UW-Madison ACM-ICPC Individual Placement Test

October 2nd, 2016

Setup

Before the contest begins, log in to your workstation and set up and launch the PC2 contest software using the following instructions. You will use this program to submit problem solutions, receive the judges' answers, and communicate clarification requests.

1. Download the custom PC2 package into a directory (different from last year) of your choosing from: `www.cs.wisc.edu/~dieter/ICPC/16-17/pc2_16-17tar.gz`
2. In a terminal window, cd to the directory where you downloaded the package and type `tar -xzvf pc2_16-17.tar.gz`
3. Type `cd pc2-9.3.3` (different from previous line) followed by the command `bin/pc2team` – this brings up your PC2 terminal that will be your interface to the judges during the contest.
4. Log in using the login ID and password given to you by the judges when you arrive. They will be of the form teamX where X is an integer.

The Contest

Begin the contest by solving the problem on the next page, `count`. This is a warmup problem designed to get you used to submitting problems via PC2. Code your solution to the problem and submit it as follows:

1. Click on the 'submit run' tab in your PC2 window.
2. In the dropdown menu labeled "Problem", choose `count`. Choose the programming language you used from the "language" dropdown menu. Then select your source code file by clicking the "select" button in the 'main file' section.
3. Submit your code by clicking "Submit" (note: clicking "Test" doesn't really do anything unless you've created your own test files, so don't expect it to automatically test your program against the sample input).
4. Wait – you will receive a judgment from the judge shortly by way of a pop-up window. If your answer comes back something other than "Yes", try again.

All input comes from standard in, all output should be sent to standard out. You may use any online Java, C++, or Python documentation, but not any other resource. You may use the printer at any time. Collaboration with others or searching the web for solutions to these problems is prohibited. Please turn off your cell phones. You may submit problem clarifications via the PC2 program at any time, but please read the problems thoroughly before doing so.

Warmup Problem: Count

Can you count from one up to any number N? Write a program to prove it!

Input

The input begins with a single number that describes the number of test cases. Each test case follows on its own line, and consists of a single positive integer $N \leq 1\,000$ that describes how high you should count for that test case.

Output

The output for each test case should be on its own line, and consist of the numbers 1 through N (inclusive), each separated by a space.

Sample Input

```
3
3
5
10
```

Sample Output

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

The following pages contain seven lettered problems. Please let me know if you do not see all seven.

A: Help out a PhD Student!

Bryce Sandlund forgot how to add two numbers while doing research for his PhD. And now he has a long list of addition problems that he needs to solve, in addition to his computer science ones! Can you help him?

On his current list Bryce Sandlund has two kinds of problems: addition problems of the form "a+b" and the ever returning problem "P=NP". Bryce Sandlund is quite a distracted person, so he might have have to solve this last problem several times, since he keeps forgetting the solution. Also, he would like to solve these problems by himself, so you should skip these.

Input

The first line of input will be a single integer N ($1 \leq N \leq 1\,000$) denoting the number of test cases. Then follow N lines with either "P=NP" or an addition problem of the form "a+b", where $a, b \in [0, 1\,000]$ are integers.

Output

Output the result of each addition. For lines containing "P=NP", output "skipped".

Sample Input

```
4
2+2
1+2
P=NP
0+0
```

Sample Output

```
4
3
skipped
0
```

B: Cake

Warning: The test data for this problem has been modified from its original form. The original problem accepted an answer within 10^{-3} of the exact solution. This version changes to rounding to four decimal places, causing some precision issues.

My birthday is coming up and traditionally I'm serving cake. Not just one cake, no, I have a number N of them, of various tastes and of various sizes. F of my friends are coming to my party and each of them gets a piece of cake. This should be one piece of one cake, not several small pieces since that looks messy. This piece can be one whole cake though.

My friends are very annoying and if one of them gets a bigger piece than the others, they start complaining. Therefore all of them should get equally sized (but not necessarily equally shaped) pieces, even if this leads to some cake getting spoiled (which is better than spoiling the party). Of course, I want a piece of cake for myself too, and that piece should also be of the same size.

What is the largest possible piece size all of us can get? All the cakes are cylindrical in shape and have the same height 1, but the radii of the cakes can be different.

Input

One line with a positive integer: the number of test cases. Then for each test case:

- One line with two integers N and F with $1 \leq N, F \leq 10\,000$: the number of cakes and the number of friends.
- One line with N integers r_i with $1 \leq r_i \leq 10\,000$: the radii of the cakes.

Output

For each test case, output one line with the largest possible volume V such that me and my friends can all get a cake slice of size V. The answer should be given as a floating point number rounded to four decimal places.

Sample Input

```
3
3 3
4 3 3
1 24
5
10 5
1 4 2 3 4 5 6 5 4 2
```

Sample Output

```
25.1327
3.1416
50.2655
```

C: Password

As a malicious hacker you are trying to steal your mother's password, and therefore you have installed a keylogger on her PC (or Mac, so you like). You have a log from your mother typing the password, but unfortunately the password is not directly visible because she used the left and right arrows to change the position of the cursor, and the backspace to delete some characters. Write a program that can decode the password from the given keylog.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with a string L, satisfying $1 \leq Length(L) \leq 1\,000\,000$, consisting of:
  - '-' representing backspace: the character directly before the cursor position is deleted, if there is any.
  - '<' (and '>') representing the left (right) arrow: the cursor is moved 1 character to the left (right), if possible.
  - alphanumeric characters, which are part of the password, unless deleted later. We assume 'insert mode': if the cursor is not at the end of the line, and you type an alphanumeric character, then all characters after the cursor move one position to the right.

Every decoded password will be of length > 0.

Output

For every test case in the input, the output should contain a single string, on a single line: the decoded password.

Sample Input

```
2
<<IP<C>>Cd-
ThIsIsS3Cr3t
```

Sample Output

```
ICPC
ThIsIsS3Cr3t
```

D: Gemstones

The pirate captain and his crew gazed happily upon the chest brimming with gemstones. Diamonds, rubies, sapphires, opals, and many other kinds of gems added to the sparkling collection.

"We're rich, my lads!" the captain said. "Now, here's what we're going to do. Each of us will take two stones now, then we'll bury the rest because... because that's what pirates do! Also, everyone must take two different kinds of gems. We don't know what the merchants in the next port will be buying, and I don't want to listen to any belly-aching from someone who only took rubies if our next port happens to have a flourishing ruby mine nearby."

"Who chooses first?" asked the cabin boy, fearing that he already knew the answer.

"I'll choose first," said the captain, "and then each man from there in order of rank, starting with the first mate down to the cabin boy." The lower-ranked crew members began to grumble. "Then," continued the captain, "the cabin boy immediately chooses his second gem, and we proceed in reverse order until we get to me, and I will be the last to choose my second stone."

The crew quickly agreed, and, almost as quickly, began trying to figure out how to claim the best stones under this arrangement.

Write a program to determine what stone a crewman should pick first in order to maximize the total value they can be guaranteed to acquire no matter how the rest of the crew chooses.

Input

The input will consist of one or more data sets.

Each data set starts with a line containing 2 integers: M, the number of kinds of gemstone available, and N, the number of crewmen who select their first stone after you and select their second before you. Each data set will have $2 \leq M \leq 26$, $0 \leq N \leq 10$. A value of 0 for M indicates the end of input.

The remainder of the dataset consists of M lines, each describing one kind of available gem. The line begins with an uppercase alphabetic character indicating the type of gem (e.g., 'D' might denote diamonds, 'Z' might stand for zircons). No two kinds of gem will have the same code, but the codes may occur in any order. The alphabetic code is followed by $N+1$ integers, each denoting the value in doubloons of one of the $N+1$ most valuable remaining gems of that kind. These will be presented on the line in non-increasing order of value. Values will be in the range $0 \ldots 10\,000$.

Output

For each data set, print a single line containing the letter code for the type of gem that you should pick first to maximize the total value that you can be guaranteed to receive on both picks, no matter what choices the rest of the crew makes. If there is more than one type of gem tied for the maximum total value, print the one that comes earliest alphabetically.

Sample Input

```
4 3
D 10000 9000 8000 6000
R 5000 500 50 5
S 500 500 50 50
Z 1 0 0 0
2 1
S 6000 5000
D 5000 4000
0 0
```

Sample Output

```
R
D
```

E: Random Walk

The Army of Coin-tossing Monkeys (ACM) is in the business of producing randomness. Good random numbers are important for many applications, such as cryptography, online gambling, randomized algorithms, and panic attempts at solutions in the last few seconds of programming competitions.

Recently, one of the best monkeys has had to retire. However, before he left, he invented a new, cheaper way to generate randomness compared to directly using the randomness generated by coin-tossing monkeys. The method starts by taking an undirected graph with 2^n nodes labelled $0, 1, \ldots, 2^n - 1$. To generate k random n-bit numbers, they will let the monkeys toss n coins to decide where on the graph to start. This node number is the first number output. The monkeys will then pick a random edge from this node and jump to the node that this edge connects to. This new node will be the second random number output. They will then select a random edge from this node (possibly back to the node they arrived from in the last step), follow it and output the number of the node they landed on. This walk will continue until k numbers have been output.

During experiments, the ACM has noticed that different graphs give different output distributions, some of them not very random. So, they have asked for your help testing the graphs to see if the randomness is of good enough quality to sell.

They consider a graph good if, for each of the n bits in each of the k numbers generated, the probability that this bit is output as 1 is greater than 25% and smaller than 75%.

Input

The input will consist of several data sets. Each set will start with a line consisting of three numbers k, n, e separated by single spaces, where k is the number of n-bit numbers to be generated and e is the number of edges in the graph ($1 \leq k \leq 100$, $1 \leq n \leq 10$ and $1 \leq e \leq 2\,000$). The next e lines will consist of two space-separated integers v_1, v_2 where $0 \leq v_1, v_2 < 2^n$ and $v_1 \neq v_2$. Edges are undirected and each node is guaranteed to have at least one edge. There may be multiple edges between the same pair of nodes.

The last test case will be followed by a line with $k = n = e = 0$, which should not be processed.

Output

For each input case, output a single line consisting of the word `Yes` if the graph is good, and `No` otherwise.

Sample Input

```
10 2 3
0 3
1 3
2 3
5 2 4
0 1
0 3
1 2
2 3
0 0 0
```

Sample Output

```
No
Yes
```

F: Circular Automaton

A *cellular automaton* is a collection of cells on a grid of specified shape that evolves through a number of discrete time steps according to a set of rules that describe the new state of a cell based on the states of neighboring cells. The *order of a cellular automaton* is the number of cells it contains. A automaton of order n has cells numbered from 1 to n.

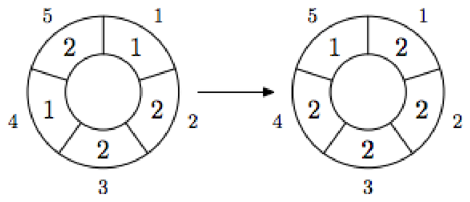
The *order of a cell* is the number of different values it may contain. Usually, values of a cell of order m are considered to be integer numbers from 0 to $m-1$.

One of the most fundamental properties of a cellular automaton is the type of grid on which it is computed. In this problem we examine the special kind of cellular automaton — circular cellular automaton of order n with cells of order m. We will denote such kind of circular automaton as n, m-*automaton*.

A distance between cells i and j in an n, m-*automaton* is defined as $\min(|i-j|, n-|i-j|)$. A *d-environment of a cell* is the set of cells at a distance not greater than d.

On each *d-step*, values of all cells are simultaneously replaced by new values. The new value of cell i after a *d-step* is computed as a sum of values of cells belonging to the *d-environment* of the cell i, modulo m.

The following picture shows a single 1-step of the $5, 3$-automaton.



The problem is to calculate the state of an n, m-automaton after k d-steps.

Input

The input consists of multiple test cases. Continue to process test cases until end of file.

The first line of each test case contains four integer numbers n, m, d, and k ($1 \le n \le 500$, $1 \le m \le 1\,000\,000, 0 \le d < \frac{n}{2}, 1 \le k \le 10\,000\,000$). The second line contains n integer numbers from 0 to $m-1$ —initial values of the automaton's cells.

Output

Output the values of the n, m-*automaton*'s cells after k d-steps.

Sample Input

```
5 3 1 1
1 2 2 1 2
5 3 1 10
1 2 2 1 2
```

Sample Output

```
2 2 2 2 1
2 0 0 2 2
```

G: Race

Professor Hill really likes orienteering, a sport that requires locating control points in rough terrain. To entertain the UW-Madison ICPC Placement Test participants, Prof. Hill wants to organize an orienteering race. However, it would be too harsh for the participants to be outdoors as the cold Madison winter arrives, so he decided to jump on the new trend of indoor races, and set the race inside the Computer Sciences Department.

Prof. Hill has already decided on the locations of the control points. He has also decided on the exact length of the race, so the only thing remaining is to decide in which order the control points should be visited such that the length of the total race is as he wishes. Because this is not always possible, he asks you to write a program to help him.

Note from the ICPC Student Coach: the race for this year has been cancelled since we neglected to apply for an orienteering permit in time from the university administration. (We still need you to solve the problem so that you may participate with a World Final's team in spring.)

Input

The input consists of multiple test cases. Continue to process test cases until end of file.

Each test case consists of:

- one line with two integers n ($2 \le n \le 14$) and L ($1 \le L \le 10^{15}$), the number of control points and the desired length of the race, respectively;
- n lines with n integers each. The jth integer on the ith line, d_{ij}, denotes the distance between control point i and j ($1 \le d_{ij} \le L$ for $i \ne j$, and $d_{ii} = 0$). For all $1 \le i, j, k \le N$ it is the case that $d_{ij} = d_{ji}$ and $d_{ij} \le d_{ik} + d_{kj}$.

Output

Output one line with "`possible`" if it is possible to visit all control points once in some order and directly return to the first one such that the total distance is exactly L, and "`impossible`" otherwise.

Sample Input

```
4 10
0 3 2 1
3 0 1 3
2 1 0 2
1 3 2 0
3 5
0 1 2
1 0 3
2 3 0
```

Sample Output

```
possible
impossible
```