Division 2

Hosted by:

College of Charleston

Florida International University

Kennesaw State University

University of West Florida

Congruent Numbers

A *congruent* number is an integer that is the area of some right triangle where the length of each side of the triangle is a *rational* number. For this problem, we'll only consider the legs of the right triangle, and not the hypotenuse.

A *rational* number is a fraction, ***p***/***q***, where ***p***, the numerator, and ***q***, the denominator, are integers. Note that if ***q*** = 1, then ***p*/1** is an integer, so any integer is a *rational* number.

Given two *rational* numbers which are the non-hypotenuse legs of a right triangle, determine if the area of that triangle is a *congruent* number. For the purposes of this problem, it is not necessary for the length of the hypotenuse to be a *rational* number.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with four integers ***p1***, ***q1***, ***p2*** and ***q2*** ($1 \le p1,q1,p2,q2 \le 100{,}000$) where ***p1*/*q1*** and ***p2*/*q2*** are the *rational* numbers which are the sides of a right triangle.

Output

Output a single integer, which is **1** if the area of the triangle is an integer, **0** if not. Note that the area has to be an integer, not just a *rational* number.

Sample Input	Sample Output
3 1 4 1	1
15 1 28 3	1
1 2 3 4	0
1 1 10 1	1

Unloaded Die

Consider a standard six-sided die, with sides labelled 1 through 6. We consider a die to be *fair* if each of its sides is equally likely to be landed on after rolling it. We consider a die *loaded* if its fairness is compromised. For example, if the side marked 6 twice as likely to come up as any other side, we are dealing with a *loaded* die.

For any die, define the expected value of rolling the die to be equal to the average of the values of the sides weighted by the probability of those sides coming up. Intuitively, this is the number you would get if you rolled the die many times and averaged all the results together. A *fair* die has an expected result of 3.5. That is, since all sides are weighed the same, they each have probability of 1/6, and we get:

1/6 + 2/6 + 3/6 + 4/6 + 5/6 + 6/6 = 3.5

Suppose you are given a *loaded* die, and you would like to "unload" it to make it more closely resemble a *fair* die. To do so, you can erase one side's label and replace it with a new (real) number. You want to do so in such a way that

1) The expected result of rolling the die is 3.5, just like a *fair* die, and
2) The absolute value of difference between the old label and the new label on the side you change is as small as possible.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with six decimal numbers, where the i^{th} number (i = 1..6) is the probability that the side with value i is rolled. All of these numbers will be between **0.0** and **1.0**, and they are guaranteed to sum to **1.0**.

Output

Output a single number on a single line: the absolute value of the difference between the label you erase and the label you write in. Output this number to exactly 3 decimal places, rounded.

Sample Input	Sample Output
0.16666 0.16667 0.16667 0.16666 0.16667 0.16667	0.000
0.2 0.2 0.1 0.2 0.2 0.1	1.000

Halfway

A friend of yours has written a program that compares every pair of... something. With ***n*** items, it works like this: First, it prints a 1, and it compares item 1 to items 2, 3, 4, ..., ***n***. It then prints 2, and compares item 2 to items 3, 4, 5, ..., ***n***. It continues like that until every pair has been compared exactly once. If it compares item ***x*** to item ***y***, it will not later compare item ***y*** to item ***x***. It will not compare any item to itself.

Your friend wants to know when his program is halfway done. Assuming that all comparisons take the same amount of time, what will be the last number printed when the program is exactly halfway done? For an odd number of comparisons, this is when it's doing the middle comparison. For an even number, it's the first of the two middle comparisons. Note that since the earlier items have more comparisons than the later items, the answer is **not** simply ***n***/2.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with single integer ***n*** ($2 \le n \le 10^9$), indicating the number of items your friend is comparing.

Output

Output a single integer representing the last number your friend's program prints before it performs the halfway comparison.

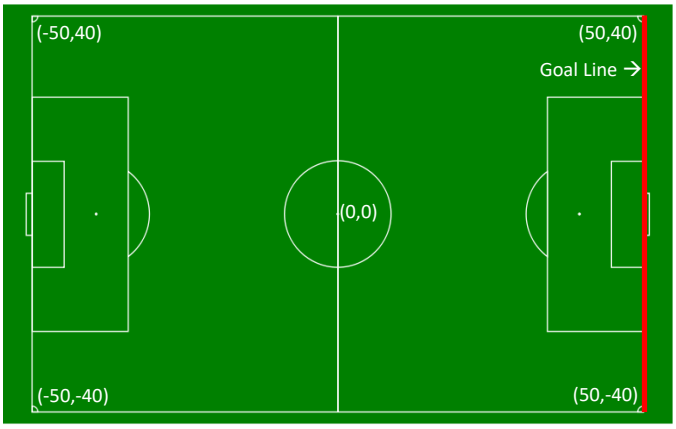
Sample Input	Sample Output
4	1
7	2
10	3
1919	562
290976843	85225144

Law 11

Of the seventeen enumerated laws of Soccer, Offside as described by Law 11 is perhaps the most contentious. In short, offside consists of two conditions: the offside position, and a state of play that turns the offside position into an offense. In an attempt to consistently enforce offside offenses (and generally ratchet down the poor sportsmanship of parents who consider themselves smarter than the referees, but too selfish to serve as referees themselves), your team has been contacted to implement the analysis algorithm for *Offside Enforcement Technology*, soon to be known as *OET*. This phase of OET will focus solely on the Offside Position, which may be stated as:

- Any part of a player's head, body, or feet (but not hands or arms) is in the opponents' side of the field (excluding the halfway line), AND
- Any part of a player's head, body, or feet (but not hands or arms) is closer to the opponents' goal line than BOTH the ball AND all but 1 opponent

For the purposes of OET, consider each player to be a single point, and determine if the offense is in an offside position. The field will be 100m × 80m, with the point (**0**,**0**) being the center of the halfway line, and the opponent's goal line running from (**50**,**-40**) to (**50**,**40**).



Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will have exactly 23 lines of input. Each line will contain two integers, ***x*** and ***y*** (-50 ≤ ***x*** ≤ 50, -40 ≤ ***y*** ≤ 40), which indicate the position of the ball or a player on the field in meters. The first line will indicate the position of the ball, then the next 11 will be the offense and the last 11 will be the defense. No two players will be in the same position.

Output

Output a single integer, **1** of the play is offside, and **0** if it is not.

Sample Input	Sample Output
9 4 -45 -40 -11 18 44 -16 36 -32 -29 -30 7 -33 47 27 5 -37 -11 18 -15 -6 -7 -1 1 22 -39 11 -9 -9 10 38 -43 -8 -16 -29 43 -27 2 -27 -4 -30 49 -15 -48 10	1
9 4 -45 -40 -11 18 44 -16 36 -32 -29 -30 7 -33 47 27 5 -37 -11 18 -15 -6 -7 -1 1 22 -39 11 -9 -9 10 38 -43 -8 -16 -29 48 -27 2 -27 -4 -30 49 -15 -48 10	0

Long Long Strings

To store DNA sequences your company has developed a general *LongLongString* class that can store strings with a theoretically unlimited number of characters. Individual character can be referenced by index. The leftmost character is at position 1. The class can execute simple programs with three basic operations:

- **insert**(***p***, ***c***) - inserts the character ***c*** at position ***p***. All characters past ***p*** are pushed to the right by 1.
- **delete**(***p***) – deletes the character at position ***p***. All character past ***p*** are pushed to the left by 1.
- **end** – ends the program

Your job is two write a program that compares two string editing programs and determines if they are *different*. They are *not different* if, when applied to **any** string, they produce identical results. Otherwise, they are *different*.

For example:

- [**delete**(1) **delete**(2) **end**] and [**delete**(3) **delete**(1) **end**] are *not different*
- [**delete**(2) **delete**(1) **end**] and [**delete**(1) **delete**(2) **end**] are *different*.
- [**insert**(1,X) **delete**(1) **end**] and [**end**] are *not different*.
- [**insert**(14,B) **insert**(14,A) **end**] and [**insert**(14,A) **insert**(15,B) **end**] are *not different*
- [**insert**(14,A) **insert**(15,B) **end**] and [**insert**(14,B) **insert**(15,A) **end**] are *different*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of exactly two programs, one after the other. Each program will end with an **end** statement, and each will be no longer than **2,000** instructions. Each line of each program will consist of exactly one of:

`I` ***p*** ***c***

or

`D` ***p***

or

`E`

Where ***p*** ($1 \le p \le 10^{10}$) is a position in the string, and ***c*** is a single capital letter (**A**..**Z**). `I` means **insert**, `D` means **delete**, and `E` means **end**.

Output

Output a single integer, **1** if the programs are *different*, and **0** of they are *not different*.

Sample Input	Sample Output
D 1 D 2 E D 3 D 1 E	0
D 2 D 1 E D 1 D 2 E	1
I 1 X D 1 E E	0
I 14 B I 14 A E I 14 A I 15 B E	0
I 14 A I 15 B E I 14 B I 15 A E	1

Move Away

Tommy has just completed college and is looking for his first job. A priority in his life is living close to his friends, but he wants to live as far away from his parents as possible.

You are given the locations of Tommy's friends and the maximum distance he would be willing to live away from each friend. You also know that Tommy's parents live at (0, 0) in the coordinate plane. Determine how far Tommy can live from his parents. (There will always be at least one point meeting these requirements.)

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with a single integer ***n*** ($1 \le n \le 50$), representing the number of friends Tommy has. The next ***n*** lines will each contain three integers: ***x***, ***y*** ($-1{,}000 \le x, y \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), representing the (***x***, ***y***) coordinate of his friend and the maximum distance ***d*** he is willing to live away from that friend.

Output

Output a single decimal number on a single line, equal to the maximum distance he can live from his parents while still being close enough to all of his friends. Output this number to exactly 3 decimal places, rounded.

Sample Input	Sample Output
4 1 0 1 0 1 1 -1 0 1 0 -1 1	0.000
2 -1 0 1000 2 0 1000	999.999

Purple Rain

Purple rain falls in the magic kingdom of Linearland which is a straight, thin peninsula. On close observation however, Prof. Nelson Rogers finds that actually it is a mix of Red and Blue drops. In his zeal, he records the location of each of the raindrops to fall with its corresponding color in different locations along the peninsula. He wants answer the following question: which section of Linearland had the *least* purple rain? That is, which section had the greatest difference between red rain and blue rain?

After some thought, he decides to model the problem as follows: Divide the peninsula into ***n*** sections and describe it as a sequence of **R** or **B** values depending on whether the rainfall in that section is primarily red or blue. Then, find the part consisting of consecutive sections where the absolute difference of the count of **R**s and **B**s is maximized.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with a string ***s*** ($1 \le |\boldsymbol{s}| \le 100{,}000$), where every character in ***s*** is either a capital **B** or a capital **R**. This string describes the peninsula, from west to east.

Output

Output two integers, indicating the start and end of the part of the peninsula which maximizes the difference between **R**s and **B**s. The first character of ***s*** is at position **1**, and the last is at position ***n***. Output the smaller index first. If there are multiple parts that feature the same maximal absolute difference, print the one with the smallest starting position. If there are multiple such parts starting at that same smallest starting position, print the shortest of those.

Sample Input	Sample Output
BBRRBRRBRB	3 7
BBRBBRRB	1 5

Rainbow Roads

Your city has decided to spice up its image – by painting its roads different colors!

Your city will paint a road the same uniform color between two intersections if there are no intersections in between (for our purposes, we'll refer to dead ends and cul-de-sacs as intersections), but along its full length, a road may be painted many different colors. Interestingly, there is exactly one path along its roads between any two intersections in the city.

The city council wants to label some intersections as *Super* intersections, and put up signs designating them so. They consider a path a *Rainbow* if there are no intersections along the path where the road in and the road out are the same color. An intersection is a *Super* intersection if the path from that intersection to every other intersection is a *Rainbow*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line of input containing a single integer n ($1 \le n \le 50{,}000$) which is the number of intersections. The intersections are numbered 1..n.

Each of the next n-1 lines will contain three integers, a, b and c ($1 \le a,b,c \le n$, $a \ne b$), which describe a road between intersection a and intersection b with color c. It is guaranteed that the given roads satisfy the constraint that there is exactly one path between any pair of intersections. The roads are two-way roads, so a road from a to b also goes from b to a.

Output

On the first line, output a single integer indicating the number of *Super* intersections. On the following lines, output a list of integers, one per line. These are the *Super* intersections. Print them in numerical order, smallest to largest.

Sample Input	Sample Output
8 1 3 1 2 3 1 3 4 3 4 5 4 5 6 3 6 7 2 6 8 2	4 3 4 5 6
8 1 2 2 1 3 1 2 4 3 2 7 1 3 5 2 5 6 2 7 8 1	0
9 1 2 2 1 3 1 1 4 5 1 5 5 2 6 3 3 7 3 4 8 1 5 9 2	5 1 2 3 6 7
10 9 2 1 9 3 1 9 4 2 9 5 2 9 1 3 9 6 4 1 8 5 1 10 5 6 7 9	4 1 6 7 9

Arithmetic Sequences

An *Arithmetic Sequence* of integers is one in which the next number in the sequence is obtained by adding a constant to the current number. For example, this is an arithmetic sequence (the constant is 7):

3, 10, 17, 24, 31, …

Given a part of an arithmetic sequence with some numbers missing, fill in the missing numbers.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with exactly ten integers. Eight of them will be **0**, the other two will be positive. The two positive integers may be anywhere among the ten integers, and will be no larger than **1,000**. The **0** values represent missing values from the sequence.

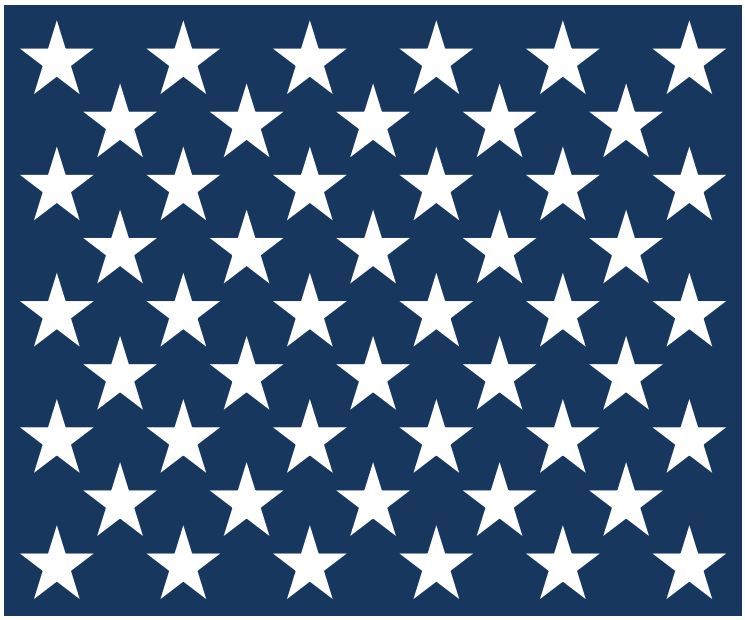
Output

If it is possible to complete the sequence with integers, then output ten integers on a single line, with a single space between them, by replacing the **0** values with the correct numbers. If it is not possible to complete the sequence with integers, simply output a single **-1**. Although the two non-zero inputs are positive, the rest of the sequence might not be. Likewise, while the two non-zero inputs are ≤**1,000**, the rest of the sequence might not be.

Sample Input	Sample Output
5 0 15 0 0 0 0 0 0 0	5 10 15 20 25 30 35 40 45 50
5 0 0 15 0 0 0 0 0 0	-1
0 0 0 15 0 3 0 0 0 0	33 27 21 15 9 3 -3 -9 -15 -21
0 0 19 0 0 0 0 0 19 0	19 19 19 19 19 19 19 19 19 19

Star Arrangements

The recent vote in Puerto Rico favoring US statehood has made flag makers very excited. An updated flag with 51 stars rather than the current 50 would cause a huge jump in US flag sales. The current pattern for 50 stars is five rows of 6 stars (30), interlaced with four offset rows of 5 stars (20).



This pattern has some appealing properties: adjacent rows differ by no more than one star. This star arrangement can be represented uniquely in a compact notation by the first two rows' star counts: **6 5**.

A 51-star flag can have three rows of 9 stars, interlaced with three rows of 8 stars (27 + 24 = 51), or **9 8**. If Guam were to also become a state, a 52-star flag could have 13 rows of 4 stars, or **13 13** (because there are 13 stars in each of the first 2 rows).

A visually appealing star field satisfies these conditions:

1) There are at least 2 rows of stars.
2) All odd numbered rows have the same number of stars.
3) All even numbered rows have the same number of stars.
4) The difference in the number of stars between any two adjacent rows is either always 0, or always 1
5) The first row cannot have fewer stars than the second, nor can it have only 1 star.

Given a number of states, describe all possible appealing star fields in compact notation.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single line with single integer ***n*** ($3 \le n \le 10^6$), indicating the number of stars.

Output

Output all possible star field arrangements, in compact notation. Output them one arrangement per line, with a single space between the two integers. Output them in increasing order of the first integer. If the first integer of two arrangements is the same, output them in increasing order of the second integer.

Sample Input	**Sample Output**
3	2 1
50	2 1 2 2 3 2 5 4 5 5 6 5 10 10 13 12 17 16 25 25
51	2 1 3 3 9 8 17 17 26 25
52	2 2 4 4 7 6 13 13 26 26

Treasure Map

You have found a treasure map! The map leads you to several gold mines. The mines each produce gold each day, but the amount of gold that they produce diminishes each day. There are paths between the mines. It may take several days to go from one mine to another. You can collect all of the day's gold from a mine when you are there, but you have to move on, you cannot stay for multiple days at the same mine. However, you can return to a mine after leaving it.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers ***n*** ($2 \le n \le 1{,}000$) and ***m*** ($1 \le m \le 1{,}000$), where ***n*** is the number of mines, and ***m*** is the number of paths.

The next ***n*** lines will each describe a mine with two integers, ***g*** ($1 \le g \le 1{,}000$) and ***d*** ($1 \le d \le 1{,}000$), where ***g*** is the amount of gold mined on day 1, and ***d*** is the amount by which the gold haul diminishes each day. For example, if ***g***=9 and ***d***=4, then on day 1, the mine produces 9, on day 2 it produces 5, on day 3 it produces 1, and from day 4 on, it produces 0 (the mines cannot produce negative amounts of gold). The mines are numbered 1..***n*** in the order that they appear in the input, and you start at mine 1 on day 1.

The next ***m*** lines will each describe a path with three integers, **a**, ***b*** ($1 \le a < b \le n$) and ***t*** ($1 \le t \le 100$), where the path goes from mine ***a*** to mine ***b***, and takes ***t*** days to traverse. The paths go in both directions, so that a path that goes from ***a*** to ***b*** can also be used to go from ***b*** to ***a***.

Output

Output a single integer, which is the maximum amount of gold that you can collect.

Sample Input	Sample Output
2 1 10 1 10 2 1 2 1	42
3 2 10 5 3 1 5 1 1 2 1 2 3 1	16
3 3 20 6 8 2 6 1 1 2 1 2 3 1 1 3 1	38
2 1 1 1 10 5 1 2 2	1