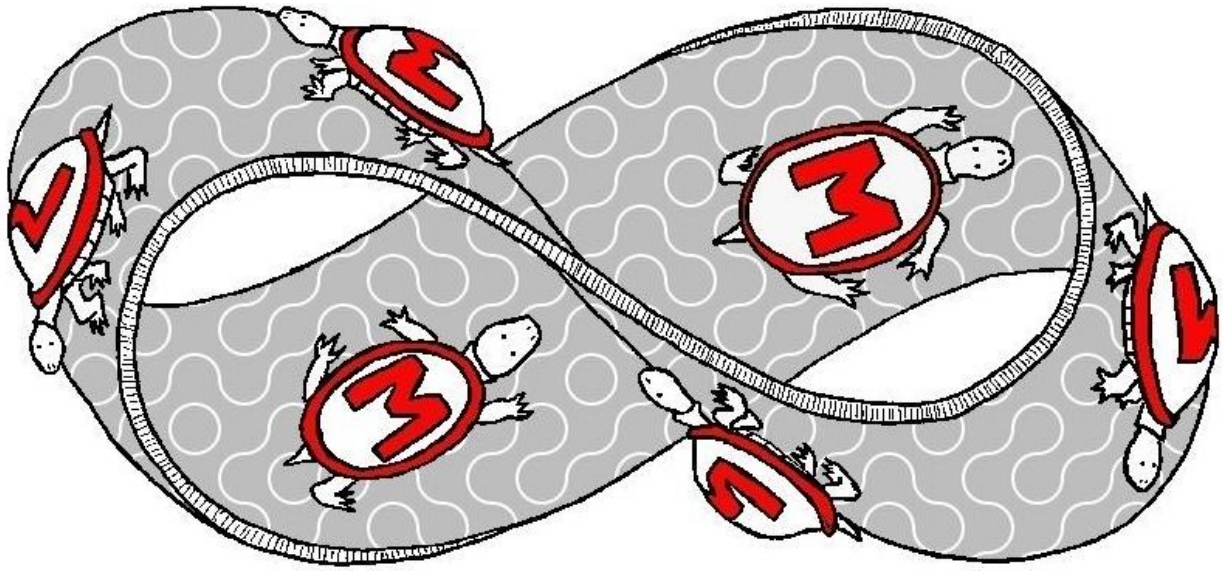
2018 University of Maryland High School Programming Contest



1. Mixing Dino Genes

Jurassic World has a new head dino-biochemist, and she is revolutionizing gene splicing via a novel, patent-pending method. Specifically, if she wants to mix the DNA of two dinosaurs, she proceeds as follows.

- Prepare a petri dish of A milliliters of liquid DNA from Dinosaur #1.
- Prepare a petri dish of B milliliters of liquid DNA from Dinosaur #2.
- Mix'em up!
    - Take C milliliters of liquid DNA from dish #1 and places it into dish #2.
    - Take C milliliters of (the now perfectly mixed!) liquid DNA from dish #2 and place it back into dish #1.
- Keep mixing …

She repeats this mixing process iteratively for a very long time. If she repeated this process forever, then the ratio of liquid DNA from Dinosaur #1 to liquid DNA from Dinosaur #2 in each dish would converge to A/B. However, she desperately wishes to work a 9am-5pm job, and so is interested in **how many iterations of mixing** it takes before the mixture is within some tolerance of A/B. (Once the mixture is "close enough" to A/B, she can immediately 3D print a new dinosaur!)

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case starts with the number **A** of milliliters in dish #1, the number **B** of milliliters in dish #2, the mixture amount **C**, and a tolerance **T**. You can assume **A**, **B**, and **C** are positive integers with **C < min(A, B)**, both petri dishes have capacity at least **A**+**B** and **T** is a positive real number.

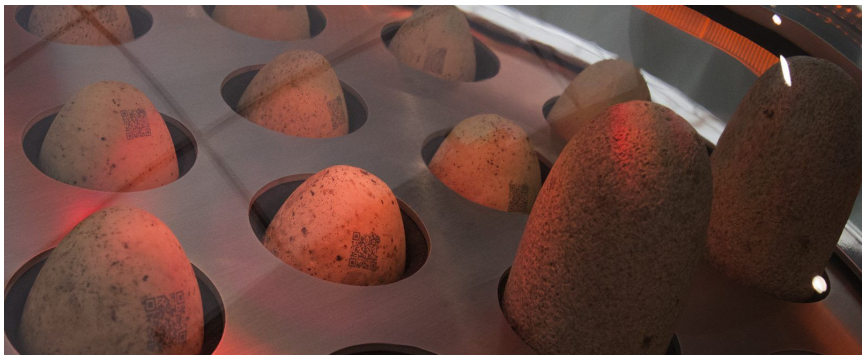
Output:
The output is a list of **I**, the number of iterations before the mixture of liquid DNA in both petri dishes is within tolerance **T** of the value **A/B**.

Examples:

Input:	**Output:**
3 32 16 1 0.00001 32 16 4 0.00001 1 1 1 0.001	 145 38 1

2. Flipping Eggs

Dinosaur eggs are notoriously finicky, and must be kept at a relatively constant, warm temperature before hatching. The Jurassic World staff have arranged all of their **M** eggs in a circle underneath a large heat lamp, and have installed a robotic egg-flipping arm in the middle of that circle. Every morning, they schedule the arm so that every **K**th egg *that has not been flipped yet that day* is flipped. Furthermore, every day, **L** of the eggs at the end of the flipping cycle must *not* be flipped. The Jurassic World staff is interested in determining, given **M** total eggs and an integer **K** representing the step size of the egg-flipping robotic arm, the indices of the **L** final unflipped eggs---in the order that they would've been flipped otherwise.



For example, if there are **M**=8 eggs, and **K**=3, then the final **L**=2 eggs that will *not* be flipped are the 7th and 4th eggs, in that order.

You can assume that the arm always starts counting at the 1st egg (that is, if **K**=3, then the first egg to be flipped is the 3rd egg), and that all integer inputs are positive. The Jurassic World scientists start counting eggs at 1---that is, if **M**=5, then the eggs are number 1, 2, 3, 4, and 5.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case starts with the number **M** of eggs, the step rate **K** of the egg-flipping arm, and the **L** final un-flipped egg indices to report. You can assume **M** is positive, **K** is not greater than **M**, and **L** is not greater than **M**.

Output:

The output is a list of **L** integer indices, separated by a space, representing the positions of the final **L** eggs that are left un-flipped by the robotic arm.

Examples:

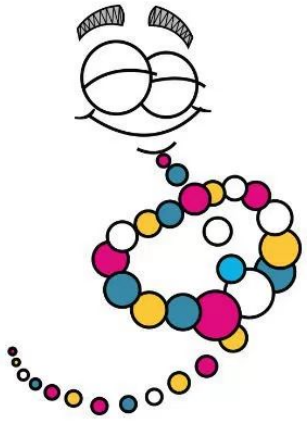
Input:	**Output:**
3 8 3 8 8 3 2 41 3 2	 4 7 2 6 3 1 5 8 7 4 31 16

3. Chain Reaction

Mr. DNA, the animated DNA helix and chief technology evangelist of Jurassic World, is explaining to park patrons how tricky mixing chemicals together can be. He explains that each chain of mixings of a particular amount of chemical results in a multiplicative increase, or decrease, in the amount of the final chemical compound created. Furthermore, because each individual component of the chemical reaction is mixed by hand, the park only deals with (positive) rational amounts of the component. For example, if the park mixes 2/3ml of Chemical A with 3/4ml of Chemical B and 7/2ml of Chemical C, then

2/3 x 3/4 x 7/2 = 7/4

milliliters of the final compound is created. The park is interested in knowing beforehand exactly how much of each compound will be created---reduced to lowest terms, so that humans might understand it.



More generally, the park is interested in computing the product of a set of positive rational numbers, without running into overflow and underflow problems---which are bound to happen when very small or very large measurements are used! For example, take:

1/2 x 1/2 x … x 1/2 x 1/2 (repeated 10000 times) x 2/1 x 2/1 x … x 2/1 x 2/1 (repeated 10000 times)

The product of the above 2000 rationals is simply 1, but a naive algorithm to compute this product might run into both overflow and underflow issues. Your goal is to help Mr DNA multiply rationals correctly, and to reduce them to lowest terms.

Important: you **may not** use the `BigDecimal`, `BigInteger`, and similar packages from `java.math`.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case starts with the number **n** of rationals to be multiplied. This is followed by the **num/denom** definitions of each of the **n** rationals; each numerator is followed by a slash, and each rational is separated by whitespace (e.g., the line “2 1/2 3/4" lists 2 rationals, 1/2 followed by ¾). You may assume that all the numerators and denominators are positive integers.

Output:

The output is a rational in lowest terms consisting of a numerator, followed by a slash, followed by a denominator. Again, the numerator and denominator should both be integers. If the rational is a whole number, represent it as a numerator divided by 1 (e.g., if your answer is “2”, print “2/1”).

Examples:

Input:	**Output:**
3 2 1/3 1/1000 4 1/1000 1/1000 1000/1 1000/1 3 3/5 10000/2 4/25000	 1/3000 1/1 12/25

4. Primetime Investigation

Simon Masrani, CEO of the Masrani Corporation and current owner of Jurassic World, is under investigation by the Securities and Exchange Committee (SEC). To satisfy his investigators, he must reveal -- at a moment's notice -- only "prime" evidence that his company's finances and corporate governance are in order. As a pedantic and law-abiding executive, Masrani has kept his company's documentation in ascending chronological order since the founding of the firm, with the "prime" documentation occurring whenever the current count of documents is a prime number. That is, the first document is not "prime," the second document is "prime," the third is also "prime," the fourth isn't, the fifth is, and so on.

Thus, when the SEC investigator asks for the Nth oldest piece of prime information, Masrani would like to be able to immediately index into his database of documentation to retrieve the information.

Formally, given a positive integer N, return the Nth prime. For example, N=1 should return the number 2, N=2 should return the number 3, N=3 should return the number 5, and so on. You may assume that $N < 2^{16}$.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case is represented by exactly one positive integer **n**.

Output:

The output is the **n**th prime. For example, if **n**=5, then the output is exactly "11".

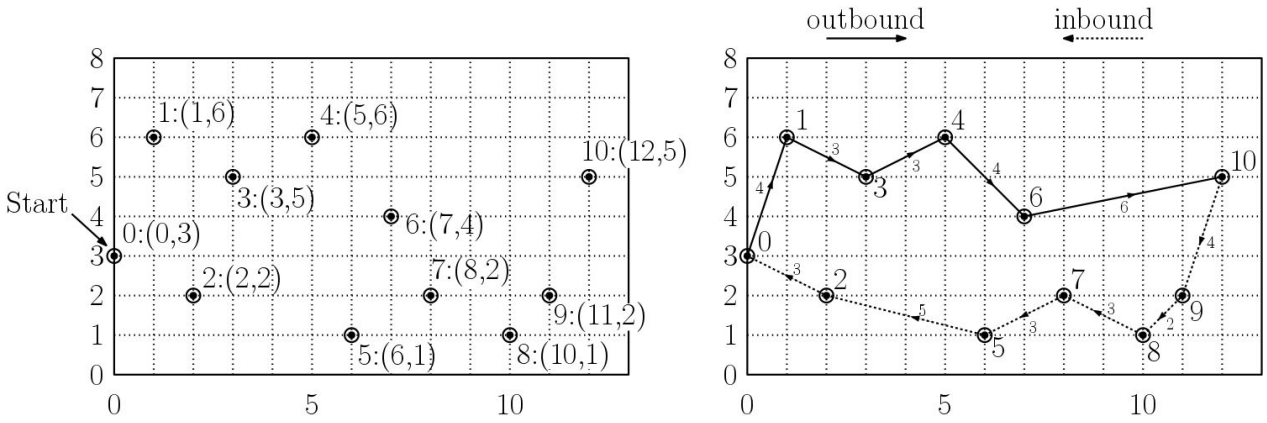
Examples:

Input:	**Output:**
3 1 2 12	 2 3 37

5. In-and-Out Tour

Brothers Zach and Gray are out in their gyrosphere exploring Jurassic World. They discover a map that shows the locations of a number of dinosaur eggs. From the starting point of the gyrosphere ride, they wish to visit all the eggs as quickly as possible. There are $\mathbf{n \geq 1}$ eggs, located at points $\mathbf{p_1}$, …, $\mathbf{p_n}$, where each point $\mathbf{p_i}$ has the coordinates $\mathbf{(x_i, y_i)}$. Their tour starts and ends at a point $\mathbf{p_0 = (x_0, y_0)}$. The start point $\mathbf{p_0}$ lies to the west of all the other points, that is, it has the smallest x-coordinate (see the figure below).

The park's gyrospheres have a built-in restriction. Each journey consists of two portions, an *outbound portion* where the x-coordinates increase strictly, and an *inbound portion* where the x-coordinates decrease strictly. Your job is to help Zach and Gray compute the shortest tour that visits all the eggs, subject to this *in-and-out restriction*.



To simplify computations, we define the distance between two points to be the sum of the absolute differences in their x- and y-coordinates, also known as the Manhattan or taxicab distance

$$\mathbf{dist(p_i, p_j) = |x_i - x_j| + |y_i - y_j|}.$$

The path segments in the above figure, right, are labeled with these distances.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case starts with the number **n** of egg locations. This is followed by the **(x,y)** coordinates of **n+1** points, first the start point and then the **n** egg locations. All the coordinates are integers (possibly negative). You may assume that they are given in *strictly increasing* order of x-coordinate (**x[i] < x[i+1]**). We have provided a skeleton program that reads the data, storing the input coordinates in two arrays **x[0..n]** and **y[0..n]** where **(x[0],y[0])** is the start point.

Output:

For each test case, your program will compute a minimum-length tour that starts and ends at $\mathbf{p_0}$, and visits all the egg locations exactly once, subject to the in-out restriction. Specifically, the program determines the indices of the points along the tour, starting and ending with 0, as well as the total length of the tour.

Because distances are symmetrical, all solutions have two equivalent versions by swapping the inbound and outbound portions. To unify the output, we make the convention that point $\mathbf{p_1}$ must be visited on the outbound portion of the journey, thus the output tour has the form 0, 1, …, 0

Note:

We have provided a skeleton program that reads the input and prints the output. All you need to provide is a function `GetInOutTour()` with the following signature:

```
private static void GetInOutTour(int[] x, int[] y, boolean[] outBound)
```

This function is given the (x,y) coordinates of all the points (in two arrays, each of length **n+1**), and stores the result in an (**n+1**)-element array `outBound`, where for **1≤ i ≤ n**, `outBound[i]` is true if point $\mathbf{p_i}$ is visited on the outbound portion of the tour and false if on the inbound portion. (The values of `outBound[0]` and `outBound[n]` can be set arbitrarily.)

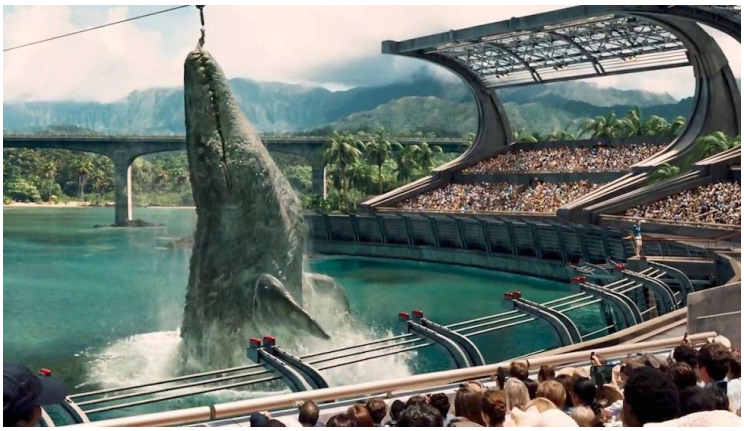
Example:

Input:	**Output:**

1
10
0 3
1 6
2 2
3 5
5 6
6 1
7 4
8 2
10 1
11 2
12 5
	Trial: 1
	  Points:
	  [0]: (0,3)
	  [1]: (1,6)
	  [2]: (2,2)
	  [3]: (3,5)
	  [4]: (5,6)
	  [5]: (6,1)
	  [6]: (7,4)
	  [7]: (8,2)
	  [8]: (10,1)
	  [9]: (11,2)
	  [10]: (12,5)
	  Tour: 0 1 3 4 6 10 9 8 7 5 2 0
	  Total distance = 40

6. Zipline to Savings

There is a new island-themed area of the park that has been built in the middle of the ocean. There are hundreds of fairly tiny artificial islands. They have run ziplines (so unidirectional) between various pairs of the islands. They have also decided to spice things up a bit and decided to release a large number of Hainosaurus in the waters around and between the islands.



As an interesting savings deal for your vacation package, you are allowed to give a list of ***n*** islands from which they will pick on onto which they will drop you and a destination that if you reach, you'll get your discount. Of course, the only way to have a chance of living to do that is by going island to island via zipline, starting where they drop you and ending at the other of the islands they tell you to reach.

Their savings policy is a bit unusual. If you succeed at getting from your start point to your destination point, they will discount your vacation price by ***n***-hundred (where ***n*** is the number of islands you said they could pick from). So, if you choose 5 of the islands and complete your mission, you get a $500 rebate, but if you choose 50 of the islands and complete your mission, you get a $5,000 rebate.

You don't have too long to give your list after they show you the map of ziplines, so you'll need an efficient algorithm that given a proposed set of ziplines will find all of the island clusters that guarantee a discount, and of those pick the largest possible such cluster.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case starts with the number **n** of islands, followed by the number **z** of ziplines. This is followed by the **(start,end)** island pairs between which each zipline runs.

Output:

For each test case, your program will compute the largest grouping of islands where you can get from any island in the group to any other island in the group using the ziplines.

Note:

We have provided a skeleton program that reads the input and prints the output. All you need to do is implement the body of `largestIslandChain(Scanner)` to return the correct value. Note that the skeleton has code to read all of the data, but does not have code to store or utilize it.

Example:

Input:	**Output:**

2
 	
8
11
0 1
1 6
2 3
2 4
3 1
4 7
5 0
5 1
6 5
7 2
7 3
 	
8
13
0 1
1 2
1 4
1 5
2 3
2 6
3 2
3 7
4 0
4 5
5 6
6 5
6 7
	`Biggest group size: 4`
	
	`Biggest group size: 3`

7. Everybody Run! (their code)

To distract the park visitors while the dinosaurs run amok, the hotel staff challenges them to think about clever ways to divide one integer by another without actually using things like division or mod or even multiplication, but rather just adding and subtracting numbers. Being a clever computational thinker who had not learned any lessons from watching dinosaur movies, you have decided to write an efficient program to do just that so you can go watch the dinosaurs from the patio.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case consists of a single integer, a "/" character, and an additional integer. For example, "4/2" would be the input for dividing the numerator 4 by the denominator 2. **You may assume the numerator is divisible by the denominator**.

Output:

For each test case, your program will output an integer quotient. For example, if the input were "4/2", then the output would be the single integer "2".

Examples:

Input:	**Output:**
3 1/1 4/2 52/13	 1 2 4

8. Clever Girl

Robert Muldoon, the original game warden of Jurassic Park, is tired of ensuring that all electric fences surrounding the old velociraptor pen are in order. Instead, he has moved all of the velociraptors into a new, concrete-walled condominium high rise, where they can live a pseudo-urban lifestyle under his ever-watchful eye. However, Muldoon forgot to take into consideration one of the oldest rules in the book: raptors hunt, and hang out, in packs of two. Thus, to convince the raptor population to move into its new condominium residence, Muldoon had to promise each hunting pair that it would share either a wall (that is, the rooms of each of the raptor pairs would be side-by-side), or a ceiling/floor (that is, the rooms of each raptor pair would be directly above or below each other in the high rise).

Jurassic World's corporate overseers have determined that the optimal footprint for this building will be a 3x3 grid of rooms (so, 9 rooms per floor), at a height of **K** floors (so, 9**K** total rooms, arranged as a stack of 3x3 grids). Every raptor is paired. The corporation has asked Muldoon to count every possible configuration of rooms such that no room is left unpaired, *modulo 10,007*. He has asked you to help.

You can assume no raptor will share a room with another raptor. (Raptors do not share.) All rooms must be filled.

Input/Output Format

Input:

The first line in the test data file contains the number **x** of test cases. After this, the **x** test cases are given one by one. Each test case is represented by exactly one positive integer **K**, representing the number of floors of the 3x3 raptor condominium high rise. You can assume **K** is a positive integer less than 5,000.

Output:

The output is the number of legal configurations of rooms such that no pair of raptors is allocated a nonadjacent room, *modulo 10,007.*

Examples:

Input:	**Output:**
3 2 4 1576	 229 7728 229