Problem A. Absurdistan Roads

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

The people of Absurdistan discovered how to build roads only last year. After the discovery, every city decided to build their own road connecting their city with another city. Each newly built road can be used in both directions.

Absurdistan is full of surprising coincidences. It took all N cities precisely one year to build their roads. And even more surprisingly, in the end it was possible to travel from every city to every other city using the newly built roads.

You bought a tourist guide which does not have a map of the country with the new roads. It only contains a huge table with the shortest distances between all pairs of cities using the newly built roads. You would like to know between which pairs of cities there are roads and how long they are, because you want to reconstruct the map of the N newly built roads from the table of shortest distances.

You get a table of shortest distances between all pairs of cities in Absurdistan using the N roads built last year. From this table, you must reconstruct the road network of Absurdistan. There might be multiple road networks with N roads with that same table of shortest distances, but you are happy with any one of those networks.

Input

For each test case:

- A line containing an integer N ($2 \leq N \leq 2000$) – the number of cities and roads.
- N lines with N numbers each. The j-th number of the i-th line is the shortest distance from city i to city j. All distances between two distinct cities will be positive and at most 1 000 000. The distance from i to i will always be 0 and the distance from i to j will be the same as the distance from j to i.

Output

For each test case:

- Print N lines with three integers 'a b c' denoting that there is a road between cities $1 \leq a \leq N$ and $1 \leq b \leq N$ of length $1 \leq c \leq 1\,000\,000$, where $a \neq b$. If there are multiple solutions, you can print any one and you can print the roads in any order. At least one solution is guaranteed to exist.

Print a blank line between every two test cases.

Examples

standard input	standard output
4 0 1 2 1 1 0 2 1 2 2 0 1 1 1 1 0 4 0 1 1 1 1 0 2 2 1 2 0 2 1 2 2 0 3 0 4 1 4 0 3 1 3 0	2 1 1 4 1 1 4 2 1 4 3 1 2 1 1 3 1 1 4 1 1 2 1 1 3 1 1 2 1 4 3 2 3

Problem B. Battle for Silver

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

Piet Hein was a Dutch naval officer during the Eighty Years' War between the United Provinces of The Netherlands and Spain. His most famous victory was the capture of the *Zilvervloot* ('Silver Fleet') near Cuba in 1628, where he intercepted a number of Spanish vessels that were carrying silver from the Spanish colonies in the Americas to Spain. Details about this famous naval battle are sketchy, so the description below may contain some historical inaccuracies.

The Silver Fleet consisted of vessels containing silver coins. Piet Hein's basic strategy was simple: tow away a number of vessels from the fleet, in order to capture their contents.

In an attempt to prevent the Dutch from carrying out this plan, the Spanish tied all the ships in their fleet together using huge iron chains. Each vessel in their fleet was fixed to at least one other vessel; any two vessels were connected by at most one chain; and the Spanish made sure that the chains did not cross each other, otherwise they could get tied up into a knot. As an end result, the vessels and the chains connecting them formed a connected, planar graph.

However, the Spanish preventive measures only made their situation worse. As an experienced naval officer, Piet Hein knew that towing away a group of ships was easiest if, for every two ships in the group, the ships were connected by a chain. He called such groups *chaingroups*.

Piet Hein ordered his men to tow away all the ships in the chaingroup that contained the largest amount of booty, after severing the links with the remaining ships in the Spanish fleet with a few highly accurate canon shots. The total booty in a chaingroup is the total number of silver coins in the vessels that make up the chaingroup.

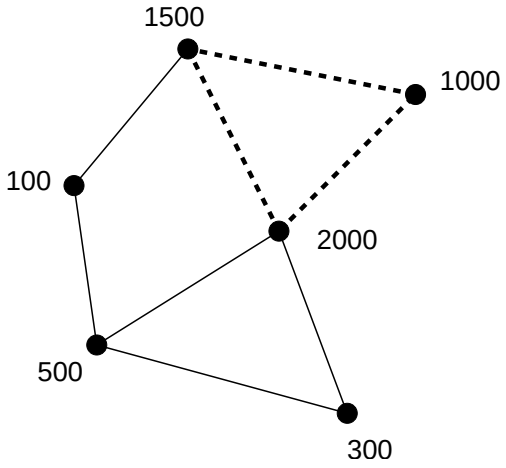


Figure 1: The Silver Fleet represented as a graph: each dot denotes a vessel in the fleet, while each line denotes a chain that connects two vessels. The vessels that are connected in the figure by the dashed lines correspond to the chaingroup that provides the highest total value of silver coins. In this case, Piet Hein loots 4500 silver coins from the fleet.

Given a description of the Silver Fleet, find the value of the chaingroup with the highest amount of booty (i.e., total number of silver coins in the ships that make up the chaingroup).

Input

For each test-case:

- A line containing two integers v ($2 \leq v \leq 450$) and e ($1 \leq e \leq 900$), the number of vessels in the fleet and the number of chains, respectively.
- Then, v lines specifying $S_1, S_2, \ldots, S_v$, the amount of silver coins carried by vessel i ($1 \leq i \leq v$). The S_i will be positive integers, where $100 \leq S_i \leq 6000$.
- Then, for each chain, a line containing two integers c_{start} and c_{end}, the two vessels connected by the chain, where ($1 \leq c_{start} < c_{end} \leq v$).

Each fleet forms a connected, planar graph.

Output

For each test case, one line containing a single positive integer: the number of silver coins that is captured by Piet Hein's fleet.

Examples

standard input	standard output
4 6 100 5000 1000 2000 1 2 1 3 1 4 2 3 2 4 3 4 6 8 1500 1000 100 2000 500 300 1 2 1 3 1 4 2 4 3 5 4 5 4 6 5 6	8100 4500

Problem C. Card Trick

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

I am learning magic tricks to impress my girlfriend Alice. My latest trick is a probabilistic one, i.e. it does work in most cases, but not in every case. To perform the trick, I first shuffle a set of many playing cards and put them all in one line with faces up on the table. Then Alice secretly selects one of the first ten cards (i.e. she chooses x_0, a secret number between 1 and 10 inclusive) and skips cards repeatedly as follows: after having selected a card at position x_i with a number $c(x_i)$ on its face, she will select the card at position $x_{i+1} = x_i + c(x_i)$. Jack (`J`), Queen (`Q`), and King (`K`) count as 10, Ace (`A`) counts as 11. You may assume that there are at least ten cards on the table.

Alice stops this procedure as soon as there is no card at position $x_i + c(x_i)$. I then perform the same procedure from a randomly selected starting position that may be different from the position selected by Alice. It turns out that often, I end up at the same position. Alice is very impressed by this trick.

However, I am more interested in the underlying math. Given my randomly selected starting position and the card faces of every selected card (including my final one), can you compute the probability that Alice chose a starting position ending up on the same final card? You may assume that her starting position is randomly chosen with uniform probability (between 1 and 10 inclusive). I forgot to note the cards that I skipped, so these cards are unknown. You may assume that the card face of every single of the unknown cards is independent of the other card faces and random with uniform probability out of the possible card faces (i.e. `2-10`, `J`, `Q`, `K`, and `A`).

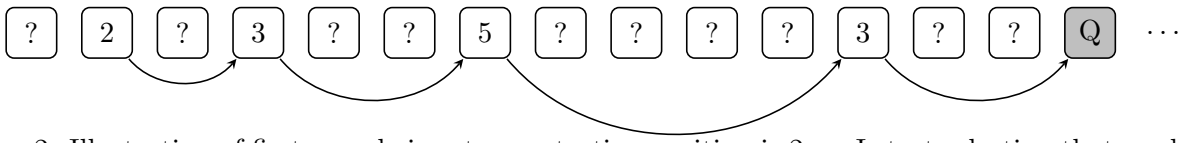


Figure 2: Illustration of first sample input: my starting position is 2, so I start selecting that card. Then I keep skipping cards depending on the card's face. This process iterates until there are not enough cards to skip (in this sample: `Q`). The final `Q` card is followed by 0 to 9 unknown cards, since `Q` counts as 10.

Input

For each test case:

- A line containing two integers n ($1 \le n \le 100$) and m ($1 \le m \le 10$) where n is the number of selected cards and m is the 1-based position of my first selected card.
- A line with n tokens that specify the n selected card faces (in order, including the final card). Each card face is given either as an integer x ($2 \le x \le 10$) or as a single character (`J`, `Q`, `K`, or `A` as specified above).

Output

For each test case, print one line containing the probability that Alice chooses a starting position that leads to the same final card. Your output should have an absolute error of at most 10^{-7}.

Examples

standard input	standard output
5 2 2 3 5 3 Q 1 1 A 1 2 A 1 10 A 6 1 2 2 2 2 2 2 7 1 2 2 2 2 2 2 2 3 10 10 J K	0.48713777570233253480715 73 0.10000000000000000000000000 0.10000000000000000000000000 0.17489233570253142396974 90 0.58307132103217674451174 68 0.62792296111157495562803 50 0.33465658276032720018919 74

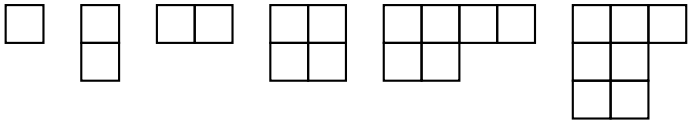
Problem D. Diagrams & Tableaux

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

A *Young diagram* is an arrangement of boxes in rows and columns conforming to the following rules:

- the boxes in each row and each column are contiguous,
- the left borders of all rows are aligned, and
- each row is not longer than the one above.

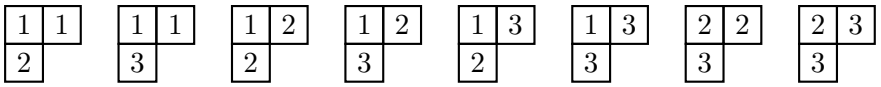
Here are some examples of Young diagrams:



A *semi-standard Young tableau* for a given number N is a Young diagram that has its boxes filled according to the following rules:

- Each box contains a single integer between 1 and N, inclusive,
- each integer is greater than or equal to the integer in the box to its left, and
- each integer is strictly greater than the integer in the box above.

Here is a list of all semi-standard Young tableaux for $N = 3$, based on a particular Young diagram:



Your task is to count how many semi-standard Young tableaux are possible, based on a given Young diagram, with a given N.

Input

Each test case consists of two lines. The first line of each test case specifies the Young diagram. This line starts with the number k satisfying $1 \le k \le 7$, the number of rows, followed by k positive integers $l_1, l_2, \ldots, l_k$. These integers specify the number of boxes on each row of the Young diagram, and they satisfy $7 \ge l_1 \ge l_2 \ge \cdots \ge l_k \ge 1$. The second line contains the integer N, satisfying $k \le N \le 7$.

Output

For each test case, print one line containing the number of semi-standard Young tableaux based on the given Young diagram, with the given N.

Examples

standard input	standard output
1 1 1 1 1 2 2 2 1 4 4 3 2 1 1 4	1 2 20 20

Problem E. Exponential Towers

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

The number 729 can be written as a power in several ways: 3^6, 9^3 and 27^2. It can be written as 729^1, of course, but that does not count as a power. We want to go some steps further. To do so, it is convenient to use `'^'` for exponentiation, so we define $a\hat{}b = a^b$. The number 256 then can be also written as `2^2^3`, or as `4^2^2`. Recall that `'^'` is right associative, so `2^2^3` means `2^(2^3)`.

We define a *tower of powers* of *height* k to be an expression of the form $a_1\hat{}a_2\hat{}a_3\hat{}\ldots\hat{}a_k$, with $k > 1$, and integers $a_i > 1$.

Given a tower of powers of height 3, representing some integer n, how many towers of powers of height *at least* 3 represent n?

Input

The input file contains several test cases, each on a separate line. Each test case has the form $a\hat{}b\hat{}c$, where a, b and c are integers, $1 < a, b, c \leq 9585$. The expression $a\hat{}b\hat{}c$ represents some number n.

Output

For each test case, print the number of ways the number $n = a\hat{}b\hat{}c$ can be represented as a tower of powers of height at least three.

The magic number 9585 is carefully chosen such that the output is always less than 2^{63}.

Examples

standard input	standard output
4^2^2 4^2^2 8^12^2 8192^8192^8192 2^900^576	2 10 1258112 342025379

Problem F. First Date

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

In 1582, pope Gregory XIII decreed a calendar reform to bring the mean length of the calendar year (counted in days) more in line with the actual number of days in an astronomical year. This calendar reform entailed a transition from the Julian calendar to the Gregorian calendar.

Both the Julian and Gregorian calendars have regular years with 365 days, and so-called *leap years* with 366 days. In regular years, the month February has 28 days, while in leap years, it has an extra 'leap day': February 29th.

The single difference between the Julian and Gregorian calendars is in their rule to determine if a year is a leap year. In the Julian calendar, leap years are those years that are divisible by 4. The Gregorian calendar's rule for determining leap years is a bit more complicated: years divisible by 4 are leap years, unless they are divisible by 100 but not by 400.

Following these rules, the years 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, and 2400 are all leap years in the Julian calendar. In the Gregorian calendar, the only leap years in this list are 1600, 2000, and 2400.

The old Julian calendar has a mean year length of 365.25 days, while the new Gregorian calendar has a mean year length of 365.2425 days. Given that the actual number of days in an astronomical year is about 365.24219 days, you can see why the Gregorian calendar is an improvement.

As part of the Gregorian reform, a number of dates were skipped, to reverse the effects of having used the Julian calendar for over 1500 years. Specifically, the Gregorian reform decreed that October 4, 1582 (Julian) was to be followed by October 15, 1582 (Gregorian).

However, by the end of the 16th century the Reformation was in full swing. While the Catholic countries tended to follow the Papal decree, many countries continued using the Julian calendar until much later. For example, the United Kingdom switched from the Julian calendar to the Gregorian calendar on September 2, 1752 (Julian) which was followed by September 14, 1752 (Gregorian) — by that time, 11 dates had to be skipped to make the switch. The last European country to switch was Greece, which made the transition as late as February 15, 1923 (Julian) which was followed by March 1, 1923 (Gregorian), skipping 13 dates.

Given the last day for which the Julian calendar is in effect for some country (expressed as a Julian date), determine the next day's Gregorian date, i.e., the first date that uses the Gregorian calendar.

Input

For each test case, the input consists of one line containing a date in the Julian calendar, formatted as `YYYY-MM-DD`. This date will be no earlier than October 4, 1582, and no later than October 18, 9999. The given date represents the last day that the Julian calendar is in effect for some country.

Output

For each test case, print the first Gregorian date after the calendar transition.

Examples

standard input	standard output
1582-10-04	1582-10-15
1752-09-02	1752-09-14
1900-02-25	1900-03-10
1923-02-15	1923-03-01

Problem G. Grachten

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

Damn! I did not only oversleep (and today is *the* contest day!) but I also got stuck somewhere in Delft on my way from the hotel to the contest site. Everywhere around me are grachten, these city-canals that are part of many cities in the Netherlands. I am in a bit of hurry, because the NWERC contest starts in a few minutes.

To make matters even worse, some bridges in Delft are closed due to a cycling race through the city. Thus, I decided to jump over some of the grachten instead of searching for open bridges.

Everyone knows that computer scientists like me are good at algorithms but not very good athletes. Besides, I am a bit faint-hearted and don't want to get wet. So I need your help to calculate the distance I have to jump over a gracht.

Luckily, I did attend the excursion in Delft city center yesterday, where I learned that all paving stones in Delft are squares and have the same size. This way, I can do some measurements on my side of the gracht (my units are paving stones):

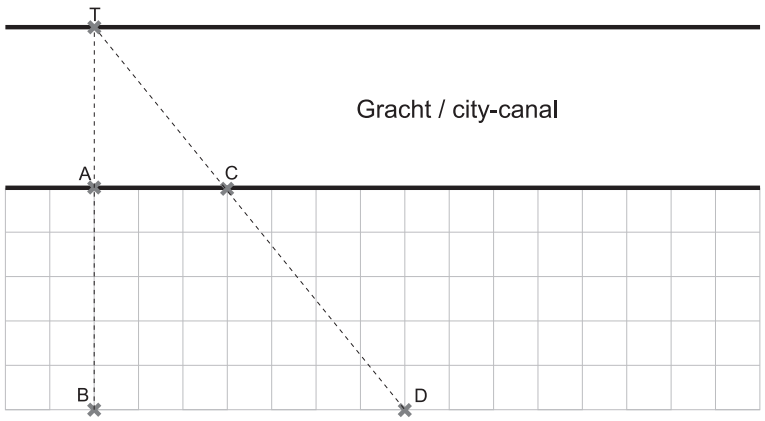


Figure 3: Illustration of first sample input.

I walked from point C to point D via points A and B while counting the paving stones.

Points A and C are always on the edge of the gracht. Points B and D have the same distance to the gracht. The target point T is always on the edge of the other side of the canal; it is the intersection point of the line through B and A, and the line through D and C. The angle between AT and AC is 90 degrees, and the two edges of the canal are parallel lines.

Please calculate the distance between A and T (necessary jump distance) for me.

Input

For each test case, the input consists of one line containing three positive integers that specify the distances between A and B, A and C, and B and D.

You may safely assume that no distance is larger than 1000 and the distance between B and D is larger than the distance between A and C.

Output

For each test case, print one line of output: the distance between A and T as a **reduced** fraction (i.e. remove all common factors of numerator and denominator).

Examples

standard input	standard output
5 3 7	15/4
5 3 8	3/1
1 2 3	2/1
23 42 47	966/5
500 500 1000	500/1
1 1 1000	1/999

Problem H. Highway of the Future

Input file:	`standard input`
Output file:	`standard output`
Time limit:	20 seconds
Memory limit:	256 Megabytes

It is the year 23413 and the quantum road authority (QRA) needs your help designing a new quantum highway. The biggest difference between a quantum highway and a regular highway is that quantum cars can switch lanes instantly, i.e. at t_1 a quantum car can be in one lane, while (as long as $t_1 \neq t_2$) at t_2 it can be in another.

Of course, in the year 23413 the future prediction authority (FPA) has information on exactly who will be using this new highway. For each quantum car that will travel along your quantum highway, the FPA supplies you with a value t, which is the time at which the quantum car will enter your highway, and a value v which represents the speed at which the quantum car will travel along your quantum highway.

The length of your highway will be 100 length units. In one time unit, a quantum car going at a speed of v speed units, will have travelled exactly v length units. The size of a quantum car is negligible compared to the length of your highway; it should be considered as a point.

Your job is to make sure that there will be no collisions on this quantum highway. Quantum cars are equipped with very sophisticated collision prevention mechanisms: as long as there are enough lanes in your quantum highway, cars will "magically" switch lanes to avoid collisions. A collision still occurs if, at any time, the number of cars at a particular position along the highway exceeds the number of lanes. Such collisions can happen even at the exact begin or end of the highway, as the sample cases illustrate.

What is the least number of lanes required to make sure there will be no collisions?

Input

For each test case:

- A line containing one integer n ($1 \leq n \leq 35\,000$): the number of quantum cars that will travel on your highway.
- n lines containing two integers:
  - t_i: the time at which quantum car i enters your quantum highway ($1 \leq t_i \leq 10\,000$).
  - v_i: the speed of quantum car i ($1 \leq v_i \leq 100$).

Output

For each test case, print one line containing one integer: the number of lanes required to make sure there will be no collisions.

Examples

standard input	standard output
3 15 20 19 100 10 10 3 10 20 10 10 10 30 2 10 10 10 10 2 10 1 20 100	3 3 2 2

Problem I. Infix to Prefix

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

Jaap wrote a solution to a lab assignment. The task was quite simple: convert an arithmetic expression in infix notation to an expression in Polish (prefix) notation. In infix notation operators are written between the operands (e.g. 12 + 5) while prefix notation places operators to the left of their operands (e.g. + 12 5).

This is the syntax of the expression Jaap had to convert:

Expression ::= *Number*
| '(' *Expression Op Expression* ')'
| '(' '-' *Expression* ')'

Op ::= '+' | '-'

Number ::= *Digit* | *Number Digit*

Digit ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

If a *Number* has more than one digit, it will not start with a '0'.
A *Number* has no more than 9 digits.

At this point we have to admit that the assignment was not very well specified. More specific, the syntax of the resulting expression was not given. So Jaap had to make some decisions himself — and he made the wrong decisions.

This was his first mistake: he believed that in prefix notation spaces are superfluous. This is true in infix notation, as there will always be an operator between two numbers. In prefix notation, however, numbers must be separated from one another. Omitting spaces in prefix notation, as Jaap did, gives rise to expressions like `+1234`, which has three different interpretations. (Exercise: draw the three different syntax trees.)

This was Jaap's second mistake: he believed that in prefix notation parentheses are superfluous. Prefix notation without parentheses is unambiguous only if the arity of the operators is fixed. Ambiguity occurs, for example, in the presence of both a unary and a binary minus. The expression: `--34`, can be read as `(- (- 3 4))`, evaluating to 1, or as `(- (- 3) 4)`, evaluating to -7, and even as `(- (- 34))`, evaluating to 34.

We do not ask you to reconstruct Jaap's program. We ask you to find out how ambiguous his output is.

Input

Each test case consists of a single line with a nonempty string of length ≤ 1000. The string contains only the characters '+' and '-' and digits '0' through '9'. This string is the output of Jaap's program.

Output

For each test case, print one line containing two numbers: the smallest and largest value that can be obtained by different interpretations of the input expression.

Examples

standard input	standard output
--34 +1234 --09 +111111111111 --0071	-7 34 46 235 -9 9 222222 111111222 -71 -71

Problem J. Jingle Balls

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5 seconds
Memory limit:	256 Megabytes

It will soon be time to decorate the Christmas tree. The NWERC judges are already debating the optimal way to put decorations in a tree. They agree that it is essential to distribute the decorations evenly over the branches of the tree.

This problem is limited to binary Christmas trees. Such trees consist of a trunk, which splits into two subtrees. Each subtree may itself split further into two smaller subtrees and so on. A subtree that does not split any further is a twig. A twig may be decorated by attaching at most one ball to it.

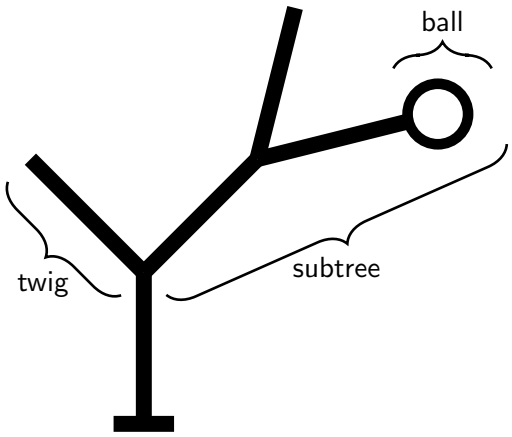


Figure 4: Example of a tree with subtrees, twigs and one ball.

A decorated tree has an even distribution of balls if and only if the following requirement is satisfied: At every point where a (sub)tree splits into two smaller subtrees t_1 and t_2, the total number of balls in the left subtree $\mathrm{N}(t_1)$ and the total number of balls in the right subtree $\mathrm{N}(t_2)$ must either be equal or differ by one. That is: $\mid \mathrm{N}(t_1) - \mathrm{N}(t_2) \mid \leq 1$.

In their enthusiasm, the judges initially attach balls to arbitrary twigs in the tree. When they can not find any more balls to put in the tree, they stand back and consider the result. In most cases, the distribution of the balls is not quite even. They decide to fix this by moving some of the balls to different twigs.

Given the structure of the tree and the initial locations of the balls, calculate the minimum number of balls that must be moved to achieve an even distribution as defined above.

Note that it is not allowed to add new balls to the tree or to permanently remove balls from the tree. The only way in which the tree may be changed is by moving balls to different twigs.

Input

For each test case, the input consists of one line describing a decorated tree.

The description of a tree consists of a recursive description of its subtrees. A (sub)tree is represented by a string in one of the following forms:

- The string '`()`' represents a twig without a ball.
- The string '`(B)`' represents a twig with a ball attached to it.
- The string '`(`$t_1 t_2$`)`' represents a (sub)tree that splits into the two smaller subtrees represented by t_1 and t_2, where t_1 and t_2 are strings in one of the forms listed here.

A tree contains at least 2 and at most 1000 twigs.

Output

For each test case, print one line of output.

If it is possible to distribute the balls evenly through the tree, print the minimum number of balls that must be moved to satisfy the requirement of even distribution.

If it is not possible to distribute the balls evenly, print the word '`impossible`'.

Examples

standard input	standard output
((B)()) ((((B)(B))((B)()))(B)) (()(((B)(B))(B)))	0 impossible 1

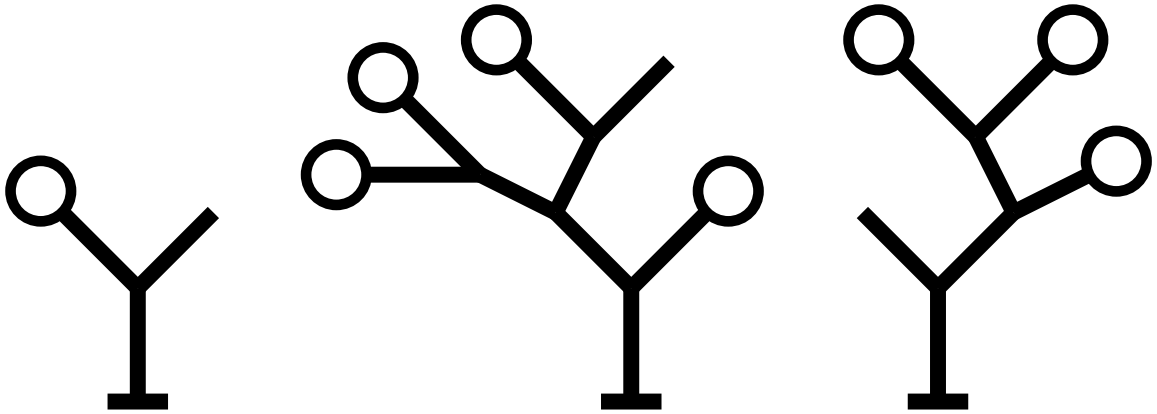


Figure 5: Trees corresponding to the example input cases.