SPRING 2014 PROBLEM PACKET

ADVANCED TIER

SPONSORED BY:

RULES AND REGULATIONS

- All participants must be Penn State students and must provide an active PSU email address.
- Teams will be broken up by skill level--intermediate and advanced. Students that have taken 300-level and up CMPSC classes will be put in the advanced tier. Those who have taken 200-level and below will be put in the intermediate tier. All non-CSE students will be placed in the intermediate tier.
- Teams may consist of 1-3 people of any skill level.
- A team's skill level will determined by the highest skill level of a member in the team.
- Intermediate teams have the option to compete in the advanced tier, but advanced teams may not compete down.
- Teams will have four hours to complete as many problems as possible.
- The competition will consist of 10 problems ranging in difficulty from easy to hard for both skill levels.
- Teams are limited to one computer that they must bring themselves.
- Teams are permitted to bring books and use Internet sources during the competition.
- Solutions to problems must be written in C, C++, or Java.
- Source code for solutions will be uploaded to the judges who will compile and test teams' solutions. Compiled solutions will be limited to an execution time that will vary per problem. If a solution takes too long to finish executing, it is incorrect.
- Only standard libraries may be used.
- The C99 and C++98 standards are used for C and C++ respectively with GCC. Java solutions are compiled with Java 7.
- All input must be through standard input, and all output must be through standard output. Otherwise, a solution is incorrect.

HINTS

- All submissions are subject to a time limit; take too long, and your process is killed.
- If you use C or C++, you must use `int main()` rather than `void main()` and NEVER use `system("pause")`.
- If you need `stdlib.h` or `math.h`/`cmath` you MUST explicitly `#include` these even though Visual Studio does not enforce this.
- Do not prompt for input. Anything your program prints is considered output and will be judged.
- Read each problem carefully and pay attention to the given input bounds.
- The submission system needs the Java JDK installed on your computer to run. It's a big download, so do that now if you haven't already.
- You'll also need to download the submission system from http://bit.ly/1cUZjjT.

1 | THE LOTTERY

PROBLEM

You're a software engineer at a company called Initech. Unfortunately, Initech has begun the process of downsizing. Rather than keeping their best-qualified candidates, they've decided to lay off people through a lottery.

The lottery is run by placing all employees in a line. Your boss, Bill Lumbergh, counts off from 1 to *X*, where *X* is a number that's chosen by taking a card off the top of a deck. Every time *X* is reached, that employee is fired and removed from the line. The counting starts at 1 again with the next employee in the line. After Lumbergh gets to the end of the line, he draws another card from the top of the deck, and starts the count again at 1 with the first person in the line. The last *N* people in line get to continue working at Initech.

Your friend Lawrence has figured out a way to replace the real deck of cards with a stacked one just before the lottery happens, but you still don't know how many employees are supposed to be at this lottery until you get there. Therefore, since you're the best software engineer at Initech, you write a program that takes in the stacked deck you use and the number of employees at the lottery, and outputs the positions in line which will ensure that you won't end up in the unemployment line.

INPUT

Each test case is a line of 22 numbers, all integers. The first number, $1 \le X \le 50$ is how many employees are required to be a part of this lottery. The next number, $1 \le N \le X$, is how many people Initech wants to keep employed. The last 20 numbers represent the values of the top 20 cards in the stacked deck. The card values are any integer from 1 to 11.

OUTPUT

The output is one line per test case, consisting of the number of positions in the line where you should try to be. If the lottery isn't over by the time the 20th card is used, print the positions of all of the remaining employees in the line-up.

SAMPLES

Input

```
8 2 3 2 4 7 6 9 6 11 11 6 3 2 7 6 4
7 1 5 3 2
45 7 11 6 8 2 1 3 5 9 10 4 6 7 4 2 2
3 4 5 5 3
```

Output

```
1 4
27 29 34 36 38 42 45
```

2 | STUDY BUDDIES

PROBLEM

A certain professor is notorious for giving massive, time-consuming problem sets. Two students in the class decided that their time would be spent more wisely if they collaborated. They want to split the problems up on each assignment so that they'll each be doing exactly half of the work. They thought that they could just each do half of the problems, but some problems are more difficult and time-consuming than others, which wouldn't be fair to the person who gets the more difficult problems. So now they rate each problem on a scale of 1 to 6 in difficulty. Their goal is to divide the problems up so that they each get the same total value.

This, however, brings up another problem. In some cases, it's impossible to divide the problems up this way. In a simple example, if they only have two problems with two different ratings, then it would be impossible to divide them fairly. Thus, they need a program to determine whether or not they can partition the homework problems fairly.

INPUT

Each line in the input file describes one homework assignment. The line consists of six integers, a_1, … , a_6, where a_i is the number of problems that are valued at i. Each integer is separated by a space, i.e., "1 2 3 4 5 6". The very last line of the input file is "0 0 0 0 0 0", which should not be processed. The total maximum number of problems is 20000 (their professor is out of control).

OUTPUT

For each test case, output "Assignment #*n*", with *n* being the number of the test case. On the next line, print "Yes" if the problems can be divided up evenly and "No" if they can't. After each test case, output a blank line.

SAMPLES

Input

```
6 1 0 1 0 0
1 0 3 0 0 1
0 0 0 0 0 0
```

Output

```
Assignment #1
Yes

Assignment #2
No
```

3 | AS SEEN ON TV

PROBLEM

Your roommate, Mike Teavee, is a huge TV junkie, and over the years, your TV has gradually started breaking from overuse. It still works, but just barely. Every time Mike turns on the TV, it starts at channel 1. Because the remote no longer works, in order to get to the channel he wants, Mike has to press the up and down buttons on his television set. To make matters worse, the up and down buttons no longer increment by one channel – each button skips past a certain interval of channels. Finally, if Mike tries to go to a channel lower than 1 or higher than the number of channels he has available, nothing happens. For example, if he's on channel 1 and presses the down button, the TV will stay on channel 1.

The show Mike wants to watch is on a certain channel. Will he be able to get to that channel, and if so, how many times will he have to press the up and down buttons to do so? You're a good roommate, so you write a program to figure it out for him.

Note: Assume that every number from 1, 2, …, n has a corresponding channel, with n being the total number of channels.

INPUT

The first line is *N* number of test cases, with *N* being an integer from 1 to 100. For each test case, you're given *c d u t* with $2 \le c \le 10^7$, $1 \le d, u \le c-1$, and $1 \le t \le c$, with *c* being the number of channels, *d* being the interval of channels skipped when Mike presses down, *u* being interval of channels skipped when Mike presses up, and *t* being the target channel.

OUTPUT

If it's possible to get the channel Mike wants, output the minimum number of buttons he'll have to press on a single line. If it's not possible, output "Try Netflix!"

SAMPLES

Input

```
4
99 3 7 555
200 2 4 98
36 3 9 25
8 2 2 1
```

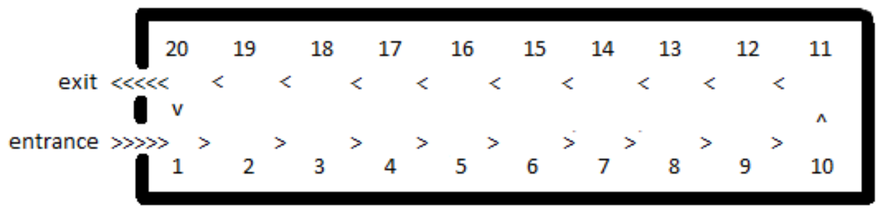
Output

```
82
Try Netflix!
4
0
```

4 | FUN WITH PARKING

PROBLEM

There are many things wrong with your apartment complex, but probably the most frustrating is that it doesn't always have enough parking spaces for all of its tenants. Luckily, there are usually people coming and going at all hours of the day, so if you wait long enough you'll usually get a spot. The lot has 20 spaces, all numbered from 1-20, in which traffic flows one way counter-clockwise starting at space #1:



Once cars have passed space #20, they can either exit the lot or circle around again.

At the start of this problem, all spaces are occupied by cars. Additionally, there are *N* cars waiting for a space, where *N* Is an integer from 1 to 20. Each car that isn't parked is waiting behind one of the occupied spots. When a car leaves its spot, it is occupied by either a car waiting at that position or the car closest to that position (keeping in mind that cars can only go one way). When a car moves up *K* positions to fill a parking space, all other cars waiting in the lot move up *K* spaces. All of the waiting cars circle around the lot if they move past spot #20; none of the waiting cars exit.

INPUT

Input starts with a line from standard input indicating the number of test cases. Following that is a blank line. After that, each test case is split into two parts. The first part is a list of waiting cars by the position they're waiting at, given in an integer. There is one integer per line. Integer '999' marks the end of the waiting cars. The second part is in the same format, but it represents the spaces that the cars leave, in order. Each test case is separated by a blank line.

OUTPUT

The output gives the waiting car's initial position and then its final position: $p_{initial}$ p_{final}, separated by a space. They must appear in the order that the initial waiting positions are given in the input. If the car is not able to park, print 0 for its p_{final}. Print a blank line between test cases.

SAMPLES

Input

```
1

5
20
18
7
3
999
3
2
12
19
```

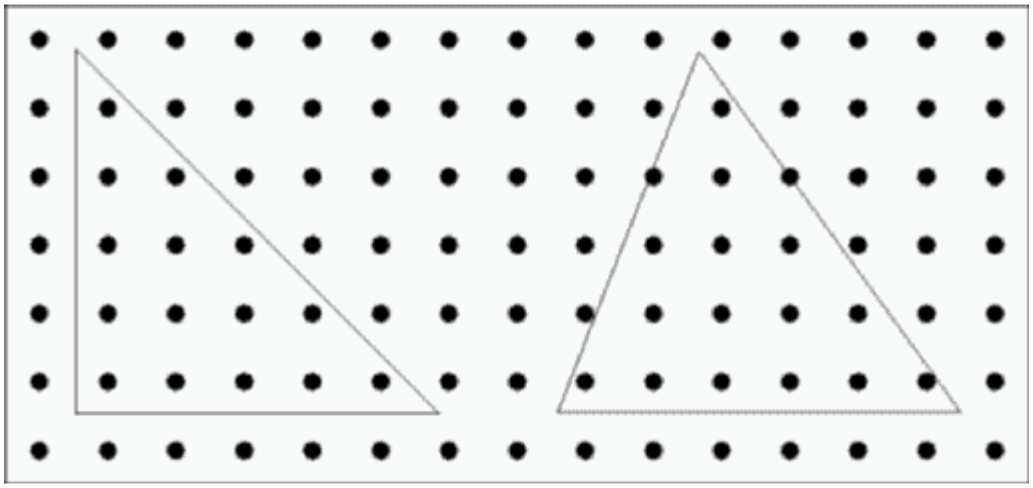
Output

```
5 19
20 2
18 0
7 12
3 3
```

5 | THE LORAX

PROBLEM

The Lorax is a protector of the forest. He loves his trees and will do anything to save them. All of the trees in the forest are uniformly spaced in both directions due to the Lorax's careful tending. The Lorax counts the trees starting from the bottom left of his forest:



A logging company has decided to cut down the trees in the Lorax's forest! They want to cut down all the trees contained within a triangular region of the forest. Write a program that will determine how many trees the Lorax has to save from the loggers. You can assume that any tree lying exactly on the border of the triangle is considered to be in the triangle.

INPUT

Each line will contain 6 real numbers in the range of 0. – 100 that represent the coordinates of the triangle. The end of the file will be terminated by 6 zeroes.

OUTPUT

The output will be one line per triangle that will contain the number of trees the Lorax has to save.

SAMPLES

Input

```
1.8 1.8 1.8 7 7 1.8
10.9 7 8.3 1.8 14.3 1.8
0 0 0 0 0 0
```

Output

```
15
17
```

6 | TRUST YOUR CALCULATOR. IT’S SOMETHING TO COUNT ON.

PROBLEM

Do you remember doing long multiplication back in the 4th grade? Do you remember how stupid you thought it was because you knew calculators existed? Now you get to write your own calculator that would have helped out your 4th grade self! Spread your knowledge to the younger generation by writing a long multiplication calculator.

INPUT

Each line will contain two integers *x* and *y*, separated by one or more characters of whitespace. Each integer will have no more than 10 digits. The last line of input will contain a ‘#’ to mark the end of input

OUTPUT

For each line, multiply *x* by *y*, displaying the results shown in the example below. If there are less than 2 lines of numbers between the horizontal lines of numbers, do not print the second line. Display 0 digits only when they are significant. The length of the first line should be the same as the number of digits as *x* or *y* depending on which one is bigger. The length of the second line should be the same number of digits as the product.

SAMPLES

Input

```
5   8
134      62
21  3
15     0
#
```

Output

```
 5
 8
 -
40

 134
  62
 ---
 268
8040
----
8308

21
 3
--
63

15
 0
--
 0
```

7 | ANCIENT LANGUAGE

PROBLEM

On a dig in Egypt, your friend finds a set of stone tablets with a new language that uses letters that have English letter equivalents. The tablets contained an index that showed your friend the ordering of the letters. Your friend tried to determine what the order of the characters was, but she gave up in frustration. Your friend asks you to write a program to finish her work. Your program will take a set of strings that have been sorted according to the index found on the stone tablets.

INPUT

The input is an ordered list of strings of capitalized letters, one string per line. The size of the string n is $1 \leq n \leq 20$. The end of the list is signaled by a line with the character '%'. The order in which the strings are entered matters.

OUTPUT

A single line containing capitalized letters in the order they should be sorted. All letters input may or may not be used in the output.

SAMPLES

Input	**Output**

`JFF`
`JBF`
`JBXM`
`JX`
`%`
	`FBX`

`PMK`
`PPK`
`PKL`
`KLM`
`LM`
`%`
	`MPKL`

8 | THE CAKE BALL CONFECTIONER

PROBLEM

Your friend has demanded your assistance in making cake balls. These will be made by mixing yellow cake, strawberries, and vanilla frosting in a certain ratio, after which your friend will roll them into balls and cover them in chocolate. Unfortunately, your friend has neither pure yellow cake, strawberries, nor vanilla frosting. However, he does have frosted yellow cake, strawberry/vanilla frosting filling, and other combinations of these three ingredients. Your task is to write a program to determine whether making the desired cake balls is possible with the given ingredients.

As an example, you have half a strawberry cake eaten by someone who didn't like frosting. Thus, it has a cake:strawberry:frosting ratio of 1:1:2. You also have a cake with the same ingredients in a 5:2:3 ratio. By mixing these in a 1:3 ratio, you could get a 16:7:11 ratio. A 5:5:9 ratio, however, would be impossible.

INPUT

You have been given multiple sets of inputs to test your program, based on the ingredients available and ratios desired. The first line indicates the number of different cakes in your supply (*n*). The next *n* lines indicate the cake:strawberry:frosting ratios of each of the cakes. At least one number in each of these lines will be non-zero, and none will have weights greater than 15000. Finally, the last line is the desired ratios for the final cake ball filling mixture. The inputs will be terminated with a 0.

OUTPUT

The output should be the word "recipe," followed by the input number, a line break, and then either "impossible" or "possible" depending on whether or not cake balls can be made given the ratios. Multiple outputs should be separated with whitespace.

SAMPLES

Input

```
2
1 1 2
5 2 3
16 7 11
2
1 1 2
5 2 3
5 5 9
0
```

Output

```
Recipe 1
Possible

Recipe 2
Impossible
```

9 | PRODUCING PAIRS

PROBLEM

Bored with the traditional 52-card deck, Croupier decides to modify his deck to include a varied number of suits (this way it can have more pictures!). From this completely randomized deck, he draws cards and place them on the table. Once a pair (same suits) is found, both cards that form that pair are removed.

One day, Croupier was experimenting with a five-suit deck (hearts, clubs, spades, diamonds, and circles). Interestingly, the majority of the time, there were either two or three cards on the table. Your problem is to write a program to determine the probability that exactly *a* cards are on the table after *b* cards have been drawn, given that there are *c* suits in the deck.

INPUT

The input has several test cases, each in its own line. Each test case includes, in this order, *c b a*, separated by spaces, where:

- $0 \leq c \leq 100$
- $a, b \leq 1000000$

It is then terminated with a 0.

OUTPUT

The output should be one number per line, rounded to three decimal places.

SAMPLES

Input	**Output**
`5 100 2`	`0.625`
`5 4 2`	`0.704`
`0`	

10 | SOCIAL INSECURITY

PROBLEM

Recently, the University of Maryland had 310,000 of its student records hacked. For some unknown reason (split personality? unlawful friends/roommates? magic?), this data has fallen into your hands. ...Well, may as well make it look pretty right? Nothing else you'd do with it.

You will sort this data based on name. However, some names seem to have had their ASCII codes truncated! There are weird names like 75 fred 50 Bob-George. To make names and their corresponding social numbers as easy to find as possible, you decide to follow the following conventions:

1) Only the letters A to Z will be used, ignoring case.

2) Numbers will be sorted based on their name. For example, 20 Billy-Frank may come right before Twenty Billy-Fred.

3) All special non-numerical/alphabetical characters will be treated as spaces. Therefore, Tom-Xavier would come after Tom Victor but before Tommy.

4) Multiple spaces or special non-numerical/alphabetical characters will be treated as a single space.

5) Any names in all capitals, like JKF, will be assumed to be an acronym and be treated as if there are spaces between the letters. Therefore, JKF would come before Jeremy and after J. Roberts.

You make a few observations that may make this process easier:

1) Each word is either all alphabetical or all numerical. There are no names like fr3d j4ck r0b3rtz.

2) No words start with special characters.

Social security numbers are 9 digits long, and range from 000-00-0000 to 999-99-9999.

INPUT

Your input will be the data of students and faculty members of UMD, one per line. It will be the social security number followed immediately (no space) by the name.

OUTPUT

Your output should sort names alphabetically according to the conventions listed above. The first 52 columns will contain the name, left justified, with any extra space filled with a space. This will then be followed by a social security number, which will have a space between the third and fourth and fifth and sixth digits.

SAMPLES

Input

```
12345678975 fred 50 Bob-George
13489723020 Billy-Frank
324563568Twenty Billy-Fred
789456695Tom-Xavier
384238434Tom Victor
762367678Tommy
345537782JKF
453452345Jeremy
793456723J. Roberts
```

Output

```
J. Roberts                                     793 45 6723
JKF                                            345 53 7782
Jeremy                                         453 45 2345
75 fred 50 Bob-George                          123 45 6789
20 Billy-Frank                                 134 89 7230
Twenty Billy-Fred                              324 56 3568
Tom Victor                                     384 23 8434
Tom-Xavier                                     789 45 6695
Tommy                                          762 36 7678
```