contains a number from 1 to $n/2$. There are exactly two cards of each number. The cards are laid face down on the table and your goal is to find what number is written on each of them. In one move, you pick up two cards. Before revealing them, you randomly shuffle them so that you do not know which card was where. Then you look at the two cards. After that, you shuffle them again face down before you put them back where they were. That way, you know the numbers of the two cards and where they are, but not which card ended up where, or even what card came from which spot initially.

Your task is to write a program that will beat this game.

Interaction

This is an interactive problem. Your submission will be run against an *interactor*, which reads the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

- The interactor first sends an integer n ($2 \leq n \leq 10^5$, n is even). The only time n is not equal to 10^5 is in the sample.
- Your submission then repeatedly sends two integers x and y ($1 \leq x, y \leq n$, $x \neq y$) preceded by "?".
- The interactor replies with two integers a and b ($1 \leq a, b \leq n/2$), indicating that after the query either the xth card is a and the yth card is b, or the xth card is b and the yth card is a.
- When you are ready to print the answer, print "!" followed by n integers $a_1, a_2, \ldots, a_n$, where a_i is the number of the ith card *at the time of writing*.

You may use at most $92\,000$ moves. Printing the answer does not count as a move.

The interactor is not adaptive. Initial positions of the cards are determined by the interactor before any interaction takes place. When you make a move the two cards will be shuffled uniformly randomly by the interactor both before being shown to you and before being placed back down. The initial layout of the cards is guaranteed to be a random permutation of $1, 1, 2, 2, \ldots, n/2, n/2$.

Your submission will be run on exactly 100 test cases, each of which will have $n = 10^5$.

Make sure you flush the buffer after each write.

A testing tool is provided to help you develop your solution.

Read	Sample Interaction 1	Write
6		
		? 2 1
3 2		
		? 5 3
2 3		
		? 6 4
1 1		
		? 1 3
3 3		
		! 3 2 3 1 2 1

Problem C
Cutting Edge
Time limit: 1 second

In recent years, the automated manufacturing of various kinds of 3D objects has been a growing trend among hobbyists worldwide. Your friend Lewis has fully bought into this trend, in the sense that his garage is now lined with various kinds of 3D printers and other expensive machinery.

The newest addition to his ever expanding collection is an automated cutting machine. The machine works by cutting off material from a workpiece that is initially in the shape of a rectangular cuboid. Each cut then slices through the workpiece along a plane and only keeps the material on one of the two sides of that plane. This means that the final shape of the workpiece is necessarily a convex polyhedron.

The programming interface of Lewis' machine is fairly particular. Instead of directly specifying the planes along which the piece should be cut, the user inputs a list of points and the machine then computes the cutting planes such that all the points belong to the final shape and the volume of the shape is minimal. Formally, it computes the convex hull of the input points. While this setup can be quite convenient for many applications, it is also a bit restrictive because the machine only allows using integer multiples of 1 mm for the coordinates of the points.

Lewis wants to use his machine to cut gaming pieces for a tabletop game. He does not particularly care about the shape of the pieces, but he does require them to have a specific weight. The workpieces have constant density, so that he just needs to ensure that the final shape has a specific volume. Still, this proves to be challenging because of the machine's input requirements. Help Lewis find a valid input for his cutting machine.

Input

The input consists of:

- One line with four integers a, b, c and v ($1 \leq a, b, c \leq 10^6, 1 \leq v \leq 6 \cdot a \cdot b \cdot c$), where the workpiece has dimensions a mm $\times$ b mm $\times$ c mm and the final shape must have a volume of $\frac{v}{6}$ mm^3.

Output

Output an integer n ($4 \leq n \leq 100$), followed by n points that specify the final shape. Each point is given by three integers x, y and z ($0 \leq x \leq a, 0 \leq y \leq b, 0 \leq z \leq c$). The shape is then calculated as the convex hull of these n points and its volume needs to be exactly $\frac{v}{6}$ mm^3. If there is more than one valid solution, you may output any one of them. It is guaranteed that a solution always exists. Note that outputting duplicate points is allowed.

Sample Input 1

```
1 1 1 1
```

Sample Output 1

```
4
0 0 0
1 0 0
0 1 0
0 0 1
```

Sample Input 2

```
3 1 2 7
```

Sample Output 2

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

Sample Input 3

```
2 2 2 25
```

Sample Output 3

```
9
0 0 2
2 0 0
0 2 0
1 0 2
2 1 2
2 0 1
0 2 2
1 2 2
2 1 2
```

Problem D
Dyson Circle
Time limit: 4 seconds

A Dyson Sphere is a theoretical construction around the sun or another star, that captures the entire energy output of the star. Science fiction writers have speculated that advanced civilizations will eventually build such a sphere, as the energy demands of such a society continue to grow without bounds. In our three-dimensional space, Dyson spheres are still in the realm of fiction. *Dy & Son*, the main energy company of your dimensional neighbours, has tasked you with carrying out a feasibility study in the two-dimensional world of Flatland.

Dy & Son has developed a modular Dyson Circle. It consists of independent square Dyson Units that can chain together to form a closed loop that gathers energy. Your task is to figure out how many of these Dyson units they actually need to enclose the star or stars they are interested in. Note that they want a single Dyson Circle, not a separate one for each star.

For convenience, both the stars Dy & Son wants to enclose and the Dyson Units used for this are modeled as squares of exactly 1 by 1 *Intergalactic Unit*, aligned to the *Intergalactic Coordinate System*. Your Dyson units connect if they have at least a corner in common. See Figure D.1 for an example.

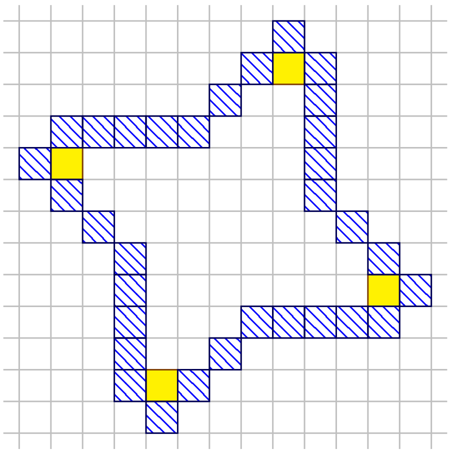


Figure D.1: Illustration of Sample Input 1: four stars (yellow squares) and an optimal Dyson Circle (dashed blue squares) surrounding them, and the remaining blackness of space shown in white.

Formally, select some squares in the plane to turn into Dyson Units, such that the remaining squares can be split into *inside* and *outside* squares. All the stars must be inside squares. The inside squares must form a contiguous region (connected via edges) and not connect to the outside squares (via edges). The outside squares form a contiguous region stretching off to infinity.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 2 \cdot 10^5$), the number of stars.
- n lines, each containing two integers x and y ($-10^6 \leq x, y \leq 10^6$), the location of the

center of a star.

No two stars are at the same location.

Output

Output the least number of Dyson units required to capture the energy of all stars in the input.

Sample Input 1

```
4
2 5
-5 2
-2 -5
5 -2
```

Sample Output 1

```
32
```

Sample Input 2

```
2
1 1
3 2
```

Sample Output 2

```
8
```

Sample Input 3

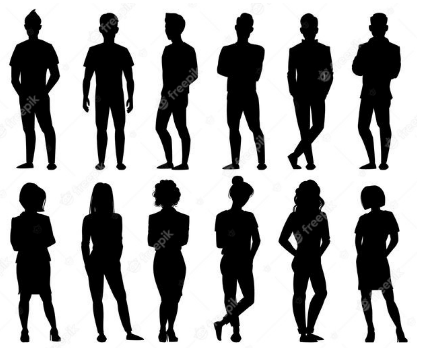
```
2
2 3
4 5
```

Sample Output 3

```
9
```

Problem E
Exchange Students
Time limit: 4 seconds

A group of n exchange students at Reykjavik University is lining up to get their group photo taken. From left to right, the heights of the students are $g_1, g_2, \ldots, g_n$. However, the photographer would like to rearrange the students such that the order of their heights becomes $h_1, h_2, \ldots, h_n$. In order to do this, the photographer will repeatedly exchange two exchange students. Two exchange students can only be exchanged if they can see each other, that is, if every student in between them has strictly smaller height than the two students to be exchanged.


Silhouettes via Freepik

Determine the minimum number of exchanges needed to arrange the students in the photographer's preferred order, and find a suitable sequence of exchanges of this minimal length. The photographer only has time for at most $2 \cdot 10^5$ exchanges. If more are needed, you only need to determine the first $2 \cdot 10^5$ exchanges.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 3 \cdot 10^5$), the number of students.
- One line with n integers $g_1, g_2, \ldots, g_n$ ($1 \leq g_i \leq 10^9$), the heights of the students.
- One line with n integers $h_1, h_2, \ldots, h_n$ ($1 \leq h_i \leq 10^9$), the order of heights the photographer prefers.

It is guaranteed that $(h_1, \ldots, h_n)$ is a permutation of $(g_1, \ldots, g_n)$.

Output

First output an integer s, the minimum number of exchanges needed. Then print $\min(s, 2 \cdot 10^5)$ exchanges, each consisting of two integers i and j, the one-based positions of the students that must be exchanged in this step.

If there are multiple valid solutions, you may print any one of them.

Sample Input 1	Sample Output 1

3
2 1 3
3 1 2
	1
	1 3

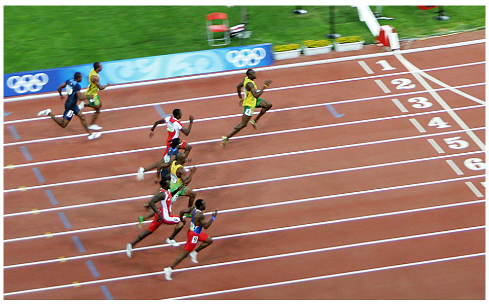
Sample Input 2

```
5
6 7 5 9 6
9 6 7 6 5
```

Sample Output 2

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

Problem F
Flatland Olympics
Time limit: 4 seconds



2008 Beijing Olympics by PhotoBobil

It is the day after Olympia, and you—as the organizer—are happy that *everything* worked well in these troublesome times. Well, not everything....

Since this morning e-mails have been filling up your inbox, containing complaints about obscured views during the most important race: the 100-meter dash. They demand their money back, or threaten exposing you on social media. To make things worse, spectators have not just complained once, but they have sent you a separate e-mail for every person that blocked their view at some point during the race! They even wrote multiple e-mails when two or more people blocked their view at the same time. And not only that, some visitors complained to the main sponsor *Dy & Son* who in turn has urged you to improve the situation.

Since you expect that a greater number of visitors will be allowed to spectate at the next Olympic games, you assume that there will be even more complaints if you do not address this issue. If the situation will be too bad, you may even lose your sponsor Dy & Son. Therefore, you decide to count the number of complaints beforehand. To do this, you model the running track as a straight line segment, and count the maximal number of complaints you could get based on the seating of the visitors. Depending on the number of complaints you expect, you will determine if you need to rework the seating or just reconfigure your spam blocker and try to find a new sponsor.

Input

The input consists of:

- One line containing four integers x_s, y_s, x_e and y_e ($|x_s|, |y_s|, |x_e|, |y_e| \leq 10^9$), where $s = (x_s, y_s)$ is the starting point of the running track and $e = (x_e, y_e)$ is the end point of the running track. Both s and e belong to the running track.
- One line containing an integer n ($1 \leq n \leq 10^5$), the number of visitors.
- n lines, each containing two integers x and y ($|x|, |y| \leq 10^9$), where (x, y) is the location of the seat of a visitor.

It is guaranteed that the track has a positive length, i.e. $s \neq e$. Further, you can assume that all visitors are seated at distinct locations and that no visitor is seated on the track.

Output

Output the total number of complaints that you would receive for the given seating.

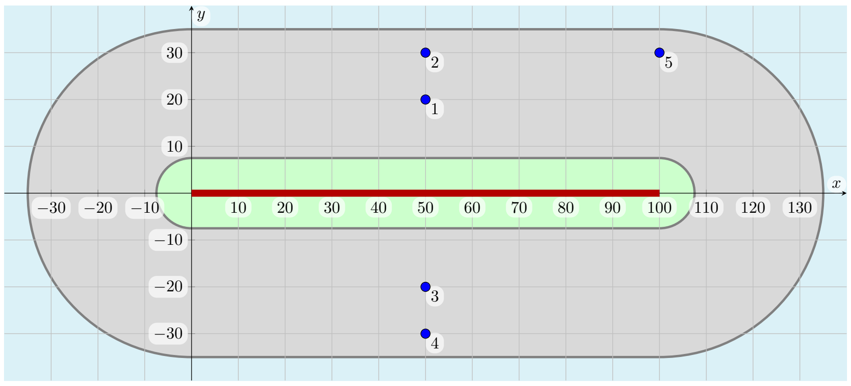


Figure F.1: Illustration of Sample Input 2. The running track is drawn as a red line and the seats of the visitors are highlighted in blue. The second visitor will complain about the first visitor and the fourth visitor will complain about the third visitor.

Sample Input 1

```
0 0 100 0
4
50 20
50 30
50 50
120 0
```

Sample Output 1

```
3
```

Sample Input 2

```
0 0 100 0
5
50 20
50 30
50 -20
50 -30
100 30
```

Sample Output 2

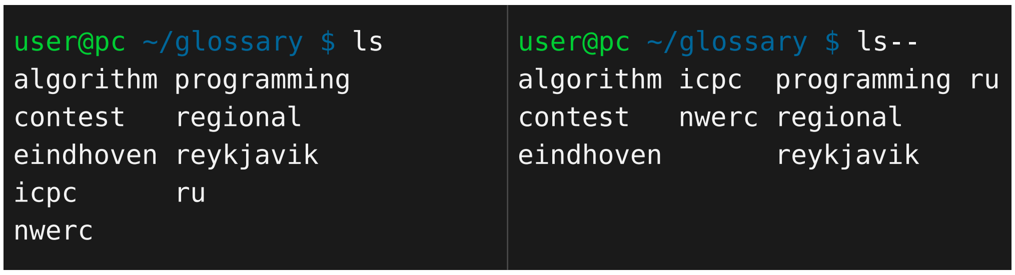
```
2
```

Problem G
Glossary Arrangement
Time limit: 5 seconds

In Unix based operating systems, one of the most frequently used commands is `ls`, which displays a list of all the files in a directory in lexicographic order. In the most basic form, `ls` prints each filename on a separate line, but to improve readability and save screen space, the list is usually split into multiple columns which are displayed side by side.

A friend of yours is writing a book about NWERC and has put you in charge of editing the glossary of relevant terms at the end of each chapter. Each glossary must use the same multi-column layout as `ls`, so you decided to go for the lazy option: For each word in the glossary, you created an empty file with that name, and simply let `ls` do the heavy lifting.

Unfortunately, your friend is not satisfied with your layouts and complains that some of them take up too much vertical space. The problem with your approach is that `ls` always forms columns of equal height, except for the last column, which may be shorter. This sometimes ends up using more lines than would be needed if the column heights could be chosen more freely:



```
user@pc ~/glossary $ ls
algorithm programming
contest   regional
eindhoven reykjavik
icpc      ru
nwerc
```

```
user@pc ~/glossary $ ls--
algorithm icpc  programming ru
contest   nwerc regional
eindhoven       reykjavik
```

Figure G.1: Saving two lines using variable column heights.

Begrudgingly, you decide to write your own improved version of `ls`, `ls--`, that also displays the contents of a directory in lexicographic order, but uses variable column heights to always achieve the lowest possible number of lines.

Columns have a fixed width, which is the length of the longest filename within the column, and are separated by a single space. The names in each column must be left-aligned and padded to the column width using spaces. Also, the terminal you are using has a fixed width of w characters which the table may not exceed.

Given the contents of a directory as a list of filenames, find an optimal column layout that minimizes the number of lines needed to print the entire list.

Input

The input consists of:

- One line with two integers n and w ($1 \leq n, w \leq 5\,000$), the number of files and the width of the terminal.
- n lines, each with one filename s ($1 \leq |s| \leq w$, s consists of lowercase English letters).

The filenames are distinct and in lexicographic order. The total number of letters is at most 10^6.

Output

Output an optimal way of listing the given filenames:

- One line with two integers r and c, the number of lines and the number of columns used.
- One line with c positive integers $a_1, a_2, \ldots, a_c$, the widths of the columns.
- The table of filenames, subject to the following formatting:
  - There are c columns, where column i has width a_i for each i, and within each column there are at most r filenames that are aligned on the left and grouped at the top.
  - The filenames are aligned using spaces, with exactly one space between columns.
  - The total width of the table is at most w.
  - When reading column by column, the filenames appear in lexicographic order.

Note that unlike in other problems, you strictly need to follow the above formatting rules for whitespace. However, we still allow trailing whitespace at the end of each line, even if this whitespace exceeds the width w.

Sample Input 1

```
9 30
algorithm
contest
eindhoven
icpc
nwerc
programming
regional
reykjavik
ru
```

Sample Output 1

```
3 4
9 5 11 2
algorithm icpc  programming ru
contest   nwerc regional
eindhoven       reykjavik
```

Sample Input 2

```
6 10
aaa
bb
ccccc
ddd
eeeee
fffff
```

Sample Output 2

```
4 2
3 5
aaa ccccc
bb  ddd
    eeeee
    fffff
```

Sample Input 3

```
5 15
pppp
ppppp
pq
pqab
xyzff
```

Sample Output 3

```
2 3
5 2 5
pppp  pq pqab
ppppp    xyzff
```

Problem H
Heating Up
Time limit: 3 seconds

Chilli pizza by Rahul Upadhyay, Unsplash

Jonas just entered his first chilli-eating contest. He is presented with a pizza consisting of n slices, numbered from 1 to n, each containing a selection of chilli peppers. Initially slices i and $i+1$ are adjacent on the plate (where $1 \le i < n$), and so are slices 1 and n. According to the contest rules only one slice can be consumed at a time, and the slice must be finished in its entirety before a new slice is started. Jonas is allowed to pick any slice to eat first, but after that he is only allowed to eat slices that have at most one remaining adjacent slice.

The spiciness of each slice is measured in Scoville Heat Units (SHU). Jonas has a certain spiciness tolerance, also measured in SHU, which corresponds to the spiciness of the spiciest slice that Jonas can tolerate eating. He has also noticed that, after eating a slice of k SHU, his tolerance immediately increases by k.

In order to win the contest, Jonas would like to finish all the slices of his pizza. Help him determine the minimum initial spiciness tolerance necessary to do so while abiding by the contest rules.

Input

The input consists of:

- One line with an integer n ($3 \le n \le 5 \cdot 10^5$), the number of pizza slices.
- One line with n integers $s_1, s_2, \ldots, s_n$ ($0 \le s_i \le 10^{13}$), where s_i is the spiciness of the ith slice in SHU.

Output

Output the minimum initial spiciness tolerance in SHU that Jonas needs in order to be able to eat all slices of the pizza.

Sample Input 1
```
5
5 0 10 6 1
```
Sample Output 1
```
4
```

Sample Input 2
```
7
20 23 7 2 3 7 1
```
Sample Output 2
```
2
```

This page is intentionally left blank.

Problem I
IXth Problem
Time limit: 1 second

Emily recently learned about the Roman Empire and its civilization at school. One aspect that was especially fascinating to her is the number system that they used, the *Roman numerals*. The Roman number system uses seven distinct digits, each representing a different value and denoted by a letter, where `I` is 1, `V` is 5, `X` is 10, `L` is 50, `C` is 100, `D` is 500 and `M` is $1\,000$. Multiples of 1, 10, 100 and $1\,000$ are then written according to the following table:

$\times$	1	2	3	4	5	6	7	8	9
1	I	II	III	IV	V	VI	VII	VIII	IX
10	X	XX	XXX	XL	L	LX	LXX	LXXX	XC
100	C	CC	CCC	CD	D	DC	DCC	DCCC	CM
1 000	M	MM	MMM						

Most of the numerals in this table are formed additively, i.e. by summing the values of the digits. For example, `LXX` is $50 + 10 + 10 = 70$. Columns 4 and 9, however, use so-called subtractive notation, where `IV` is read as $5 - 1$, `IX` is read as $10 - 1$, and so on.

Each number from 1 to $3\,999$ is written as a combination of numerals from the table, using at most one numeral from each row and going from bottom to top. For instance, $2\,021$ is `MMXXI` and 594 is `DXCIV`. Note that in this number system it is not possible to write numbers greater than $3\,999$ and also that subtractive notation can only be used in the six cases above (e.g. `IC` is not considered a valid Roman numeral).

Emily found a bunch of old Scrabble sets in her attic. She threw out all the tiles with letters other than the Roman digits and started forming Roman numerals from the remaining tiles. It is easy to form valid numerals from the tiles by using just one tile per numeral, but what is the minimal number of numerals that can be formed while still using all the available tiles?

Input

The input consists of:

- One line with seven integers m, d, c, ℓ, x, v and i ($0 \leq m, d, c, \ell, x, v, i \leq 10^{18}$), which respectively are the number of `M`, `D`, `C`, `L`, `X`, `V` and `I` tiles that must be used.

There is at least one tile, that is $m + d + c + \ell + x + v + i \geq 1$.

Output

Output an integer n, the minimal possible number of Roman numerals that can be formed while using all of the tiles in the input. Then output an optimal solution in the following format.

- An integer k, the number of distinct Roman numerals used in this solution.
- k pairs of a Roman numeral and a positive integer indicating how often this numeral is used in this solution.

The solution must consist of exactly n numerals in total and must use exactly the specified number of each letter. The k Roman numerals in the solution must be distinct. You do not need to minimize k. If there is more than one optimal solution, any one of them will be accepted.

Sample Input 1

```
4 1 7 1 3 1 3
```

Sample Output 1

```
2
2
MMDCCCLXX 1
MMCCCXCVIII 1
```

Sample Input 2

```
0 0 0 300 2000 1000 2100
```

Sample Output 2

```
1000
2
XXVIII 700
LXXV 300
```

Problem J
Jet Set
Time limit: 1 second

Your wealthy friends love to brag about all their travelling. Every time you see them, they have visited some new exotic place you have never heard of. All of them are all too happy to tell you they have been *all* around the world—but you're not so sure about that. Have these jetsetters made a real *circumnavigation*?

There exist many different definitions of what exactly constitutes a circumnavigation, but for the purposes of this problem we consider a circumnavigation a journey starting and ending at the same point and visiting all meridians (lines of longitude) along the way. Note that the North and South Pole are part of every meridian.

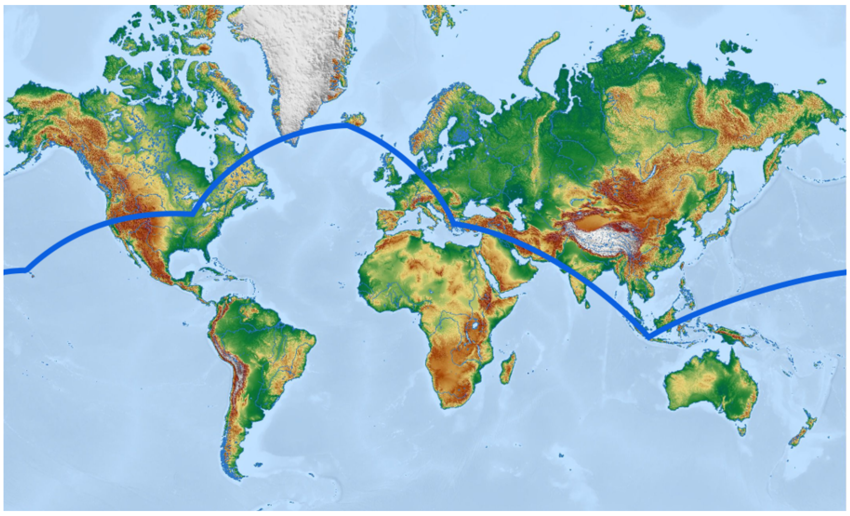


Figure J.1: Illustration of Sample Input 1, giving a circumnavigation starting and ending in Reykjavík, with additional waypoints in Athens, Jakarta, Honolulu and Chicago.

Amelia, one of your rich friends, gave you a log of her flights in the form of a list of waypoints. Her trip started at the first waypoint, visited the remaining waypoints in order, and finally went back from the last waypoint to the first. Between consecutive waypoints, Amelia always travelled along the shortest circular arc connecting the two points. Find out whether Amelia's trip can be considered a circumnavigation in the above sense, and if not find a meridian that Amelia never visited. In that case, always report exactly an integer-valued or half-integer-valued meridian.

Input

The input consists of:

- One line with an integer n ($2 \leq n \leq 1\,000$), the number of waypoints.
- n lines, each with two integers ϕ and λ ($-90 < \phi < 90, -180 \leq \lambda < 180$), the latitude and longitude of one of the waypoints.

No two consecutive waypoints along the route are equal or antipodes (opposite points on the sphere) of each other.

Output

If the route is a valid circumnavigation, output `yes`. Otherwise, output `no`, followed by a longitude λ ($-180 \leq \lambda < 180$) which the route never visited. The longitude must end in either `.0` or `.5`.

Sample Input 1

```
5
64 -22
38 24
-6 107
21 -158
42 -88
```

Sample Output 1

```
yes
```

Sample Input 2

```
2
80 30
75 -150
```

Sample Output 2

```
yes
```

Sample Input 3

```
4
45 0
0 -170
-45 0
0 170
```

Sample Output 3

```
no 173.5
```

Problem K
Knitpicking
Time limit: 1 second

Gillie in one of his resting places By Dwight Sipler (cc by-sa)

Kattis has many pairs of nice, warm, knit socks in her sock drawer that are perfect for the winter. These socks come in a wide range of colours and types, and have all been mixed together. Each morning Kattis needs to pick two matching socks.

To find matching socks, she simply randomly takes single socks out of the drawer until she has a matching pair. It may take a long time, for example when she keeps drawing right socks without a matching left one. How long does she need to keep drawing socks until she is guaranteed to have a pair to wear?

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 1\,000$), the number of groups of identical socks.
- n lines, each describing a group of identical socks with the following:
  - A string i, the type of the socks in the group. The type i consists of between 1 and 20 lowercase English letters. Socks with the same type are considered compatible for fashion purposes.
  - A string j, the fit of the socks in the group, which is either `left`, `right` or `any`, indicating whether the socks fit on the left foot, the right foot or any foot.
  - An integer k ($1 \leq k \leq 1\,000$), the number of socks in the drawer that are of this type and fit.

A given fit of a given type of sock appears at most once in the input.

Output

Output the minimum number of socks Kattis needs to draw to be guaranteed to get a matching pair. If it is not possible to get a matching pair at all, output `impossible`.

Sample Input 1

```
3
fuzzy any 10
wool left 6
wool right 4
```

Sample Output 1

```
8
```

Sample Input 2

```
3
sports any 1
black left 6
white right 6
```

Sample Output 2

```
impossible
```

Sample Input 3

```
2
warm any 5
warm left 3
```

Sample Output 3

```
4
```

Problem L
Lucky Shirt
Time limit: 2 seconds

Stack of clothes via pxfuel.com

You, a frantic competitive programmer, have collected a large number of T-shirts by competing in various programming contests. In fact, you have so many, that these are the only T-shirts you wear anymore. You keep them neatly folded in a large stack in your oversized closet. Each morning, you pick the top shirt from your stack of shirts to wear that day. At the end of the day, you throw the shirt in the laundry basket.

In order to keep a fresh supply of clean shirts, you sometimes also do laundry at night, washing all shirts in the laundry basket (including the one you wore that day). This is not according to some neat schedule however; the number of days between your wash cycles is a uniformly random integer between 1 (in which case you would wash only a single shirt) and the number of shirts you have. After washing your clothes, you put them back on top of the stack in a uniformly random order.

It is now the night after a successful programming contest, and you decide that the T-shirt you got there is your lucky shirt from now on. You wonder when you will be able to wear it again, and entertain yourself with thoughts about the good fortune you will receive when you do. You just completed your laundry and put all your shirts on the stack. Knowing the current position of your lucky shirt in the stack, what is the expected position of your lucky shirt after k more washing cycles?

Input

The input consists of:

- One line containing three integers n ($1 \leq n \leq 10^6$), the number of shirts you have, i ($1 \leq i \leq n$), the position of your lucky shirt counted from the top, and k ($1 \leq k \leq 10^6$), the number of washing cycles.

Output

Output the expected position of your lucky shirt after k washing cycles. Your answer should have an absolute or relative error of at most 10^{-6}.

Sample Input 1	Sample Output 1
`3 2 2`	`1.833333333`

Sample Input 2	Sample Output 2
`5 1 1`	`2.0`

Sample Input 3	Sample Output 3
`10 7 100`	`5.499986719`