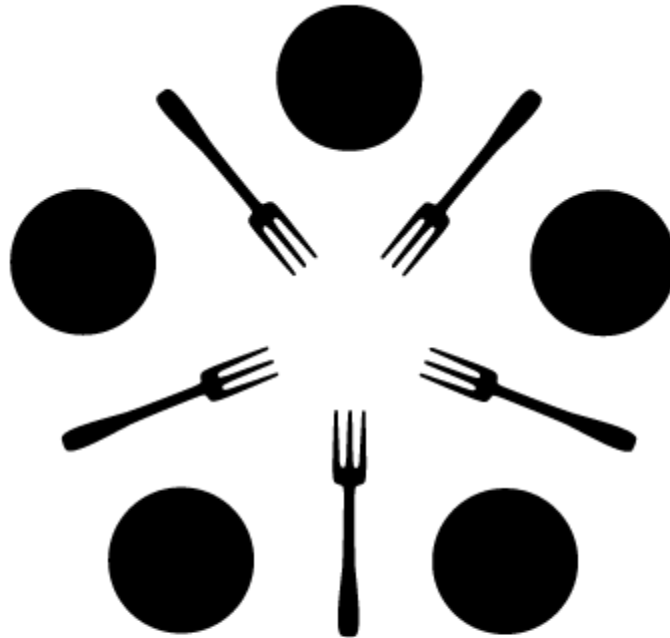


Contest Problems
Philadelphia Classic, Fall 2015
Hosted by the Dining Philosophers
University of Pennsylvania



dining philosophers
UNIVERSITY OF PENNSYLVANIA COMPUTER SCIENCE CLUB

Rules and Information

This document includes 14 problems. Novice teams do problems 1-9; standard teams do problems 5-14.

Any team which submits a correct solution for any of problems 1-4 will be assumed to be a novice team. **If you are not a novice team, please skip problems 1-4.**

Problems 1-4 are easier than problems 5-9, which are easier than problems 10-14. These problems are correspondingly labeled “Novice”, “Intermediate”, and “Advanced.” Order does not otherwise relate to difficulty, except that problem 14 is the hardest.

You may use the Internet only for submitting your solutions, reading Javadocs, and referring to any documents we link you to. You **may not** use the Internet for things like StackOverflow, Google, or outside communication.

As you may know, you can choose to solve any given problem in either **Java or Python**. If you would like to solve a problem in Java, we have provided a stub file that takes care of the parsing for you. If you would like to solve a problem in Python, you must create a file for the answer and parse the input file yourself. It may be useful to refer to the Java stubs for how this works. Python submissions should be named the same thing as the Java stub, but with a .py extension instead of .java (“EscapeVelocity.py” instead of “EscapeVelocity.java”, for example). If you use Python, you may refer to the Python standard library docs.

Do not modify any of the given methods or method headers in our stub files! They are all specified in the desired format. You may add class fields, helper methods, etc as you like, but modifying given parts will cause your code to fail in our testers.

There is no penalty for incorrect submissions. You will receive 1 point per problem solved. A team’s number of incorrect submissions will be used only as a tiebreaker.

Some problems use Java’s “long” type; if you are unfamiliar with them, they’re like an “int”, but with a (much) bigger upper bound, and you have to add “L” to the end of an explicit value assignment:

```
long myLong = 1000000000000L;
```

Otherwise, the “long” type functions just like the “int” type.

1. Novice Problem – Calculating Escape Velocity

Scientists wish to send a spaceship to Planet Z and then return back to Earth. Scientists have figured out the escape velocity needed for their spaceship to leave Earth, but they are having trouble finding the escape velocity needed to leave Planet Z because they do not yet know the mass and radius of Planet Z. The formula for the escape velocity is given by

$$v_{esc} = \sqrt{\frac{2GM}{R}}$$

where M is the mass of the planet in kilograms, R is the radius of the planet in meters, and v is the escape velocity in meters per second, assuming there is no atmospheric friction. **Note that Planet Z is in a separate universe, where G is equal to 1.**

You will be given two numbers, the first number being the mass of Planet Z in kilograms, and the second number being the radius of Planet Z in meters. Your job is to find the escape velocity of Planet Z.

Input Format

The input file has each test case on a single line. Each line consists of two floating point numbers that will fit inside a Java double.

Output Format

For each set of numbers inputted, output the escape velocity of the spaceship in meters per second, floored to the nearest integer (e.g. cast to an integer).

Sample Input	Sample Output
10.0 2.0	3
21.0 15.0	1

2. Novice Problem – Outer Space Car Chase

You've recently joined an outer space police force. For your first assignment you've been stationed along the Milky speedway, usually a boring place for newbies. However, as you're lounging around in your spacecar eating your donuts and drinking coffee, you suddenly get a call over the radio saying that an illegal vehicle is coming down the spaceway. It's your job to stop it! However, the only way you can tell which vehicle is the illegal one is by checking the license plate numbers on every spacecar you fly by to make sure they are legal numbers. You realize that doing this by hand will be extremely difficult, as you should never drive and derive. So, you must quickly write a program to check if the license plate numbers you're seeing are legal or not.

A legal license plate number will contain 10 digits and has the following property: It can be calculated such that the first digit on the left on the license plate is multiplied by 10, the second is multiplied by 9, continuing until the last digit is multiplied by 1. Then these products will be summed and, as long as the summation is divisible by 11, it is a legal license plate number.

You will be given a String of unknown length containing both numbers and dashes. This String will represent a license plate number you have seen. Don't forget, there are no limitations surrounding the placement or quantity of dashes in a license plate number. However, all license plate strings will contain exactly ten numbers. You will then return the either the correct last digit or a -1 if the digit in place is correct already.

Input Format

The input consists of multiple test cases. Each test case will consist of a license plate number string. These license plate numbers will have dashes between the numbers at different locations in each license plate number. All inputted license plate numbers will have 10 digits, separated by dashes in any position.

Example of a potential license plate number input: 37-9876-215-2

and formula to calculate check digit:

$3(10) + 7(9) + 9(8) + 8(7) + 7(6) + 6(5) + 2(4) + 1(3) + 5(2)$ without the last digit

then the last digit will be calculated by doing this summation and calculating the final number required to have the entire summation, including the last digit, be divisible by 11

Output Format

You should return an integer. If the license plate number is valid, print out -1, otherwise, print out the correct digit. Note that due to a special intergalactic numbering convention, the digit printed out may be "10", if that is what the required last digit must be to make the checksum divisible by 11.

Sample Input	Sample Output	Explanation
37-9876-215-2	5	The calculated last digit is not 2 to make the license plate valid, it is 5. Thus return 5

4-212-65-31-84	-1	The calculated digit last digit is 4 and thus this is valid. Return -1
----------------	----	--

3. Novice Problem – Rocket Tests

NASA is testing an experimental rocket and needs your help to analyze some data about its flight. You are given a list of the rocket's y-positions over time, as the rocket flies up and down. Inside the rocket's journey, the most important part to analyze is the longest increasing contiguous sequence of y-positions, so as to determine the longest period of time where the rocket was ascending. Help them to determine how long this sequence is!

You will be given a String of unknown length containing numbers. Each string position represents the y-position of the rocket at a time position. For example, given the string "345" we know the rocket had y-position 3 at time 1, 4 at time 2, and 5 at time 3. You will then return the length of the longest increasing sequence of y-positions in your String.

Input Format:

The input file will consist of multiple test cases. The input has each test case on a single line. Each line contains a string of unknown length of numbers, though each number will be guaranteed to be only a single digit.

Output Format:

For each test case, you must print the length of the longest increasing sequence of numbers on its own line.

Sample Input	Sample Output	Explanation
1234276234567	6	The longest increasing sequence in this case is "234567", which has length 6.
9746111	2	The longest increasing sequence is "46", which has length 2.

4. Novice Problem – Faulty Command System

The spaceship Andromeda has a faulty command system for navigating the universe. The command system first takes in a list of integers and finds all the local minima and local maxima. These local minima and local maxima are commands for the rocket. A local minimum is an element that is less than the two elements adjacent to it, and a local maximum is an element that is greater than the two elements adjacent to it. The first and last elements in the list are neither local minima nor maxima.

For each local minimum in the list, the spaceship moves backward the amount that was specified in the local minimum. For each local maximum in the list, the spaceship moves forward the amount that was specified in the local maximum.

After the spaceship has moved according to the local minima and maxima, the local minima and maxima are removed from the list. Then, the command system finds new local minima and maxima in the modified list and accordingly moves the spaceship in the same fashion. This process is repeated until the command system receives a list that contains no local minima and maxima.

Your program will be given a list of integers. Your program will keep track of the position of the spaceship relative to its original position after processing the commands.

Input format

The input file will consist of multiple test cases, each containing a string on its own line. This string contains a list of integers separated by commas and spaces.

Output format

For each test case, output the position of the spaceship relative to its starting position after the list has no more local minima and maxima.

Sample Input	Sample Output	Explanation
1, 3, 2, 5, 1, 1	6	Remove local mins and maxes of [1, 3, 2, 5, 1, 1] to get [1, 1, 1]. $(\text{Local maxes}) - (\text{Local mins}) = (3 + 5) - (2) = 6$
1, 5, 7, 7, 2, 3, 6	-5	Remove local mins and maxes of [1, 5, 7, 7, 2, 3, 6] to get [1, 5, 7, 7, 3, 6]. Remove local mins and maxes of [1, 5, 7, 7, 3, 6] to get [1, 5, 7, 7, 6]. $(\text{Local maxes}) - (\text{Local mins}) = (0) - (2 + 3) = -5$
1, 3, 6	0	There are no local mins and maxes in [1, 3, 6].

5. Overlap – Rocket Calculations

You are the head mathematician of an intergalactic spaceship. You are flying along, smooth sailing, when suddenly a black hole appears out of nowhere! The spaceship shakes and quakes and the power flickers on and off. Your captain orders you to get the spaceship out of the way of the black hole. However, to avoid the black hole, you need to do a lot of calculations. Unfortunately, the ship's main computing functions are down because of the power loss. You only have backup power, which evaluates expressions in a peculiar manner. The pieces of the computer are coming out of place, so you barely have any time to do the calculations to save the ship. Your crew members are counting on you to be able to program the computer and save all of your lives.

You must write a program for the computer to allow it to do the simple calculations you need in order to get the ship out of the path of the black hole. In this program you will only be passing in single digit numbers 0-9, as the backup power cannot handle double digit numbers. All calculations must be handled in a postfix fashion¹ (if you don't know what this, you are allowed to read up on it in the provided link), or the entire computer will crash and your ship will go tumbling into the black hole.

Input Format:

The input file will consist of multiple test cases containing a String of numbers and operators (+, -, *, /). All numbers in these String will be single digits 0-9. You will not need to handle multiple digit numbers. You are guaranteed that there will never be a division by zero.

Output Format:

Output the answer as a double, rounded to 1 decimal place. If you are using our Java stubs, you can just return the solution as a double in our function.

Input	Output
54+	9.0
123*+4-	3.0

¹ https://en.wikipedia.org/wiki/Reverse_Polish_notation#Postfix_algorithm

6. Overlap – Confusing Collinear Constellation

Since the ancient Greeks people have looked up at the stars and seen patterns in the randomness of their positions. When someone names these patterns it's called a constellation, and people have come up with quite a few of them over the years. NASA has decided to create a database of them but they need your help with a problem. Currently some constellations in their database contain groups of three stars that lie on a single line. If those points are supposed to be connected then one of the points isn't needed and if they are not supposed to be connected one of the points is confusing for a viewer since the eye is naturally drawn to collinear points^[citation needed]. NASA would like you to determine if any given constellation has three collinear points so that it can be re-evaluated.

You will be given an array containing $2N$ integers representing N points in the format "[X1,Y1,X2,Y2...]" and you should return a boolean that is true if 3 or more of the N points are collinear and false otherwise. NOTE: You may be tempted to loop over all possible sets of 3 points and simply test all of them, this will not work. There are N^3 such sets and we will give you cases where N is so large this will time out. You'll need to come up with something better.

Input format:

The input has each test case on a single line. Each line has all the points listed in the order "X1 Y1 X2 Y2..." where the variables are integers separated by spaces.

Output format:

The output is simple a boolean value corresponding to the answer and formatted according to Java's "System.out.println(true);" and "System.out.println(false);". Note the use of "println" since the response to each test should be on a separate line

Input Limit:

Array of integer values of length $2N$ where $1 \leq N \leq 5000$. All elements X of the array are in the range $-10000 \leq X \leq 10000$

Sample Input	Sample Output	Explanation
1 1 3 1 1 3 3 3	false	This is a square. No three points are collinear
1 1 3 1 1 3 3 3 4 4	true	The points (1, 1), (3, 3) and (4,4) are all on the same line

7. Overlap – Podracing

A long time ago, in a galaxy far, far away...

Anakin wanted to conduct an inter-planetary pod race in Galaxy Skyriver.

However, to maintain fairness in the tournament, Anakin decides to divide the planets into two groups, Ashla and Bogan, such that each group has the same amount of fuel. The podraces would then be contested between pods from Ashla and Bogan.

Unfortunately, Anakin is weak at mathematics and is unable to figure out if it is possible to divide the galaxy into two groups of planets such that each group has the same amount of fuel. Can you help him?

There are N planets in Galaxy Skyriver each with a different amount of pod fuel. The planets are numbered from 0 to $N-1$. Planet i has $A[i]$ liters of pod-fuel. Each test case comprises of an integer N that denotes the number of planets, and an array of integers that denote the fuel in each planet. For every test case, you should return true or false based on whether the planets can be divided up into two equal groups.

Input Format :

The first line of the input file contains an integer T that denotes the number of test cases. The input file puts each test case on two lines. The first line of each test case comprises the integer N , which denotes the number of planets. The second line of each test case comprises N space separated integers i.e. $A[0], A[1], A[2] \dots A[N]$, the fuel in liters, of planet numbered i .

Output Format :

The output is simply a boolean value corresponding to the answer and formatted according to Java's "System.out.println(true);" and "System.out.println(false);". Note the use of "println" since the response to each test case should be on a separate line.

Input Limit:

$1 \leq T \leq 1000$

$1 \leq N \leq 32$

$0 \leq A[i] \leq 1000000000$

Sample Input	Sample Output	Explanation
1 5 1 10 1 1 7	true	We can put planets numbered 0,2,3,4 in Ashla and planet 1 in Bogan. Both groups will have 10 units of fuel.
1 5 4 9 8 100 2	false	There is no possible way to optimally and fairly divide the planets.

8. Overlap Problem – Asteroid mining

As it is troublesome to transport minerals and other precious raw materials from the Earth out to other systems, scientists have begun to research mining these raw materials directly from asteroids out in space. Each mine has been designed like a binary tree - in other words, they consist of a number of chambers, starting with a main chamber at the top (the “root”). Each chamber can either be a “leaf” (i.e. it does not go any deeper), go deeper to a left subchamber, to a right subchamber, or both.

Due to a quirk in the mining technology, the scientists have determined that the optimal design for a mine is if it is symmetric. Informally, this means that if we were draw out the mine design and fold it into half, the two halves would match with each other. Help the scientists determine if the mine designs they have are symmetric or not.

You will be given the number of chambers **numChambers**, as well as two int arrays **leftChamber[]** and **rightChamber[]**, representing the indices of the subchambers that branch off to the left and right of each chamber. In other words, **leftChamber[i]** and **rightChamber[i]** will represent the left and right subchambers of the *i*-th chamber respectively. If there is no subchamber in that direction, the number in that spot will be -1. You should return a boolean that is true if the tree is symmetric and false otherwise.

Input format

The input file will consist of multiple test cases in order.

On the first line of each test case, there is one int **numChambers**, representing the number of chambers in the mine. This is passed as an int **numChambers** in the Java stub.

The remainder of the input consists of one line for each chamber. On the (***i*+1**)-th line, there are two integers **leftChamber[i]** and **rightChamber[i]**, representing the left and right subchamber of the *i*-th chamber. If there is no subchamber in that direction, the number in that spot will be -1. This is passed as two int arrays **leftChamber[]** and **rightChamber[]** in the Java stub.

Output format

For each test case, output a single line containing either Y if the mine is symmetric or N if the mine is not symmetric.

Input Limits

There will be at most 20 test cases.

$1 \leq \text{numChambers} \leq 100,000$

Sample Input	Sample Output	Explanation
<pre> 7 1 2 3 4 5 6 -1 -1 -1 -1 -1 -1 -1 -1 </pre>	<p>Y</p>	<p>This represents the following tree, which is symmetric:</p> <pre> 0 / \ 1 2 / \ / \ 3 4 5 6 </pre>
<pre> 5 1 2 3 -1 4 -1 -1 -1 -1 -1 </pre>	<p>N</p>	<p>This represents the following tree, which is not symmetric:</p> <pre> 0 / \ 1 2 / / 3 4 </pre> <p>Intuitively, if we fold it in half, the two halves don't match each other.</p>
<pre> 5 1 2 3 -1 -1 4 -1 -1 -1 -1 </pre>	<p>Y</p>	<p>This represents the following tree, which is symmetric:</p> <pre> 0 / \ 1 2 / \ 3 4 </pre>

9. Overlap Problem – Space Resources

It's year 3000 and the reckless humans of Earth have exhausted the planet's supply of precious metals. Unless they get their hands on these resources, humans will lose the ability to manufacture all electronics. Luckily, starship captain Zapp Brannigan happens to have just completed a research expedition on the resource-rich planet Zorb103492341. Zapp had the foresight to collect a **massive** list of all of the resources available on Zorb103492341. The humans on Earth fax Zapp a list of the precious resources they need on Earth.

Zapp needs to program his humanoid robot to find the duplicate resources in these two lists so Zapp can choose which resources to bring back to Earth. It turns out that Zapp is a horrible programmer and begs you to help.

Your program will be given two sorted lists of strings. The first list (named "requested") contains the names of all the resources needed on Earth. The second list (named "available") contains the name of resources available on Zorb103492341. You should return a list containing the names of all the resources which are in both lists. Your returned list should be sorted.

Note: the available list is significantly larger than the requested list. Thus, checking the available list one by one for each resource is likely not to run fast enough. All resource names are lowercase.

Input Format

Read the input file until it ends. The file contains multiple test cases. Each test case begins with two integers **a** and **b**. Following are **a** lines of strings belonging to the "requested" list and **b** lines of strings belonging to the "available" list. After reading all **a** and **b** lines, there may be additional test cases.

Output Format

For each test case print the duplicates list as a series of strings separated by spaces. For example, print the list [beryllium, gallium, rhodium] as

beryllium gallium rhodium

Input Limits

There will be at most 10 test cases.

$1 \leq |\text{requested}| \leq 5000$

$1 \leq |\text{available}| \leq 500000$

Sample Input	Sample Output	Explanation
<p>requested: [rhodium]</p> <p>available: [beryllium, rhodium]</p>	rhodium	Only rhodium was requested.
<p>requested: [beryllium, gallium, rhodium]</p> <p>available: [beryllium, bismuth, germanium, gold, indium, mercury, osmium, palladium, platinum, rhenium, rhodium, ruthenium, silver, tellurium]</p>	beryllium rhodium	<p>Only beryllium, gallium, and rhodium was requested. Also, gallium was requested but was not available.</p>

10. Standard Problem – Star Temperatures

You are slowly making a catalogue of all the stars in many galaxies, far far away. Your friend Lewis wants to run some statistical analyses on the temperatures of the stars, so every time you catalogue a new star, he pesters you for the median temperature of all the stars you have seen so far. If you have seen an odd number of stars, there will only be one median. If you have seen an even number of stars, Lewis wants to know the smaller of the two middle numbers (instead of the mean of the two). Help Lewis get the information he needs!

Your program will be given the number of stars in the galaxy, **numStars**, as well as an array **temperatures[]** containing the temperature of each star. You will have to return an array containing the medians of each prefix of the sequence, just as Lewis requires.

Input format

The input file will consist of multiple test cases in order.

On the first line of each test case, there is one int **numStars**, representing the number of stars in the galaxy.

On the second line of each test case, there are **numStars** numbers, each representing the temperature of a single star. These are given in the order that you need to process them, and is passed as an int array **temperatures[]** in your input.

Output format

For each test case, output **numStars** lines of output. The **n**-th line of output should contain the median(s) of the first **n** stars of that test case. For even numbered lines, output the smaller of the two numbers first, followed by the larger.

Input Limits

There will be at most 15 test cases.

$1 \leq \text{numStars} \leq 50,000$

Sample Input	Sample Output	Explanation
5 1 2 3 4 5	1 1 2 2 3	The median of [1] is 1. The medians of [1, 2] are 1 and 2, so we print 1. The median of [1, 2, 3] is 2. The medians of [1, 2, 3, 4] are 2 and 3, so we print 2. The median of [1, 2, 3, 4, 5] is 3.
6 3 5 1 2 6 4	3 3 3 2 3 3	The median of [3] is 3. The medians of [3, 5] are 3 and 5. The median of [1, 3, 5] is 3. The medians of [1, 2, 3, 5] are 2 and 3. The median of [1, 2, 3, 5, 6] is 3. The medians of [1, 2, 3, 4, 5, 6] are 3 and 4.
10 4 5 6 7 8 9 10 1 2 3	4 4 5 5 6 6 7 6 6 5	

11. Standard Problem – Safe Spacewalks

While modern spaceships are technologically impressive, the hardware which comprises them is quite error prone (those darned hardware engineers!). Occasionally, space travelers like yourself need to complete a spacewalk to repair components on the outside of the ship. As you already know, jetpacks are critical to maneuvering in zero gravity, such as during spacewalks. Unfortunately, you bought the P1000, a bargain jetpack (read: cheap and crappy) with an annoying property. Mainly, the jetpack can only keep you stabilized if you and your tools collectively weigh exactly 300 kilograms. We don't quite understand why mass should matter in a zero-gravity environment, but nevertheless it has been shown that P1000 users who do not satisfy this specification will surely drift off into space, probably forever.

Luckily for you, you have a lot of different space tools lying around which you can take with you on different space walks. Let's call a quadruple of tools valid if the mass of the tools combined with your mass is exactly 300 kilograms. A valid quadruple of tools will ensure that the jetpack is stabilized. Since you know your own mass, you know the value of "sum", which is the total mass of a valid quadruple of tools. You are curious how many quadruples of tools are valid (in other words how many quadruples add up to "sum"). Since you are a star programmer, you decide to write a program to answer your question.

Your program will be given an integer "sum" which is the total mass of a valid quadruple of tools. Your program will also be given a list of **distinct positive integers** called "tools". You should return the number of unique quadruples of distinct integers which add up to sum.

Note:

- (1, 2, 3, 4) and (4, 3, 2, 1) are not unique and should only be counted once.
- (1, 2, 3, 3) is not a valid quadruple because it does not consist of four distinct integers.

Input Format

Read the input file until it ends. The file contains multiple test cases. Each test case begins with an integer **sum** corresponding to the total mass of a valid quadruple of tools. On the next line is an integer **k**. Following are **k** lines of integers belonging to the "tools" list. After reading all **k** lines, there may be additional test cases.

Output Format

For each test case print the result on its own line.

Input Limits

There will be at most 20 test cases.

$1 \leq \text{sum} \leq 100000$

$1 \leq |\text{tools}| \leq 750$

Sample Input	Sample Output	Explanation
sum: 10 tools: [4, 2, 1, 3, 5]	1	There is 1 quadruple which adds up to 10: (1, 2, 3, 4)
sum: 20 tools: [14, 10, 2, 1, 3, 4, 5]	3	There are 3 quadruples which add up to 20: (1, 2, 3, 14), (1, 4, 5, 10), and (2, 3, 5, 10)

12. Standard Problem – Titillating Time Travel

NASA has discovered a way of gravity slingshotting past a black hole so effectively that the laws of causality are violated and the vessel arrives at its destination *before* it arrives. This process requires moving past the black hole so they can't use it to simply travel infinitely far back in time. At least not without some help from you. NASA would like to figure out if they can plan a route through space waypoints that allows the traveler to end up back in their original location at a time earlier than they started. Specifically they would like to know if this can be accomplished in a space sector with only one black hole. Space travel can only happen between pre-defined way points because of the Space Travel Problem Simplification (STPS) act of 3015 so they will give you a list of all the nodes and how long it takes to travel between them and you need to determine if a time loop exists using those points. You should note that due to the complexities of space travel not all way points are connected to each other. You should also note that because of even more complexities of space travel (space traffic) the triangle inequality does not hold (i.e if you can travel directly between two nodes that still might not be the shortest path between those points). Finally the time from waypoint a to waypoint b might not be the time from waypoint b to waypoint a since the time travels are affected by the particular space routes which are determined directionally.

You will be given as input a two dimensional array of integers called 'times' where 'times[i][j]' is the travel time from waypoint 'i' to 'j' or 0 if you can't travel between those nodes. Exactly one element of 'times' will be less than 0.

Input Format

The input consists of a single line for each test case. The line consists of a list of integers separated by spaces representing the elements of the times array listed in row-major format (first the first row is listed, then the second and so on).

Output Format

The output is simple a boolean value corresponding to the answer and formatted according to Java's "System.out.println(true);" and "System.out.println(false);". Note the use of "println" since the response to each test should be on a separate line.

Input Limit:

If the number of elements of 'times' is N^2 then $2 \leq N \leq 10000$

Sample Input	Sample Output	Explanation
<pre>0 1 5 1 0 -2 1 0 0</pre>	false	<p>Note: in the input file, this input would be on one line, this has been expanded for reading convenience.</p> <p>The best a ship could do is travel 0->1->2->0 which leads to a total travel time of 0 which isn't traveling back in time</p>
<pre>0 1 2 4 1 0 -5 1 2 0 0 0 3 5 0 0</pre>	true	<p>You could travel from 1->2->0->1 which results in a total travel time of -3</p>

13. Standard Problem – First Contact?

Your sensors suddenly pick up a mysterious signal from a distant galaxy! Excitedly, you begin decoding the signal and it seems to be... some kind of math problem?

The signal is asking for a certain property of Pascal's triangle. For those of you who are unfamiliar with Pascal's triangle, it is an (infinite) triangle of integers where each number is constructed by summing the two numbers above it. For example, the first 5 rows of the triangle are as follows:

```
  1
 1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Note that each number is the sum of the two numbers above it (e.g. the 6 in the bottom row is the sum of the two 3s above it, and each 4 in the bottom row is the sum of the 3 and 1 above it).

We wish to find out the following: given some prime P , how many integers in the first N rows of Pascal's triangle are divisible by P ? Help figure out this problem and decipher the mysteries of the universe!

Input format

The input file will consist of multiple test cases in order.

Each test case will consist of three longs **numRows**, **divisor**, and **modulus**. **numRows** will represent the number of rows of Pascal's triangle you need to consider. **divisor** will be the number you need to check for divisibility by. **modulus** will be the number you need to mod the answer by. It is furthermore guaranteed that **divisor** will be a prime number.

Output format

For each test case, output a single long, representing how many numbers there are in the first **numRows** of Pascal's triangle that are divisible by **divisor**. This number may be extremely large, so you will have to output the answer mod **modulus**.

Input Limits

There will be at most 30 test cases.

$1 \leq \text{numRows} \leq 1,000,000,000$

$3 \leq \text{divisor} \leq 1,000,000$

$1 \leq \text{modulus} \leq 1,000,000,000$

Sample Input	Sample Output	Explanation
5 3 100	3	There are 3 integers in the first 5 rows of Pascal's triangle divisible by 3.
5 3 2	1	
1000000 7 1000007	675154	

14. Standard Problem – Commander’s Conundrum

Having determined that the challenging math problem likely came from an alien race, you, as the commander of an intergalactic spaceship, face a difficult decision. With supplies running low, you must decide whether to pursue the strange symbol from the distant galaxy or return back to the Milky Way.

Being a democratic commander, you decide that you will go with the majority decision. However, your ship has a rather odd organizational structure. You have 3 subordinates, one, two, and three, and they are the only three people whose opinion you can ask. However, your subordinates decide their view based upon a majority of their subordinates, and each of your subordinates can only ask their three subordinates. In fact, everyone’s opinion in this crew is driven by the majority of their subordinates; the only people who hold an individual opinion are the lowest members of the crew (i.e. those with no subordinates)!

However, there is a twist: If every member of your crew is asked their opinion, those with no subordinates will realize they are the only one’s with an opinion, and they will rise up and mutiny the ship. Thus, your challenge is this: you must decide what the majority opinion is on the ship, but you cannot ask every member.

This is a risky proposition, so you may not succeed, but your crew is large, so you can be guaranteed to have a strong chance of succeeding. Note, however, you must be able to have a strong chance of succeeding on any input.

We will give you a special class called `Astronaut`, which implements the organizational structure described above. In other words, the class consists of an object with three subordinates of the same type, labeled one, two, and three, along with boolean values for whether the `Astronaut` has no subordinates, what their vote is if they have no subordinates, and whether their vote has been asked.

We have already written the method which parses text input and generates the `Astronaut` object representing you, the commander. You will return a boolean indicating the decision made by the commander.

Regardless of your language of choice, you must use the `Astronaut` class and cannot modify it! Our grading scripts depend on it! Please ask if you have any questions on it.

Input Format

For those using Python, you must parse our text input, which consists of a single line representing the true false opinions of the lowest members of the crew. Let us know if you need any assistance with this.

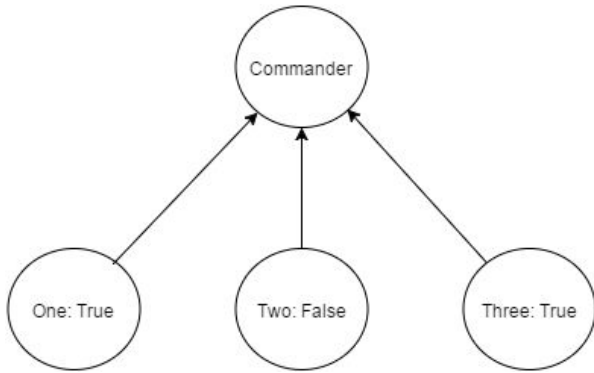
Output Format

You will output a single boolean value, `true`, meaning you will pursue the distant galaxy, or `false`, meaning you will return to the Milky Way. We will also print an additional line, the “`allAccessed`” method, which will help us check whether you asked every member of the crew. We will also then print 5 dashes, which will help you delimit test cases to check your solution.

Those using Python will need to add these latter two outputs to their main method. Finally, please print everything in Java format.

Input Limits

You can be guaranteed that the number of members of your crew is a power of three, guaranteeing that each crewmember either has three subordinates or none at all. You can also be guaranteed that your crew will be large, which should help you succeed on any input.

Sample Input	Sample Output	Explanation
 <pre>graph BT; One((One: True)) --> Commander((Commander)); Two((Two: False)) --> Commander; Three((Three: True)) --> Commander;</pre>	true	<p>The commander has three subordinates, none of whom have any subordinates, so the majority is simply the majority of those three, which is true.</p> <p>In addition, there must be some chance your solution only asked two of these three astronauts their opinion</p>