1. Caesar

Input File: caesar.dat

Prompt: Caesar really likes his ciphers, so much so that one is named after him! In a standard Caesar cipher, you substitute a letter with the letter n letters after it. For example, if n = 1, A will become B, C will become D, Z will become A, and so on. But this cipher is too easy to crack! Instead Caesar wants to shift up his cipher by first rotating the letters and then shifting the text to the left. For example, if you shift the string CAESAR by 1 letter, you will get AESARC. In Caesar's new cipher, you will perform both operations, so CAESAR with a rotation of 1 will go to BFTBSD.

Input: You will receive a certain number of lines, each line consisting of a string of upper case letters and a rotation number separated by a space. The rotation can be any integer on [0,25].

Output: For each line of input, output Caesar's new cipher with the given rotation

Sample Input:

```
CAESAR 1
ABCZ 5
```

Sample Output:

```
BFTBSD
GHEF
```

2. Subsi

Input File: subsi.dat

Prompt: Subsi likes substituting letters, and so she wrote an algorithm that will replace any instance of one letter with another in a string given a mapping of letters. However, she noticed a problem in her algorithm: If she changes all instances of one letter to another, she won't be able to tell which letters changed and will lose her original string! Therefore, if she wants to use her algorithm to change one letter into another, she must also change that other letter into whatever it's mapped to and so on. For example, if we had a mapping of A->B, B->C, and C->A, if she wanted to apply a substitution, she would have to map all instances of a b and c at the same time. Subsi wants to know how many different ways she can modify a string with her algorithm given a mapping.

Input: You will receive an integer t, indicating the number of test cases to follow. For each test case, you will receive an integer n, indicating the number of unique letters used (1<=n<=26). You will then see n lines of space separated letters c_1 and c_2, which indicates that c_1 is mapped to c_2. A letter can map to itself. All characters will be upper case letters.

Output: For each test case, output the number of unique ways Subsi could use her algorithm to modify a string that contains all letters in the alphabet on a separate line.

Sample Input:

```
1
5
A B
B C
C A
Z Y
Y Z
```

Sample Output:

```
6
```

3. Inscri

Input File: inscri.dat

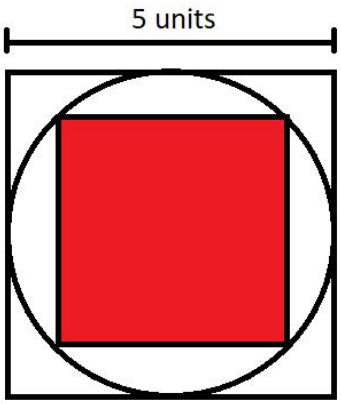
Prompt: Inscri loves the problem of inscribing shapes inside of others. Whenever he sees a square, he must inscribe the largest possible circle inside of it, and whenever he sees a circle, he must inscribe the largest possible square inside of it. Inscri also likes to recurse this process, drawing a square inside a circle inside a square and so on. Given a square and the number of total number of inscriptions to make, output the area of the final inscription to the hundredths place

Input: You will receive an integer t, indicating the number of test cases to follow. For each test you will receive a number l, indicating the original side length of the square followed by a space and the number i (1<=i) indicating the number of inscriptions.

Output: For each test case, output the area of the final inscribed shape to the nearest hundredths place

Sample Input:

```
1
5 2
```



Sample Output:

```
12.50
```

4. Golfo

Input File: golfo.dat

Prompt: Golfo loves playing golf. The golf fields he plays on are constructed of three different types of materials, grass, rough, and sand, which each require a different amount of energy for the ball to get onto. Golfo represents the golf course as a set of numbered holes on a grid, with each space on the grid represented by either a digit, the hole number, or a letter representing the type of material. It is your job to determine the minimum amount of energy for golfo to traverse the course from hole to hole. Traveling onto a grass tile requires 1 energy, onto a rough tile requires 3, a sand tile 5, and a hole 1. You start at hole zero and you stop at the final hole, which will be a digit from 1 to 9. You can not travel diagonally. You are allowed to go over holes to create a shortest path.

Input: On the first line you will first receive two space separated integers, n and m, indicating the numbers of rows and columns respectively. On the next n lines, you will receive m characters, representing what occupies each grid space. A g is for grass, r for rough, s for sand, and a digit from 0 to 9 for hole numbers.

Output: Output the minimum energy required on a path that passes through all holes in order, starting at hole zero.

Sample Input:

```
6 6
0ssss2
ggsssr
rg1ssg
rrgssg
ssgggg
rrssss
```

Sample Output:

```
15
```

Explanation:

You follow the bold path

0ssss**2**
ggsss**r**
r**g1**ss**g**
rr**g**ss**g**
ss**gggg**
rrssss

5. Mirrin

Input File: mirrin.dat

Prompt: Mirrin is writing a top-down 2d video game out of ascii text. In it, he wants to have real time mirror images of ascii text displayed calculated based on mirror position. Mirrin's game has two types of mirrors, horizontal (-) and vertical (|). Given a state without mirror images computed, output the proper mirror calculation. The mirrors will never be recursive.

Input: On the first line, you will receive two integers, n and m, indicating the number of rows and columns in the text before the mirror images need to be calculated. This includes a one character wide border of spaces and mirrors around the outside of the text to be calculated.

Output: You will output the image after the reflections have been processed.

Sample input:

```
7 7
   - -
|ahqpd
 oi2(a|
 *K2la|
|M93nA
|1samh
 --
```

Sample output:

```
        a m
        3 A
        2 a
        2 a
        q d
        - -
dpqha|ahqpd
      oi2(a|a(2io
      *K2la|al2K*
An39M|M93nA
hmasl|1samh
      --
      ls
      M9
      *K
      oi
      ah
```

6. Logi

Input File: logi.dat

Prompt: Logi loves logs so much that she wrote this function, but it takes too long to run :(. Fix her algorithm for her.

```
public static int logi(int a, int b){
        return ((int) (Math.log(a)/Math.log(b)) + ((a==2)?
(b==2)? 0 : logi(a,b-1) : (b==2) ? logi(a-1,b) : logi(a-1,b) +
logi(a,b-1)))%1000000007;
    }
```

Input: Two integers, a and b, separated by a space, representing the input for the logi function. 2<=a,b<=1000.

Output: Output logi(a, b).

Sample Input:

```
5 5
```

Sample Output:

```
54
```

7. Physo

Input File: physo.dat

Prompt: Psycho loves weights and physics, so he does his best to combine both. Physo has a massless plane with weights attached at specific coordinates. Given the center point that Physo wants to balance the plane at and the mass of the last remaining weight, determine the coordinates where the last remaining weight must be placed, rounded to the hundredths place.

You should probably already know the formula to calculate this. If you don't, try searching "balancing weights on a pivot" or "how to balance a seesaw with different weights", and find the 2d alternative. Remember, you're allowed to use google for this competition!

Input: On the first line you will receive three integers, n, x, y, and w, representing the number of points, the center pivot x coordinate, the center pivot y coordinate, and the mass to add to the plane respectively. On the next n lines you will receive three integers, xi, yi, and wi, representing the weight of one of the fixed points on the plane.

Output: Output the coordinates of where to put the weight, rounded to the hundredths place

Sample Input:

```
5 1 2 4
1 3 1
1 1 1
2 3 5
0 1 5
1 1 8
```

Sample Output:

```
1.00 4.00
```

8. Objo

Input File: objo.dat

Prompt: Objo, who just goes by Joe, really likes manipulating objects. However, Objo is tired of things being simple in their polymorphism. Class Dog extends Animal?! - Peasantry he says! Objo wants to have a class that can make and classify any object in the world. To combat this problem, Objo came up with six objective criteria to classify every object: a string, a character, an integer, a floating point number, and how much Objo likes the object on an scale of one to ten.

Objo would like to manipulate these objects with the commands:
Push <String> <Character> <Integer> <Double> <Double>
Pop
Sort <String|Character|Integer|Double|Objectivity>

Whenever Objo calls the push method, he wants to add an object to a data structure. Whenever Objo calls sort, he wants to change his data structure to start sorting by whatever new term he chooses. When Objo calls pop, he wants to retrieve the object so that for every other object `<Object to Retrieve>.<Value chosen by sort>.compareTo(<Other Object>.<Value chosen by sort>) < 0`, as defined by Java's compareTo method, aka the smallest value. You will never have to compare with duplicate values.

Input: On the first line, you will receive an integer n, denoting the number of lines to follow, each containing a separate command. The first line will always be the command Sort, all other lines can be any other command. The push command consists of Push followed by a string, character, integer, double, and double, all separated by spaces. The word Pop is all that is needed for the pop command. The sort command will start with the word Sort followed by one word, either String, Character, Integer, Double, or Objectivity, indicating the type of data to sort by, referring to the data in the push command respectively. The String and Character fields can only consist of any standard ascii character (Don't worry about emojis).

Output: Every time the pop command is called, you will output one line with the object's String, Character, Integer, Double, and Objectivity, all space separated. All doubles should be rounded to the hundredths place.

Sample Input:

```
7
Sort String
Push Cat c 5 5.00 1.00
Push Dog _ 6 7.00 10.00
Pop
Push Elephant e 10000 1002.311 7.0
Sort Objectivity
Pop
```

Sample Output:

```
Cat c 5 5.00 1.00
Elephant e 10000 1002.31 7.00
```

9. Treerie

Input File: treerie.dat

Prompt: Treerie, whose name rhymes with Drearie, her sadder sister, gets her joy in life through trees. Treerie loves her trees so much that she wishes to create a christmas tree out of a graph theory tree. Treerie wants to decorate each node in her graph with an ornament while still keeping it balanced. You see, given a root node, Treerie wants every single subtree from a node to have the same weight. Treerie can weight each of her nodes with any integer value above 0. For each choice of a root node, tell Treerie the minimum weight of the balanced tree she can construct, mod 1000000007.

Input: On the first line, you will receive an integer n (1<=n<=500), indicating the number of nodes. Nodes are numbered from 1 to n inclusive. On the next n-1 lines, you will receive two integers a and b separated by a space, indicating an undirected edge on the tree.

Output: Output n lines that indicate the minimum total weight of the tree given that the tree is rooted at the node that corresponds to the line number, mod 1000000007.

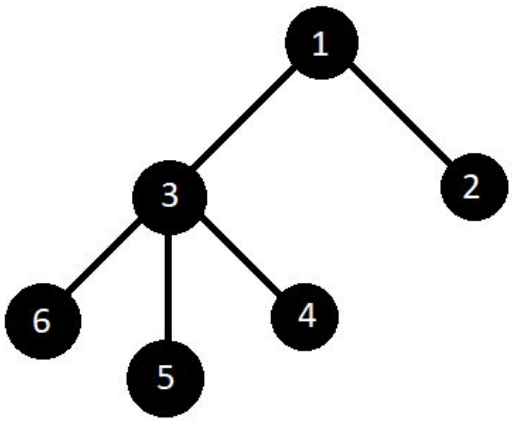
Sample Input:

```
6
1 2
1 3
3 6
3 5
4 3
```

Sample Output:

```
9
6
9
8
8
8
```

Explanation:



Let's look at this for when the root is at one. 4, 5, and 6 can have weights of 1 each, balancing the subtree 3. 3 can also have a weight of 1. For the 1 subtree to balance though, the 2 must have a weight of 4. Giving 1 a weight of 1, we then have a total weight of 9.

10. Functo

Input File: functo.dat

Prompt: Functo made a mathematical function for you, and wants you to solve it! On the domain [1,100] print out all x values that correspond to the requested y value rounded to the hundredths place in ascending order.

$$y = x + x^{1.5} - x^{1.4} + \log(x) + \sqrt{x} + \frac{1}{x+1} + \sin(x) - 1.1^{x} + 1.2^{(x-50)} + \cos(x)$$

Graphing this might help you

Input: On the first line you will receive an integer n (1<n<1000), indicating the number of lines to follow. On the following n lines, you will receive a double y, indicating the value to solve for.

Output: For each y value print out a line listing all x values between 1 and 100 inclusive that equal the y value, rounded to the hundredths place, space separated and in ascending order.

Sample Input:

```
1
5.0
```

Sample Output:

```
4.02 5.55
```

11. Xorio

Input File: xorio.dat

Prompt: Xorio enjoys xoring things. IDK man, I ran out of ideas a long time ago. Don't think too much into this problem. Given a list of numbers, Xorio wants to xor them all together and count the number of ones in the binary representation of the number. But with a twist - literally. Xorio wants to see how he can maximize the total number of ones after performing bitwise rotations on each of the integers in the list.

For example, if Xorio had 2 4 bit numbers 1100_2 and 1101_2, Xoring them normally he would get 0001_2, which only contains one 1. However, if he performs a bitwise rotation, he can then Xor 1100_2 and 0111_2, producing 1011_2 when xored, which has three ones.

Input: On the first line you will receive an integer n (1<=n<=10), indicating the number of lines to follow. On the following n lines, you will receive four space separated integers for you to solve with.

Output: For each of the n lines of input, output the maximum possible number of ones that can remain by performing bitwise rotations and xoring all 4 numbers.

Sample Input:

```
2
65535 65535 0 0
255 255 255 255
```

Sample Output:

```
32
32
```

12. Strin

Input File: strin.dat

Prompt: If you were looking for some grand and hard final problem, sorry. I don't want to write any more problems. But Strin still has one! Strin has a list of strings that take up too much space, so he wants to compress them. His grand idea for compression is that if one string shares an ending with the beginning of another, just combine them together. Tell Strin the minimum length of a final string he can create with this compression method, while keeping all strings in order. If you can't combine two strings together, just append them to each other. No whitespace allowed!

For example, if Strin has the words aaabbb and bbbccc he could compress them into aaabbbccc, with a total length of 9.

Input: On the first line you will receive a number n (1<=n<=10000), indicating the number of lines to follow. Each of the following lines will contain one lowercase string with fewer than 33 lowercase letters.

Output: Output the length of the shortest compressed string you can make on one line.

Sample Input:
```
2
aaabbb
bbbccc
```

Sample Output:
```
9
```