University of Information Technology and Sciences
Department of Computer Science and Engineering
Data Structures and Algorithms I Lab
CSE 212
Final Assignment
Dead Line: 8th January, 2024

Problem #01

In the movie "Blues Brothers", the orphanage where Elwood and Jake were raised may be sold to the Board of Education if they do not pay 5000 dollars in taxes at the Cook County Assessor's Office in Chicago. After playing a gig in the Palace Hotel ballroom to earn these 5000 dollars, they have to find a way to Chicago. However, this is not so easy as it sounds, since they are chased by the Police, a country band and a group of Nazis. Moreover, it is 106 miles to Chicago, it is dark and they are wearing sunglasses.

As they are on a mission from God, you should help them find the safest way to Chicago. In this problem, the safest way is considered to be the route which maximises the probability that they are not caught.

Input

The input file contains several test cases.

Each test case starts with two integers **n** and **m** ($2 \leq \mathbf{n} \leq 100$, $1 \leq \mathbf{m} \leq \mathbf{n}*(\mathbf{n}-1)/2$). **n** is the number of intersections, **m** is the number of streets to be considered.

The next **m** lines contain the description of the streets. Each street is described by a line containing 3 integers **a**, **b** and **p** ($1 \leq \mathbf{a}, \mathbf{b} \leq \mathbf{n}$, $\mathbf{a} \neq \mathbf{b}$, $1 \leq \mathbf{p} \leq 100$): **a** and **b** are the two end points of the street and **p** is the probability in percent that the Blues Brothers will manage to use this street without being caught. Each street can be used in both directions. You may assume that there is at most one street between two end points.

The last test case is followed by a zero.

Obs.: The safest path for the sample input is 1 -> 4 -> 3 -> 5

Output

For each test case, calculate the probability of the safest path from intersection 1 (the Palace Hotel) to intersection **n** (the Honorable Richard J. Daley Plaza in Chicago). You can assume that there is at least one path between intersection 1 and **n**.

Print the probability as a percentage with exactly 6 digits after the decimal point. The percentage value is considered correct if it differs by at most 10^{-6} from the judge output. Adhere to the format shown below and print one line for each test case.

Sample Input	Sample Output
5 7 5 2 100 3 5 80 2 3 70 2 1 50 3 4 90 4 1 85 3 1 70 0	61.200000 percent

Problem #02

John is working in a diamond mine, trying to extract the highest number of diamond "<>". He must exclude all sand particles found "." in this process and after a diamond can be extracted, new diamonds can be formed. If he has as an input. <... << .. >> ....> .... >>>. three diamonds are formed. The first is taken from <..> resulting. <... <> ....> .... >>>. The second diamond is then removed, leaving. <.......> .... >>>. The third diamond is then removed, leaving at the end ..... >>>. without the possibility of extracting new diamonds.

Input

Read an integer N that is the number of test cases. Then follows N lines each up to 1000 characters, including "<" ,">" and "."

Output

You must print the amount of diamonds that can be extrated in each test case.

Input Sample	Output Sample
2 <..><.<..>> <<<..<.....<<<<....>	3 1

Problem #03

There is a bag-like data structure, supporting two operations:

1 x
Throw an element x into the bag.

2
Take out an element from the bag.

Given a sequence of operations with return values, you're going to guess the data structure. It is a stack (Last-In, First-Out), a queue (First-In, First-Out), a priority-queue (Always take out larger elements first) or something else that you can hardly imagine!

Input

There are several test cases. Each test case begins with a line containing a single integer **n** (1 <= **n** <= 1000). Each of the next **n** lines is either a type-1 command, or an integer 2 followed by an integer **x**. That means after executing a type-2 command, we get an element **x** *without error*. The value of **x** is always a positive integer not larger than 100. The input is terminated by end-of-file (EOF). The size of input file does not exceed 1MB.

Output

For each test case, output one of the following:

stack
It's definitely a stack.

queue

It's definitely a queue.

priority queue
It's definitely a priority queue.

impossible
It can't be a stack, a queue or a priority queue.

not sure
It can be more than one of the three data structures mentioned above.

Sample Input	Sample Output
6 1 1 1 2 1 3 2 1 2 2 2 3 6 1 1 1 2 1 3 2 3 2 2 2 1 2 1 1 2 2 4 1 2 1 1 2 1 2 2 7 1 2 1 5 1 1 1 3 2 5 1 4 2 4	queue not sure impossible stack priority queue

Problem #04

Given in an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck:

Throw away the top card and move the card that is now on the top of the deck to the bottom of the deck.

Your task is to find the sequence of discarded cards and the last, remaining card.

Each line of input (except the last) contains a number $n \leq 50$. The last line contains 0 and this line should not be processed. For each number from the input produce two lines of output. The first line presents the sequence of discarded cards, the second line reports the last remaining card.

Input

The input file contains a non determinated number of lines. Each line contains an integer number. The last line contain the number zero (0).

Output

For each test case, print two lines. The first line presents the sequence of discarded cards, each number separated by a comma ',' and one blank space. The second line reports the last remaining card. No line will have leading or trailing spaces. See the sample for the expected format.

Input Samples	Output Samples
7 19 10 6 0	Discarded cards: 1, 3, 5, 7, 4, 2 Remaining card: 6 Discarded cards: 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 4, 8, 12, 16, 2, 10, 18, 14 Remaining card: 6 Discarded cards: 1, 3, 5, 7, 9, 2, 6, 10, 8 Remaining card: 4 Discarded cards: 1, 3, 5, 2, 6 Remaining card: 4

Problem #05

Raju and Meena love to play with Marbles. They have a lot of marbles with numbers written on them. In the beginning, Raju would place the marbles one after another in ascending order of the numbers written on them. Then, Meena would ask Raju to find the first marble with a certain number. She would count 1...2...3. Raju gets one point for correct answer, and Meena gets the point if Raju fails. After some fixed number of attempts, the game ends and the player with maximum points wins. Today it's your chance to play as Raju. Being a smart kid, you have in your advantage the computer. But don't underestimate Meena, she wrote a program to keep track how much time you're taking to give all the answers. So now you have to write a program, which will help you in your role as Raju.

Input

There can be multiple test cases. The total number of test cases is less than 65. Each test case consists begins with 2 integers: **N** the number of marbles and **Q** the number of queries Meena would make. The next **N** lines would contain the numbers written on the **N** marbles. These marble numbers will not come in any particular order. Following **Q** lines will have **Q** queries. Be assured, none of the input numbers are greater than 10000 and none of them are negative.

Input is terminated by a test case where **N** = 0 and **Q** = 0.

Output

For each test case output there must be a serial number of the test case. For each query, write one line of output. The format of this line will depend on whether the number is consulted or not written in one of the marbles.

The two different formats are described below:
'x found at y', if the first marble with number x was found at position y. Positions are numbered *1, 2,..., N*.
'x not found', if the marble with number x is not present.

Input Sample	Output Sample
4 1 2 3 5 1 5 5 2 1 3 3 3 1 2 3 0 0	CASE# 1: 5 found at 4 CASE# 2: 2 not found 3 found at 3

Problem #06

Read three integers and sort them in ascending order. After, print these values in ascending order, a blank line and then the values in the sequence as they were readed.

Input

The input contains three integer numbers.

Output

Present the output as requested above.

Input Sample	Output Sample
7 21 -14	-14 7 21 7 21 -14
-14 21 7	-14 7 21 -14 21 7

Problem #07

In this problem your job is to read some positive and integer numbers and print how many times each number appears in the input, you must write each of the distinct values that appear in the input, ordering by ascending value.

Input

The input contains only one test case. The first line of input contains one integer N, which indicates the quantity of numbers that will be read to X ($1 \leq X \leq 2000$) in the sequence. Each number don't appears more than 20 times in the problem input.

Output

Print the output according to the example provided below, indicating how many times each number appears in the input file, by ascending order of value.

Input Sample	Output Sample
7 8 10 8 260 4 10 10	4 aparece 1 vez(es) 8 aparece 2 vez(es) 10 aparece 3 vez(es) 260 aparece 1 vez(es)

Submission Guidelines:

1. **Adherence to Deadlines**:
   All submissions must be made within the specified deadline. Late submissions may not be accepted or evaluated.
2. **Folder Structure**:
   - Create a main folder named after your student ID (e.g., `04323200051040`).
   - Inside the main folder, create subfolders for each problem, named according to the problem (e.g., `Problem 1`, `Problem 2`).
   - Ensure each subfolder contains:
     - The solution to the respective problem (in an appropriate file format, such as `.cpp`, `.py`, `.txt`, etc.).
     - A `.png` file showing the output, if applicable. Include any additional files required to verify your solution.
3. **File Compression**:
   - Compress the main folder into a single `.zip` file before submission. Ensure the compressed file retains the naming structure (e.g., `04323200051040.zip`).
4. **Submission Format**:
   - Submit the `.zip` file through the designated submission portal or email as instructed.
   - Ensure the file is uploaded correctly and verify its integrity before submission.
5. **General Instructions**:
   - Ensure all solutions are complete and thoroughly tested before submission.
   - Include comments in your code for clarity and ease of understanding.
   - If additional documentation (e.g., a `README` file) is required, include it in the corresponding problem folder.