Problem A. Average Problem

Source file name:	Average.c, Average.cpp, Average.java, Average.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

At the Institute of Computing & Programming College (ICPC), grades are given from 0 to 100, but in multiples of five, it means that if you get a 97% as an average, then your final grade would be 95. Just the same case happens when you get a 93%, you get a 95. This happens because 95 is in both cases, the closest multiple of five to the obtained grade. This means, for example that 67.5 rounds to 65 but 67.51 rounds to 70.

Now, there are many evaluation tables assigned to different courses. And the teachers want to know the final grade of each one of their students and if the final grade went `UP`, if it went `DOWN` or if it stays the `SAME` from the original obtained grade. An evaluation table is a list of points that each assignment has. A student get an individual grade on each assignment from 1 to 100.

For example, if the evaluation table is 50 and 30, and a student gets 100 and a 50, it means that he got 100% of the 50 points and a 50% of the 30 points, then his grade will be $\frac{65}{80}$.

Input

The first line has the positive integer number T of test cases $1 \leq T \leq 100$. Each test case begin with two integers A, S ($1 \leq A \leq 12$, $1 \leq S \leq 40$) where A is the number of evaluations on the curse, and S is the number of students. The next line has A positive integer numbers $1 \leq a_i \leq 100$ representing the value of each evaluation. The next S lines are the grades of each of the students. Every line has A positive integers, where s_i is a percentage obtained in a_i evaluation.

Output

Per each student you must print a single line with the final grade and if it went `UP`, `DOWN` or remain the `SAME` as it was. Each test case must be ended with a blank line.

Example

Input	Output
4 2 3 50 50 100 90 100 91 100 89 4 1 15 15 15 15 70 80 75 75 5 1 20 30 10 20 20 100 100 0 0 100 5 1 20 30 10 20 20 100 100 10 10 100	95 SAME 95 DOWN 95 UP 75 SAME 70 SAME 75 UP

Problem B. Breaking Vacations

Source file name:	Breaking.c, Breaking.cpp, Breaking.java, Breaking.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Miguel has just started his vacations, but there is an issue, he is called to present a final test just after vacations! So he has to split his time into studying for the test and having fun on his activities.

He needs to get at least K points on his exam. And he knows how much to study every day in order to assure a percentage of his grade. Miguel has planned every day of his vacation in three different activities: studying, playing video games and playing sports. The first gives certain points of his grade and the second two reports him different amount of happiness. But he can just make one activity per day.

Miguel wants to know the maximum amount of happiness he can earn during his vacation after having assured at least K points on his final grade.

Input

The input has a first line with a positive integer T ($1 \leq T \leq 20$) that indicates the number of test cases. Each test case consists in the first line are two integers N being the number of days in Miguel's vacations and K the amount of points he needs to score in his final exam ($1 \leq N \leq 10^4$, Miguel can get really long vacations) $0 \leq K \leq 10^2$, The next N lines have three integer numbers separated by space, S denoting the points Miguel estimates he can get for studying for his test, V the happiness he can get by playing video games and P the amount of happiness he can get by playing sports. Where $1 \leq S, V, P \leq 10^3$.

Output

For each test case, print in a single line, the maximum amount of happiness Miguel can earn during his vacation, after being sure he have studied at least K points. If there is no way to guarantee at least K, print `NO SOLUTION`

Example

Input	Output
3 3 1 90 1 500 1 2 3 40 100 40 3 92 90 1 500 1 2 3 40 100 40 3 40 9 1 500 1 2 3 4 100 40	600 3 NO SOLUTION

Problem C. Cardinality of Sets

Source file name:	Cardinality.c, Cardinality.cpp, Cardinality.java, Cardinality.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

There are two sets of positive integers A and B with $|A| = m$ and $|B| = n$ (Remember that the cardinality of a set is the number of different elements it contains, the cardinality is indicated by the bars that enclose the set).

We want to determine the cardinalities of the sets that are obtained as a result $A - B$, $A \cap B$, $B - A$ and $A \cup B$.

Input

The input contains multiple test cases, each one has the following structure:

The first line contains two positive integers m n ($1 \leq m,\ n \leq 10^6$), which represent the cardinalities of the sets A and B respectively.

The second line contains m **different** positive integers (separated by a blank space) which make up the set A, these values are in the closed interval $[1,\ 10^9]$.

The third line contains n **different** positive integers (separated by a blank space) which make up the set B, these values are in the closed interval $[1,\ 10^9]$.

The input ends with a test case that has two zeros which should not be processed.

Output

For each test case, the output must contain a single line with four non-negative integers, separated by a single space, which respectively represent $|A - B|$ $|A \cap B|$ $|B - A|$ $|A \cup B|$.

Example

Input	Output
5 6 7 1 4 3 9 8 3 2 10 7 1 5 5 1 3 5 7 9 10 8 6 4 2 8 6 10 1 5 3 6 4 7 2 1 2 3 4 5 6 0 0	2 3 3 8 5 0 5 10 2 6 0 8

Problem D. Determining Fibonaccism

Source file name:	Determining.c, Determining.cpp, Determining.java, Determining.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Close to the year 1200, Leonardo Pisano Bigollo Fibonacci, invented his famous recursion:

$$fibonacci(n) = \begin{cases} 0 & \text{if} \quad n = 0 \\ 1 & \text{if} \quad n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{if} \quad n \geq 2,\ n \in \mathbb{N} \end{cases}$$

Fernando and Té are playing a game, Fernando says a non-negative integer number n $(0 \leq n < 2^{31} - 1)$ and Té has to say if exists a non-negative integer number m $(0 \leq m \leq 10^4))$ such as $fibonacci(m) \equiv n \pmod{(2^{31} - 1)}$ in other words, if the fibonacci of m is equivalent to $n \mod 2^{31} - 1$.

Your task is to help Té in order to beat Fernando at this game.

Input

The first line of the input is the number of test cases that Fernando is going to throw to Té, T $(1 \leq T \leq 50000)$. The next T lines contain a single non-negative integer number N $(0 \leq N < 2^{31})$ that is the number Fernando is asking Té.

Output

For each number Fernando asks, print in a single line `YES` if there is a fibonacci number that matches the description above, or `NO` if there is not.

Example

Input	Output
5 1 2 3 4 2144909037	YES YES YES NO YES

Explanation

$fibonacci(55) = 139583862445$ and $139583862445 \equiv 2144909037 \pmod{(2^{31} - 1)}$ therefore, 2144909037 is a valid number because exists 55 and $0 \leq 55 \leq 10000$.

Problem E. Exceeding the Goal

Source file name:	Exceeding.c, Exceeding.cpp, Exceeding.java, Exceeding.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

A cruel math professor left this homework for his students. Given three numbers, K, S and G, indicate which is the minimum amount of adding up numbers from (and including) S are needed to exceed G if every adding up number is greater than the previous one by exactly K?

For example, for $K = 1$, $S = 10$, $G = 100$ the answer is 8, since $10+11+12+13+14+15+16+17 > 100$. Notice that if $K = 5$, $S = 4$ and $G = 20$ the answer would be 3, since $4 + 9 + 14 > 20$.

The professor in charge of this exercise likes his students to lack leisure time, so he has given a lot of triads of numbers to verify. A desperate student has reached you in order to program and solve his problem.

Input

The first line of the input has a positive integer number T ($1 \leq T \leq 3 \cdot 10^5$) which is the number of test cases. Each one of the next T lines have three positive integer numbers separated by space, K, S and G, it is guaranteed that $S < G$ and $1 \leq K, S, G \leq 10^9$.

Output

For each test case, print on a single line the minimum amount of adding up numbers should be printed required to fulfil the description above.

Example

Input	Output
4 1 3 10 6 3 10 50 1 51 1 1 6	3 2 2 4

Problem F. Farthest Cell in a Maze

Source file name:	Farthest.c, Farthest.cpp, Farthest.java, Farthest.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

One of the passions of professor *Humbertov Moralov* is solving mazes. Now he is interested in identifying the farthest cell(s) from an initial starting point by making an optimal tour where the number of cells he has to visit is minimized. To make the challenge even more interesting, Professor Moralov wants the optimal route from the initial cell to each of the furthest cells to be independently written.

For ease, you can think of the maze as a grid (a matrix) of cells. A cell can be either a wall or a corridor (that means, a cell that you can walk through). At any time the professor *Moralov* can move to a new cell from their current position if they share one side and both are corridor cells.

As you are a student of the Data Structure course, the professor *Humbertov Moralov* needs your help to solve this challenge, for which you must write a program to calculate the minimum distances from the initial cell (professor's starting point) to each of the other corridor cells, then your program must identify the farthest corridor cell(s) in the minimum distances and independently write the route that must be made from the initial cell to each of the most distant cells.

Input

The input begins with a positive integer T ($1 \leq T \leq 10$), denoting the number of test cases.

Each test case begins with a line containing two positive integers W H (Wide, High, $3 \leq W, H \leq 1000$). The test case continues with H lines, each containing W characters. Each character represents the status of a cell in the maze as follows:

1. '.' - to indicate that it is a corridor cell (cell that you can walk through),
2. '#' - to indicate that it is a wall cell,
3. '@' - to indicate professor Humbertov Moralov's starting point (appears exactly once per test case).

Output

For each test case, print a first line in the following format `Case idCase:`, where `idCase` is replaced by the number of the test case in which it is found, then a second line is printed containing two positive integers F L, which respectively represent the total of cells furthest in the optimal path and the length in the optimal route. Finally, the lexicographically ordered F optimal runs of length L are presented, each one on one line. Each optimal path is a word of length L that can consist only of the uppercase letters '`D`' (Down), '`L`' (Left), '`R`' (Right) y '`U`' (Up). Since there can be multiple optimal paths to get from the professor *Moralov's* starting position to one of the farthest cells, then choose the optimal path that is first in the lexicographic order. For more clarity on the input and output format see the examples below.

Example

Input	Output
5 6 5 ...#.# .##... ..@.## #.#... ...#.# 6 5 .###.# .##... ..@.## #.#... ...#.# 5 6 ##### #..#. #.### #...# ###@# ..... 7 7 ..#.#.. ..#.#.. ###.### ...@... ###.### ..#...# ..#.#.. 13 13 ######.###### #####...##### ####.....#### ###.......### ##.........## #...........# ......@...... #...........# ##.........## ###.......### ####.....#### #####...##### ######.######	Case 1: 1 6 LLUURR Case 2: 7 4 LDDL LDDR LLUU RDRD RDRR RURR RURU Case 3: 1 6 ULLUUR Case 4: 1 6 DDRRDR Case 5: 24 6 DDDDDD DDDDDL DDDDDR DDDDLL DDDDRR DDDLLL DDDRRR DDLLLL DDRRRR DLLLLL DRRRRR LLLLLL LLLLLU LLLLUU LLLUUU LLUUUU LUUUUU RRRRRR RRRRRU RRRRUU RRRUUU RRUUUU RUUUUU UUUUUU

Problem G. Geometric Sequence

Source file name:	Geometric.c, Geometric.cpp, Geometric.java, Geometric.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

In the past days, the professor *Humbertov Moralov* came across the following sequence while reviewing his class notes from the Discrete Mathematics course:

$S(n)$	3	5	7	9	25	49	27	125	343	81	625	2401	$\cdots$
n	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$

which for clarity can be seen as:

$S(n)$	3	5	7	3^2	5^2	7^2	3^3	5^3	7^3	3^4	5^4	7^4	$\cdots$
n	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$

for which he raised the following recursive function to calculate the n-th term of the sequence:

$$S(n) = \begin{cases} 3 & \text{if} \quad n = 0 \\ 5 & \text{if} \quad n = 1 \\ 7 & \text{if} \quad n = 2 \\ S(n \ mod \ 3) \cdot S(n-3) & \text{if} \quad n \geq 3, \ n \in \mathbb{N} \end{cases}$$

Professor Moralov feels very proud of his solution, but he recognizes that is computationally expensive and that he can only do step-by-step on the board for small n values. Actually, professor Humbertov Moralov should come up with a more efficient solution, but he's sure he won't do that job!

Now we propose to you a generalization of the sequence where we have k initial terms, from which all the k terms alternate with the same power incrementally, as presented below:

$S(n)$	t_1	t_2	$\cdots$	t_k	$(t_1)^2$	$(t_2)^2$	$\cdots$	$(t_k)^2$	$(t_1)^3$	$(t_2)^3$	$\cdots$	$(t_k)^3$	$\cdots$
n	0	1	$\cdots$	$k-1$	k	$k+1$	$\cdots$	$2k-1$	$2k$	$2k+1$	$\cdots$	$3k-1$	$\cdots$

For this sequence you must propose a formula to generate the n-th term. Obviously, your solution has to be computationally more efficient than the solution that Professor Moralov has proposed.

Input

The input begins with a positive integer t ($1 \leq t \leq 2 \cdot 10^3$), which represents the total number of test cases. Then the first line of each test case contains two positive integers k and q ($2 \leq k \leq 10^2, \ \ 1 \leq q \leq 10^3$) which respectively represent the total of initial terms of the sequence and the total of queries (positions that must be evaluated in the sequence). The second line of each test case contains k space-separated integers in the range $[-10^3, \ 10^3]$. he test case ends with q lines, each containing a non-negative integer in the range $[0, \ 10^{18}]$.

Output

The output must contain q lines, each one containing the term of the sequence requested. As the terms of the sequence can become very large integers, then they must be presented modulo 10^9.

Note: Remember, the results of the modulo operation are strictly non-negative integers less than m.

Example

Input	Output
2 3 5 3 5 7 0 5 8 4 10 4 3 7 0 -2 2 18 9 16	3 49 343 25 625 999999968 0 16807

Problem H. Hidden Professions

Source file name:	Hidden.c, Hidden.cpp, Hidden.java, Hidden.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

The Institute for Common Professions Characteristics (ICPC) is setting a way to determine how many professions a person has in a resume. But there are some professions that can be named different, for example, there are *doctors* that may write *dr* instead of the full word.

There is another problem, in some resumes, people like to exaggerate the number of professions they may have by placing a lot of abbreviations for the same career. And since there is not only one way to do it, anyone can put their careers in the order they prefer. There are even some people who put a profession that are not registered or they may even repeat their professions with a different abbreviation!

For all the careers that ICPC has registered, any prefix of a profession may be used as a reference for the profession if there is not ambiguity, for example, "*doc*" may be used as a valid abbreviation for "*doctor*". An ambiguity appears when two different professions share a prefix, then that prefix is not valid as an abbreviation for a profession. For example, if you have the professions "*doctor*" and "*dentist*", "*do*" is a valid abbreviation, but "*d*" is not, since it is a shared prefix with "*dentist*"

Your task is to write a program that determines the number of different registered professions a person has on his or her resume, given the list of professions that are considered to be equivalent and the list of professions in their resumes.

Input

The first line is a positive integer T, the number of test cases $1 \leq T \leq 200$.
Each test case begins with two integers separated by space, P ($1 \leq P \leq 30$) and R ($1 \leq R \leq 1000$), that stands for the number of professions and the number of resumes, respectively.

The next $2 \times P$ lines contain the description of the known professions. Each description of a career consist of two lines: The first line has an integer C ($1 \leq C \leq 40$) which stands for the number of known names of that career. The second line has C strings separated by space, the maximum length of this strings is 40 and the minimum is 1.

The next $2 \times R$ lines contain the description of the resumes. Each resume is also described as two lines. The first line has an integer N ($1 \leq N \leq 30$) that are the number of strings in the next line. The second line has N strings representing the professions that person has, separated by space, they might or not be registered. The maximum length of each string is 40 and the minimum is 1.

In every case the strings are composed only by lower case English letters. Is is guaranteed that there are no 2 different careers with the exact same name.

Output

For each resume, print the number of different valid professions' prefixes registered that appear on the resume

Example

Input	Output
1 3 4 2 dentist dntist 2 inga ingena 3 doctor phd dr 3 salmon sin dr 4 inga ingen ingena ing 2 ph ing 2 d perm	1 1 2 0

Problem I. Imaginary Numbers

Source file name:	Imaginary.c, Imaginary.cpp, Imaginary.java, Imaginary.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Probably the most beautiful math expression ever written is *Euler's Identity*, that is usually written as

$$e^{i\pi} + 1 = 0$$

It's beauty consists in that expression is that there are combined two irrational numbers (e and π), one natural number (1) and an imaginary number, named i. It is called imaginary because it is not a *real* number, i stands for $\sqrt{-1}$, so $i^2 = -1$.

Given two numbers a and b, your task is to determine the result of this expression: $a \times i^b$. For example $30 \times i = 30i$ and $20 \times i^2 = -20$.

Input

The input consists in several test cases, one per line, each test case has two integers separated by a single space, a ($-10^{18} \leq a \leq 10^{18}$) and b ($0 \leq b \leq 10^{18}$). The input stops with a line with $a = 0$ and $b = 0$ that should not be answered

Output

For each test case, you should print the result of $a \times i^b$ as shown in the examples.

Example

Input	Output
1 1 2 2 3 3 4 4 -3 2 1000000007 80 0 0	1i -2 -3i 4 3 1000000007

Problem J. Johann's Function

Source file name:	Johann.c, Johann.cpp, Johann.java, Johann.py
Input:	Standard
Output:	Standard
Author(s):	Hugo Humberto Morales Peña - RPC & UTP Colombia

Typically in a first-year programming course in an engineering or computer science academic program, students are taught to build functions that make use of the doubly nested loop structure such as the following:

```
unsigned long long JohannsFunction(int n)
{
    int i, j;
    unsigned long long result = 0;

    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= i; j++)
        {
            result += j;
        }
    }

    return result;
}
```

The running-time function of the previous algorithm belongs to $O(n^2)$, with a value of $n = 10^6$ the total number of operations performed by the algorithm, in order to give the result would require 10^{12} steps. As a competitor of the different phases of the ICPC, you know that a solution that performs that amount of steps, will obtain a verdict of **Time Limit Exceeded** in competition, for this reason you are asked to propose a solution that has a running time as close to $O(1)$ as possible, regardless of the size of n, in which you must make use of all your experience as a competitor of ICPC!

Input

The input begins with a positive integer t ($1 \leq t \leq 10^6$), which represents the total number of test cases. Then t lines are presented, each one containing a positive integer n ($1 \leq n \leq 10^6$) for which the result of the **Johann's Function** must be calculated.

Output

The output must contain t lines, each containing a long positive integer as a result of the **Johann's Function**.

Example

Input	Output
7 1 10 100 1000 10000 100000 1000000	1 220 171700 167167000 166716670000 166671666700000 166667166667000000

Problem K. K-Uniform Array

Source file name:	Kuniform.c, Kuniform.cpp, Kuniform.java, Kuniform.py
Input:	Standard
Output:	Standard
Author(s):	José Manuel Salazar Meza - UFPS Colombia

You are given an array a consisting of n positive integers.

The array a is called *k-Uniform* if it has some sub-array of length k whose elements are all equal (recall that a sub-array of a of length k is a contiguous part of a containing exactly k elements).

You must make the array a *k-Uniform*. To achieve it, you can apply a sequence of operations zero or more times. In one operation, you can choose an index i and remove from a all elements equal to a_i (including a_i).

Your task is to find the minimum number of such operations that you must perform to make the array a *k-Uniform*.

Input

The first line contains two positive integers n ($1 \leq n \leq 10^5$) and k ($1 \leq k \leq n$) — the size of the array and the *Uniform* value respectively.

The second line contains n positive integers a_i ($1 \leq a_i \leq 10^9$) — the numbers in the array.

Output

If it is not possible to make the array a *k-Uniform* print -1. Otherwise, print the minimum number of operations that you must perform to achieve it.

Example

Input	Output
10 3 1 2 3 1 2 4 1 2 4 4	2
8 3 5 5 6 6 5 5 6 6	1
3 1 7 8 9	0

Problem L. Lighting System

Source file name:	Lighting.c, Lighting.cpp, Lighting.java, Lighting.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

Currently, the university campus has a system of lamps which can be turned on manually or with activation by a photocell sensor when a lamp adjacent to it is turned on (that is, the lamp turns on because in the reading range of the photocell sensor a lamp was turned on), this is a symmetric relationship. *Don Leo*, who is the administrator of the university campus lighting system, wants to determine the minimum number of lamps that he has to turn on manually, thereby guaranteeing that all the lamps on campus are turned on, additionally, the You also want to determine the number of lamps that are turned on from each of the lamps that are turned on manually.

Input

The input consists of multiple test cases. Your program must process all of these. The first line of each test case contains two non-negative integers n and p ($1 \leq n \leq 10^4$, $0 \leq p \leq n$), where n represents the number of lamps on the university campus and p represents the number of pairs of lamps that are adjacent (that is, that are at a distance less than or equal to the range in which one of the two photocell sensors in the lamps are activated by turning on the other). The next p lines contain two positive integer values a and b ($1 \leq a, b \leq n$, $a \neq b$), which indicate that the lamps a and b are adjacent. Entry ends with end of file (EOF).

Output

For each test case print k lines, where k represents the minimum number of lamps that must be turned on manually by Don Leo, in each of these lines two positive integer values must be printed c and t ($1 \leq c, t \leq n$), where c is the position of the lamp and t is the total number of lamps that are lit after manual lighting of the c lamp. Additionally, the k lines must be ordered in ascending order with respect to the positions of the lamps that are activated manually.

Don Leo is a very superstitious person, for this reason, if there is a group of lamps that can be turned on, then you must choose to manually turn on the lamp with the smallest position.

Example

Input	Output
7 9 1 2 1 3 1 4 2 3 2 4 3 4 5 6 5 7 6 7 11 8 1 2 2 3 3 4 5 6 6 8 7 8 8 9 10 11 6 0	1 4 5 3 1 4 5 5 10 2 1 1 2 1 3 1 4 1 5 1 6 1

Problem M. Mandatory by Summations

Source file name:	Mandatory.c, Mandatory.cpp, Mandatory.java, Mandatory.py
Input:	Standard
Output:	Standard
Author(s):	Hugo Humberto Morales Peña - RPC & UTP Colombia

Typically in a first-year programming course in an engineering or computer science academic program, students are taught to build functions that make use of the doubly nested loop structure such as the following:

```
unsigned long long JohannsFunction(int n)
{
    int i, j;
    unsigned long long result = 0;

    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= i; j++)
        {
            result += j;
        }
    }

    return result;
}
```

The running-time function of the previous algorithm belongs to $O(n^2)$, with a value of $n = 10^6$ the total number of operations performed by the algorithm, in order to give the result would require 10^{12} steps. As a competitor of the different phases of the ICPC, you know that a solution that performs that amount of steps, will obtain a verdict of **Time Limit Exceeded** in competition, for this reason you are asked to propose a solution that has a running time as close to $O(1)$ as possible, regardless of the size of n, in which you must make use of all your experience as a competitor of ICPC!

Input

The input begins with a positive integer t ($1 \leq t \leq 10^6$), which represents the total number of test cases. Then t lines are presented, each containing a long positive integer n ($1 \leq n \leq 10^{18}$) for which the result of the **Johann's Function** must be calculated.

Output

The output must contain t lines, each containing a positive integer as a result of the **Johann's Function**. Since the result can be a very large value, then display the result modulo $2^{31} - 1$.

Example

Input	Output
7 1 10 100 1000 10000 100000 1000000	1 220 171700 167167000 1360429181 1165889036 898139791

Problem N. Neither Divide Nor Use Double

Source file name:	Neither.c, Neither.cpp, Neither.java, Neither.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Milton Jesús Vera Contreras - RPC & UFPS Colombia`

Carlos is a student who use several division and double data type in his computer programs. His professor has warned that division results in precision errors, for example when the division is not exact and a repeating decimal number is generated. But Carlos insists on using division and double data type. Then his professor created this simple problem and challenged Carlos to solve it.

Input

The input consists of multiple test cases. Each test case consists of two integers a and n ($1 \leq a \leq 100, \quad 0 \leq n \leq 10^4$).

Output

For each test case two integer m and x must be printed if the division $\frac{1}{a^n}$ is exact and m is the smallest integer that satisfies this equality (in the set of integers):

$$x = \frac{1}{a^n} \cdot 10^m$$

If the division $\frac{1}{a^n}$ is inexact then print: `Precision Error`

Example

Input	Output
1 9999	0 1
2 6	6 15625
5 11	11 2048
10 15	15 1
80 15	60 28421709430404007434844970703125
7 19	Precision Error
99 23	Precision Error

Explanation

A division is exact when the remainder is zero. The dividend equals the divisor multiplied by the quotient. The quotient is an integer number or a floating point number no periodic, for example: $\left(\frac{1}{16} = 0.0625\right)$, $\left(\frac{1}{125} = 0.008\right)$.

A division is inexact when there is a leftover remainder. The dividend equals the divisor multiplied by the quotient plus the remainder. The quotient is a floating point number periodic, for example: $\left(\frac{1}{7} = 0,\underbrace{142857}\underbrace{142857}\underbrace{142857}\ldots\right)$ with period (repeating decimal) 142857, $\left(\frac{1}{14} = 0,0\underbrace{714285}\underbrace{714285}\underbrace{714285}\ldots\right)$ with period (repeating decimal) 714285.

Problem O. One Piece of Cake

Source file name:	One.c, One.cpp, One.java, One.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Rodrigo Chaves - TEC & UCR Costa Rica`

The Straw Hat Pirates are a very skilled crew that travels the seas. It consists of its captain and 9 other members. They were invited to a wedding by a famous pirate that has many children and loves to eat cake.

The crew has just arrived to the dinner party. At the start of the party there were P pieces of cake. Every crew member wants exactly one piece of cake. However, other people in the party have already eaten E pieces.

Please help them know if there is enough cake for them to eat!

Input

The first line consists of a single integer T ($1 \leq T \leq 20000$), the number of test cases. After that, T lines follow, which have two space-separated integers P and E, ($1 \leq E \leq P \leq 200$).

Output

For each case, print a single line consisting of '`YES`' (without quotes) if there is enough cake for the Straw Hat Pirates and '`NO`' (without quotes) otherwise.

Example

Input	Output
3 99 80 45 40 15 4	YES NO YES

Problem P. Professor Sabio and the Easy Homework

Source file name:	Professor.c, Professor.cpp, Professor.java, Professor.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Wilmer Emiro Castrillón Calderón - UNIAMAZONIA Colombia`

The *Professor Elio Sabio* has prepared for his students some interesting exercises, for each one he did the following process: in a sheet its drawn initially a matrix M of 3×3 size with non-negative integer numbers and lower to 100, these numbers are written carefully in such way that $M_{i,3} = M_{i,1} + M_{i,2}$ and $M_{3,j} = M_{1,j} - M_{2,j}$, then for each digit f each number M it is replaced with a letter of the alphabet (the digits of the same value are replaced by the same letter), and it is verified that in total 6 different letters are used (the first 6 of the alphabet), for example:

$$\begin{array}{ccc} 66 + 32 = 98 & & AA + BC = DE \\ - \quad - \quad - & & - \quad - \quad - \\ 43 + 6 = 49 & \Rightarrow & FB + A = FD \\ = \quad = \quad = & & = \quad = \quad = \\ 23 + 26 = 49 & & CB + CA = FD \end{array}$$

Now the *Professor Elio Sabio* has given to each student a matrix of letters and he leaves the homework of finding the initial matrix, if there are multiple options so, find all of them. However, many of his students have difficulties to solve it and they ask your help with their homework, can you help them?

Input

The input starts with a positive integer number T ($1 \leq T \leq 200$) the quantity of test cases, each case has three lines, each one has three strings of 1 or 2 characters describing the matrix of letters, it is ensured that for each matrix there are just letters in the rank A-F and each letter appears at least once. For each pair of consecutive test cases follows a line in white.

Output

For each test case print the number of cases, then print each solution in a different line with the corresponding values of A, B, C, D, E and F in this order and separated by a blank space, the answers must appear in lexicographical order, print a line in white after each test case (look the examples of output).

Example

Input	Output
2 AA BC DE FB A FD CB CA FD A BC BD A ED EB F B B	Case #1: 6 3 2 9 8 4 Case #2: 2 8 4 6 7 0 3 7 1 4 6 0

Problem Q. Queries on the Stack

Source file name:	Queries.c, Queries.cpp, Queries.java, Queries.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Yonny Mondelo Hernández - UCI Cuba`

You have an empty stack and you are given some queries. These queries are the basic stack operations such as Push, Pop and finding the Top element on the stack. Also, is needed to know in any moment the absolute difference between the maximum and the minimum value on the stack. You should process the given queries.

Input

The first line contain a integer T $(1 \leq T \leq 100)$ denoting the number of cases. Each case will begin with a line containing a integer Q $\left(1 \leq Q \leq 10^4\right)$ denoting the number of queries to process, followed by exactly Q lines based on these formats:

- 1 V: Push V $\left(0 \leq V \leq 10^4\right)$ on the Top of the stack.
- 2: Pop an element from the Top of the stack. If the stack is empty, do nothing.
- 3: Print the absolute difference between the maximum and the minimum value on the stack (see Output Format). If the stack is empty, show "`Empty!`" without quotes.

Output

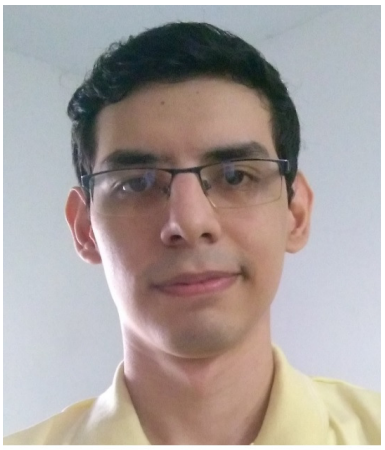
For each query of type 3 find and show the absolute difference between the maximum and the minimum value on the stack.

Example

Input	Output
2 12 2 1 10 1 15 3 2 2 3 1 18 3 1 10 1 1 3 3 3 1 999 3	5 Empty! 0 17 Empty! 0

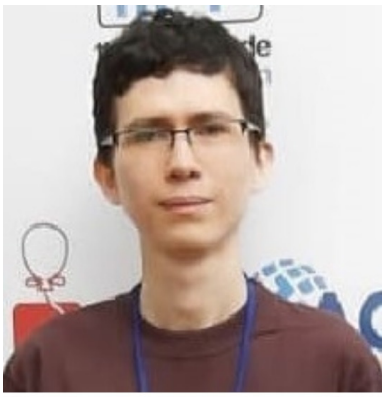
Problemsetters:

Prof. Hugo Humberto Morales Peña
Universidad Tecnológica de Pereira
Colombia

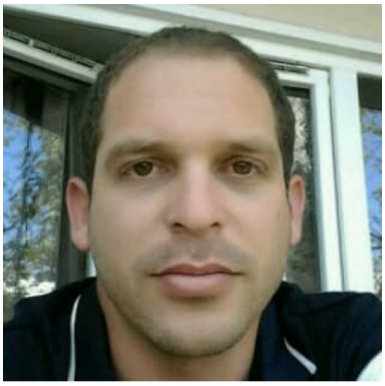


Est. José Manuel Salazar Meza
Universidad Francisco de Paula Santander
Colombia

Prof. Milton Jesús Vera Contreras
Universidad Francisco de Paula Santander
Colombia



Est. Wilmer Emiro Castrillón Calderón
Universidad de la Amazonia
Colombia



Prof. Yonny Mondelo Hernández
Universidad de las Ciencias Informáticas
Cuba

Problemsetters and problemtesters:

Prof. Eddy Ramírez Jiménez
Universidad Nacional de Costa Rica
& Técnológico de Costa Rica

Ing. Rodrigo Chaves
Técnológico de Costa Rica
& Universidad de Costa Rica

Problem A. Average Problem

Source file name:	Average.c, Average.cpp, Average.java, Average.py
Input:	Standard
Output:	Standard
Author(s):	Eddy Ramírez Jiménez - UNA & TEC Costa Rica

En el Instituto de Computación y Programación Colegial ICPC, las notas son dadas de 0 a 100, pero en múltiplos de cinco. Esto quiere decir que si usted obtiene un 97% como promedio, su nota final sería de 95. Lo mismo ocurre cuando obtienes un 93%, se obtiene un 95. Esto ocurre porque 95 es en ambos casos, el múltiplo de 5 más cercano a la nota obtenida. Esto quiere decir, por ejemplo que 67.5 redondea 65, pero 67.51 redondea a 70.

Ahora, se tienen muchas tablas de evaluación asignadas a los diferentes cursos. Los profesores quieren saber la nota final de cada estudiante y saber si la nota obtenida fue arriba (`UP`), abajo (`DOWN`) o se quedó igual (`SAME`) con respecto a la nota original. Una tabla de evaluación consiste en una lista de puntos de los cuales constan las evaluaciones de los cursos. Un estudiante tiene un porcentaje para cada uno de esos puntos.

Por ejemplo, si la tabla de evaluación es 50 y 30 y un estudiante obtiene 100 y 50, significa que obtuvo el 100% de los 50 puntos y el 50% de los 30 puntos, entonces si nota sería $\frac{65}{80}$.

Input

La primera línea tiene un número entero positivo T, que es el número de casos de prueba $1 \leq T \leq 100$. Cada caso de prueba comienza con dos números enteros positivos A, S. $1 \leq A \leq 12$, $1 \leq S \leq 40$, donde A es el número de evaluaciones que se realizaron en el curso y S es la cantidad de estudiantes en el curso. La siguiente línea tiene A números enteros positivos, $1 \leq a_i \leq 100$ representando el valor de cada evaluación. Las siguientes S líneas son las notas de cada estudiante. Cada línea contiene A enteros, donde s_i es el porcentaje de aprovechamiento de la a_i evaluación.

Output

Por cada estudiante, debe imprimir una sola línea con la nota final y si subió (`UP`), bajó (`DOWN`) o se quedó igual (`SAME`) con respecto a la nota original. Cada caso de prueba debe terminar con una línea en blanco.

Example

Input	Output
4 2 3 50 50 100 90 100 91 100 89 4 1 15 15 15 15 70 80 75 75 5 1 20 30 10 20 20 100 100 0 0 100 5 1 20 30 10 20 20 100 100 10 10 100	95 SAME 95 DOWN 95 UP 75 SAME 70 SAME 75 UP

Problem B. Breaking Vacations

Source file name:	Breaking.c, Breaking.cpp, Breaking.java, Breaking.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Miguel ha iniciado sus vacaciones, pero tiene un problema, lo han convocado a presentar un examen final justo al final de sus vacaciones. Entonces, él debe dividir su tiempo en estudiar para el examen y en divertirse en sus actividades.

Él sabe que necesita obtener al menos K puntos en su examen. También sabe cuánto estudiar cada día para poder asegurar cierto porcentaje de su nota. Miguel ha planeado todos los días de sus vacaciones en tres actividades diferentes: estudiar, jugar video juegos y practicar deportes. El primero le da cierta cantidad de puntos a su nota, los segundos, le dan cierto valor de felicidad. Pero él sólo puede hacer una actividad por día.

Miguel quiere saber el máximo valor posible de felicidad que puede acumular durante sus vacaciones tras haber asegurado al menos k puntos en su examen final.

Input

La entrada consiste en una primera línea con un entero T ($1 \leq T \leq 20$) el cual indica el número de casos de prueba. Cada caso de prueba consiste en una primera línea con dos enteros, N el número de días en las vacaciones de Miguel y K la cantidad de puntos que necesita acumular en su examen final ($1 \leq N \leq 10^4$, Miguel puede tener vacaciones muy largas $0 \leq K \leq 10^2$). Las siguientes N líneas tienen tres enteros cada una, separados por un espacio, S la cantidad de puntos que Miguel estima que puede obtener por estudiar ese día para su examen, V la felicidad que él puede obtener de jugar videojuegos y P la cantidad de felicidad que él puede obtener por practicar deportes. Donde $1 \leq S, V, P \leq 10^3$.

Output

Para cada caso de prueba, imprimir una sola línea con la máxima cantidad de felicidad que Miguel puede acumular durante sus vacaciones, luego de haber asegurado al menos K puntos. Si no hay manera de garantizar al menos esa cantidad, entonces imprimir `NO SOLUTION` que significa que no hay solución en inglés.

Example

Input	Output
3 3 1 90 1 500 1 2 3 40 100 40 3 92 90 1 500 1 2 3 40 100 40 3 40 9 1 500 1 2 3 4 100 40	600 3 NO SOLUTION

Problem C. Cardinality of Sets

Source file name:	Cardinality.c, Cardinality.cpp, Cardinality.java, Cardinality.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

Se tienen dos conjuntos de números enteros positivos A y B con $|A| = m$ y $|B| = n$ (recordar que la cardinalidad de un conjunto es la cantidad de elementos diferentes que contiene, la cardinalidad se indica con las barras que encierran al conjunto).

Se quiere determinar las cardinalidades de los conjuntos que se obtienen como resultado de $A - B$, $A \cap B$, $B - A$ y $A \cup B$.

Input

La entrada contiene múltiples casos de prueba, cada uno de ellos como se describe a continuación.

La primera línea contiene dos números enteros positivos m n ($1 \leq m,\ n \leq 10^6$), los cuales representan las cardinalidades de los conjuntos A y B respectivamente.

La segunda línea contiene m números enteros positivos **diferentes** (separados por un espacio en blanco) los cuales conforman el conjunto A, dichos valores están en el intervalo cerrado $[1,\ 10^9]$.

La tercera línea contiene n números enteros positivos **diferentes** (separados por un espacio en blanco) los cuales conforman el conjunto B, dichos valores están en el intervalo cerrado $[1,\ 10^9]$.

La entrada finaliza con un caso de prueba que tiene dos ceros el cual no debe ser procesado.

Output

Para cada caso de prueba, la salida debe contener una sola línea con cuatro números enteros no negativos, separados por un solo espacio los cuales representan respectivamente $|A - B|$ $|A \cap B|$ $|B - A|$ $|A \cup B|$.

Example

Input	Output
5 6 7 1 4 3 9 8 3 2 10 7 1 5 5 1 3 5 7 9 10 8 6 4 2 8 6 10 1 5 3 6 4 7 2 1 2 3 4 5 6 0 0	2 3 3 8 5 0 5 10 2 6 0 8

Problem D. Determining Fibonaccism

Source file name:	Determining.c, Determining.cpp, Determining.java, Determining.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Cerca del año 1200, Leonardo Pisano Bigollo Fibonacci, inventó su famosa recursión:

$$fibonacci(n) = \begin{cases} 0 & \text{si} \quad n = 0 \\ 1 & \text{si} \quad n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & \text{si} \quad n \geq 2,\ n \in \mathbb{N} \end{cases}$$

Fernando y Té están jugando un juego, Fernando dice un número entero no negativo n $(0 \leq n < 2^{31} - 1)$ y Té debe decir sí existe un número entero no negativo m $(0 \leq m \leq 10^4))$ tal que $fibonacci(m) \equiv n \pmod{(2^{31} - 1)}$ en otras palabras, si el fibonacci de m es equivalente a $n \mod 2^{31} - 1$.

Su tarea es ayudar a Té para que venza a Fernando en este juego.

Input

La primera línea de la entrada consiste en un número entero positivo T, de casos de prueba que Fernando le va a lanzar a Té, T $(1 \leq T \leq 50000)$. Las siguientes T líneas contienen un número entero no negativo N $(0 \leq N < 2^{31})$ que es el número que Fernando le está preguntando a Té.

Output

Para cada número que Fernando pregunte, debe imprimir una sola línea con `YES` (sí en inglés) si hay un número de fibonacci que cumpla lo descrito arriba o `NO` si no lo hay.

Example

Input	Output
5 1 2 3 4 2144909037	YES YES YES NO YES

Explanation

$fibonacci(55) = 139583862445$ y $139583862445 \equiv 2144909037 \pmod{(2^{31} - 1)}$ por lo tanto, 2144909037 es un número válido porque existe el 55 y $0 \leq 55 \leq 10000$.

Problem E. Exceeding the Goal

Source file name:	Exceeding.c, Exceeding.cpp, Exceeding.java, Exceeding.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Un profesor cruel de matemática dejó esta tarea a sus estudiantes. Dados tres números, K, S y G, indicar cuál es la mínima cantidad de sumandos necesaria desde (e incluyendo) a S, de modo que se exceda a G si cada sumando es mayor que el anterior exactamente por K.

Por ejemplo, para $K = 1$, $S = 10$, $G = 100$ la respuesta es 8, dado que $10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 > 100$. Notar que si $K = 5$, $S = 4$ y $G = 20$ la respuesta es 3, dado que $4 + 9 + 14 > 20$.

El profesor a cargo de este ejercicio le gusta que sus estudiantes no tengan tiempo libre, entonces les ha asignado muchas triadas para verificar. Un estudiante desesperado lo ha contactado para que programe y resuelva su problema.

Input

La primera línea de la entrada consiste en un número T ($1 \leq T \leq 3 \times 10^5$) que es el número de casos de prueba. Cada una de las siguientes T líneas tienen tres enteros separados por espacio, K, S y G está garantizado que $S < G$ y $1 \leq K, S, G \leq 10^9$

Output

Para cada caso de prueba, imprima en una sola línea el número mínimo de sumandos que se requiere para completar lo descrito arriba.

Example

Input	Output
4 1 3 10 6 3 10 50 1 51 1 1 6	3 2 2 4

Problem F. Farthest Cell in a Maze

Source file name:	Farthest.c, Farthest.cpp, Farthest.java, Farthest.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

Una de las pasiones del profesor *Humbertov Moralov* es resolver laberintos. Ahora está interesado en identificar la o las celdas más lejanas desde un punto de partida inicial haciendo un recorrido óptimo donde se minimice la cantidad de celdas que tenga que visitar. Para volver aun más interesante el reto el profesor Moralov quiere que se escriba de forma independiente el recorrido óptimo que se realiza desde la celda inicial a cada una de las celdas más lejanas.

Por facilidad, usted puede considerar que el laberinto es una cuadrícula (una matriz) de celdas. Una celda puede ser una pared o puede ser un pasillo (es decir, una celda por la cual se puede caminar). En cada momento el profesor *Moralov* puede moverse a una nueva celda desde su posición actual si ellas comparten un lado y las dos son celdas de pasillo.

Como usted es estudiante del curso de Estructura de Datos el profesor *Humbertov Moralov* necesita de su ayuda para poder resolver este reto, para lo cual usted debe escribir un programa para calcular las distancias mínimas desde la celda inicial (celda de punto de partida del profesor) a cada una de las otras celdas de pasillo, luego su programa debe identificar la celda o celdas de pasillo más lejanas en las distancias mínimas y de forma independiente escribir el recorrido que se debe realizar desde la celda inicial a cada una de las celdas más lejanas.

Input

La entrada comienza con un entero positivo T ($1 \leq T \leq 10$), denotando el número de casos de prueba.

Cada caso de prueba comienza con una línea que contiene dos números enteros positivos W H (Wide (ancho), High (alto), $3 \leq W, H \leq 1000$). El caso de prueba continua con H líneas, cada una de las cuales contiene W caracteres. Cada carácter representa el estatus de una celda en el laberinto como sigue:

1. '.' - para indicar que es una celda de pasillo (celda por la cual se puede caminar),
2. '#' - para indicar que es una celda de pared,
3. '@' - para indicar el punto de partida del profesor Humbertov Moralov (aparece exactamente una vez por caso de prueba).

Output

Para cada caso de prueba, imprima una primera línea con el siguiente formato `Case idCase:`, donde `idCase` se reemplaza por el número del caso de prueba en el cual se encuentra, luego se imprime una segunda línea conteniendo dos números enteros positivos F L, que representan respectivamente el total de celdas más lejanas en el recorrido óptimo y la longitud en el recorrido óptimo. Por último, se presentan los F recorridos óptimos de longitud L ordenados lexicográficamente, cada uno de ellos en una línea. Cada recorrido óptimo es una palabra de longitud L que puede estar conformado únicamente por las letras mayúsculas '`D`' (Down), '`L`' (Left), '`R`' (Right) y '`U`' (Up). Como pueden existir múltiples recorridos óptimos para llegar desde la posición inicial del profesor *Moralov* hasta una de las celdas más lejanas, entonces escoja el recorrido óptimo que está de primero en el orden lexicográfico. Para mayor claridad en el formato de entrada y salida mirar los ejemplos a continuación.

Example

Input	Output
5 6 5 ...#.# .##... ..@.## #.#... ...#.# 6 5 .###.# .##... ..@.## #.#... ...#.# 5 6 ##### #..#. #.### #...# ###@# ..... 7 7 ..#.#.. ..#.#.. ###.### ...@... ###.### ..#...# ..#.#.. 13 13 ######.###### #####...##### ####.....#### ###.......### ##.........## #...........# ......@...... #...........# ##.........## ###.......### ####.....#### #####...##### ######.######	Case 1: 1 6 LLUURR Case 2: 7 4 LDDL LDDR LLUU RDRD RDRR RURR RURU Case 3: 1 6 ULLUUR Case 4: 1 6 DDRRDR Case 5: 24 6 DDDDDD DDDDDL DDDDDR DDDDLL DDDDRR DDDLLL DDDRRR DDLLLL DDRRRR DLLLLL DRRRRR LLLLLL LLLLLU LLLLUU LLLUUU LLUUUU LUUUUU RRRRRR RRRRRU RRRRUU RRRUUU RRUUUU RUUUUU UUUUUU

Problem G. Geometric Sequence

Source file name:	Geometric.c, Geometric.cpp, Geometric.java, Geometric.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

En días pasados el profesor *Humbertov Moralov* se encontró con la siguiente sucesión mientras revisaba sus apuntes de clase del curso de Matemáticas Discretas:

$S(n)$	3	5	7	9	25	49	27	125	343	81	625	2401	$\cdots$
n	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$

la cual por claridad puede ser vista como:

$S(n)$	3	5	7	3^2	5^2	7^2	3^3	5^3	7^3	3^4	5^4	7^4	$\cdots$
n	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$

para la cual planteó la siguiente función recursiva para calcular el n-ésimo término de la sucesión:

$$S(n) = \begin{cases} 3 & \text{si} \quad n = 0 \\ 5 & \text{si} \quad n = 1 \\ 7 & \text{si} \quad n = 2 \\ S(n \ mod \ 3) \cdot S(n-3) & \text{si} \quad n \geq 3,\ n \in \mathbb{N} \end{cases}$$

El profesor Moralov se siente muy orgulloso de su solución, pero ... él reconoce que es costosa computacionalmente y que solo puede hacer paso a paso en el tablero para valores de n pequeños ... realmente el profesor Humbertov Moralov debería de plantear una solución más eficiente, ¡pero de seguro él no realizará dicho trabajo!

Ahora se le propone a usted una generalización de la sucesión donde se tienen k términos iniciales, a partir de los cuales se alternan todos los k términos con la misma potencia de forma incremental, como se presenta a continuación:

$S(n)$	t_1	t_2	$\cdots$	t_k	$(t_1)^2$	$(t_2)^2$	$\cdots$	$(t_k)^2$	$(t_1)^3$	$(t_2)^3$	$\cdots$	$(t_k)^3$	$\cdots$
n	0	1	$\cdots$	$k-1$	k	$k+1$	$\cdots$	$2k-1$	$2k$	$2k+1$	$\cdots$	$3k-1$	$\cdots$

Para dicha sucesión usted debe plantear una fórmula para generar el n-ésimo término, obviamente, su solución tiene que ser computacionalmente más eficiente que la solución que ha planteado el profesor Moralov.

Input

La entrada comienza con un número entero positivo t ($1 \leq t \leq 2 \cdot 10^3$), el cual representa el total de casos de prueba. Luego la primera línea de cada caso de prueba contiene dos números enteros positivos k y q ($2 \leq k \leq 10^2$, $1 \leq q \leq 10^3$) los cuales representan respectivamente el total de términos iniciales de la sucesión y el total de consultas (posiciones que deben ser evaluadas en la sucesión). La segunda línea de cada caso de prueba contiene k números enteros espacio-separados en el rango $[-10^3, 10^3]$. El caso

de prueba finaliza con q líneas, cada una de ellas conteniendo un número entero no negativo en el rango $[0,\ 10^{18}]$.

Output

La salida debe contener q líneas, cada una de ellas conteniendo el término de la sucesión pedido. Como los términos de la sucesión pueden llegar a ser números enteros muy grandes, entonces estos deben ser presentados módulo 10^9.

Nota: Recordar que los resultados de la operación módulo son números enteros no negativos estrictamente menores a m.

Example

Input	Output
2 3 5 3 5 7 0 5 8 4 10 4 3 7 0 -2 2 18 9 16	3 49 343 25 625 999999968 0 16807

Problem H. Hidden Professions

Source file name:	Hidden.c, Hidden.cpp, Hidden.java, Hidden.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

En el Instituto de Características de Profesiones Comunes (ICPC) hay una manera de determinar cuántas profesiones una persona tiene en su curriculum. Pero hay algunas profesiones que pueden ser nombradas diferente, por ejemplo, hay *doctores* que puede escribir *dr* en lugar de toda la palabra.

Hay otro problema, en algunos curriculums a algunas personas les gusta exagerar el número de profesiones, ellos pueden haber puesto muchas abreviaciones para la misma carrera, en el orden que prefieran. Incluso hay quienes ponen una profesión que no está registrada o incluso, pueden repetir profesiones con abreviaciones diferentes.

Para todas las carreras que el ICPC ha registrado, cualquier prefijo puede ser usado como una referencia para la profesión, si no hay ambigüedad, por ejemplo, "*doc*" puede ser usada como una abreviación válida para "*doctor*". Una ambigüedad aparece cuando dos profesiones diferentes comparten el mismo prefijo. Por ejemplo, si usted tiene la profesión "*doctor*" y "*dentista*", "*do*" es una abreviación válida para "*doctor*", pero "*d*" no lo es, dado que es un prefijo común con "*dentista*"

Su tarea consiste en escribir un programa que determine el número de profesiones válidas diferentes que una persona tiene en su curriculum, dada la lista de profesiones que son consideradas equivalentes y la lista de profesiones en sus curriculums.

Input

La primera línea es un entero positivo T, el número de casos de prueba $1 \leq T \leq 200$. Cada caso de prueba comienza con 2 números enteros positivos separados por espacio, P $(1 \leq P \leq 30)$ y R $(1 \leq R \leq 1000)$, que representan el número de profesiones y curriculums respectivamente. Las siguientes $2 \times P$ líneas contienen la descripción de las profesiones conocidas. Cada descripción de una carrera consiste de dos líneas: La primera línea tiene un entero C $(1 \leq C \leq 40)$ que es el número de nombres diferentes de una carrera. La segunda línea tiene C cadenas separadas por espacio, el largo máximo de cada cadena es de 40 y el mínimo de 1. Las siguientes $2 \times R$ líneas contienen la descripción de los curriculums. Cada curriculum también es descrito con dos líneas. La primera línea contiene un entero positivo N $(1 \leq N \leq 30)$ que es el número de cadenas en la siguiente línea. La segunda línea contiene N cadenas representando las profesiones que una persona tiene, separados por un espacio, puede que esté o no registrada. El largo máximo de cada cadena es de 40 y el mínimo de 1.

Todos los casos de prueba están compuestos por letras minúsculas en inglés. Está garantizado que no hay 2 carreras diferentes con el mismo nombre.

Output

Para cada curriculum, imprimir el número de prefijos válidos de profesiones registradas que aparecen en dicho curriculum.

Example

Input	Output
1 3 4 2 dentist dntist 2 inga ingena 3 doctor phd dr 3 salmon sin dr 4 inga ingen ingena ing 2 ph ing 2 d perm	1 1 2 0

Problem I. Imaginary Numbers

Source file name:	Imaginary.c, Imaginary.cpp, Imaginary.java, Imaginary.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Eddy Ramírez Jiménez - UNA & TEC Costa Rica`

Probablemente la expresión más bella alguna vez escrita es la *Identidad de Euler*, normalmente escrita así:

$$e^{i\pi} + 1 = 0$$

Su belleza radica en que se combinan dos números irracionales (e y π), un número entero (1) y un número imaginario, denominado i.

Es llamado imaginario porque no es un número *real*, i representa a $\sqrt{-1}$, entonces $i^2 = -1$. Dados dos números a y b, su tarea es determinar el resultado de la expresión dada por: $a \times i^b$. Por ejemplo $30 \times i = 30i$ y $20 \times i^2 = -20$.

Input

La entrada consiste en muchos casos de prueba, uno por línea. Cada caso con dos enteros separados por un espacio. a ($-10^{18} \leq a \leq 10^{18}$) y b ($0 \leq b \leq 10^{18}$). La entrada se detiene con una línea con $a = 0$ y $b = 0$ que no debe ser contestada.

Output

Para cada caso, debe imprimir el resultado de $a \times i^b$ tal como se muestra en los ejemplos.

Example

Input	Output
1 1	1i
2 2	-2
3 3	-3i
4 4	4
-3 2	3
1000000007 80	1000000007
0 0	

Problem J. Johann's Function

Source file name:	Johann.c, Johann.cpp, Johann.java, Johann.py
Input:	Standard
Output:	Standard
Author(s):	Hugo Humberto Morales Peña - RPC & UTP Colombia

Normalmente en un curso de programación de primer año en un programa académico de Ingenierías o de Ciencias de la Computación se le enseña a los estudiantes a construir funciones que hacen uso de los ciclos de repetición anidados como la siguiente:

```
unsigned long long JohannsFunction(int n)
{
    int i, j;
    unsigned long long result = 0;

    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= i; j++)
        {
            result += j;
        }
    }

    return result;
}
```

La complejidad temporal del algoritmo previo pertenece a $O(n^2)$, lo que implica que con el valor de $n = 10^6$, el número total de operaciones realizadas por el algoritmo, para poder dar el resultado requeriría 10^{12} pasos. Como un competidor de las diferentes fases de la ICPC, usted sabe que esa solución obtendrá el veredicto de **Time Limit Exceeded** en la competencia. Por esta razón se le pide proponer una solución que tenga una complejidad temporal tan cercana a $O(1)$ como sea posible, independientemente del tamaño de n. ¡Para ello requerirá de toda su experiencia como un competidor de la ICPC!

Input

La entrada comienza con un número entero positivo t ($1 \leq t \leq 10^6$), el cual representa el total de casos de prueba. Luego son presentadas t líneas cada una de ellas conteniendo un número entero positivo n ($1 \leq n \leq 10^6$) para el cual se debe calcular el resultado de la **Función de Johann**.

Output

La salida debe contener t líneas, cada una de ellas conteniendo un entero positivo largo como resultado de la **Función de Johann**.

Example

Input	Output
7 1 10 100 1000 10000 100000 1000000	1 220 171700 167167000 166716670000 166671666700000 166667166667000000

Problem K. K-Uniform Array

Source file name:	Kuniform.c, Kuniform.cpp, Kuniform.java, Kuniform.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`José Manuel Salazar Meza - UFPS Colombia`

A usted le es dado un arreglo a que consta de n enteros positivos.

El arreglo a es llamado *k-Uniforme* si tiene algún sub-arreglo de longitud k cuyos elementos son todos iguales. (Tomar en cuenta que un sub-arreglo de a de longitud k es una parte contigua de a conteniendo exactamente k elementos).

Usted debe hacer del arreglo a *k-Uniforme*. Para alcanzarlo, usted puede aplicar una serie de operaciones, cero o más veces. En una operación, usted puede escoger un índice i y remover de a todos los elementos iguales a a_i (incluyendo a_i).

Su tarea es encontrar el número mínimo de esas operaciones que debe efectuar para hacer el arreglo a *k-Uniforme*.

Input

La primera línea contiene dos enteros positivos n ($1 \leq n \leq 10^5$) y k ($1 \leq k \leq n$) — el tamaño del arreglo y el valor *Uniforme* respectivamente.

La segunda línea contiene n enteros positivos a_i ($1 \leq a_i \leq 10^9$) — los números en el arreglo a.

Output

Si no es posible hacer del arreglo a *k-Uniforme* imprima -1. De otro modo, imprima el número mínimo de operaciones que debe realizarse para lograrlo.

Example

Input	Output
10 3 1 2 3 1 2 4 1 2 4 4	2
8 3 5 5 6 6 5 5 6 6	1
3 1 7 8 9	0

Problem L. Lighting System

Source file name:	Lighting.c, Lighting.cpp, Lighting.java, Lighting.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Hugo Humberto Morales Peña - RPC & UTP Colombia`

En el campus universitario actualmente se cuenta con un sistema de lámparas las cuales pueden ser encendidas de forma manual o con activación por sensor de fotocelda cuando una lámpara adyacente a ella es encendida (es decir, la lámpara se enciende porque en el rango de lectura del sensor de fotocelda una lámpara fue encendida), esta relación es simétrica. *Don Leo*, quien es el administrador del sistema de iluminación del campus universitario quiere determinar la cantidad mínima de lámparas que tiene que encender de forma manual con lo cual se garantice que se encienden todas las lámparas en el campus, adicionalmente, él también quiere determinar la cantidad de lámparas que son encendidas a partir de cada una de las lámparas que son encendidas de forma manual.

Input

La entrada consiste de múltiples casos de prueba. Su programa debe procesar todos estos. La primera línea de cada caso de prueba contiene dos números enteros no negativos n y p ($1 \leq n \leq 10^4$, $0 \leq p \leq n$), donde n representa el número de lámparas en el campus universitario y p representa el número de pares de lámparas que son adyacentes (es decir, que están a una distancia menor o igual al radio de alcance en el cual alguno de los dos sensores de fotocelda en las lámparas se activa con el encendido de la otra). Las siguientes p líneas contienen dos valores enteros positivos a y b ($1 \leq a, b \leq n$, $a \neq b$), los cuales indican que las lámparas a y b son adyacentes. La entrada finaliza con fin de archivo (EOF).

Output

Por cada caso de prueba imprimir k líneas, donde k representa la mínima cantidad de lámparas que deben ser encendidas de forma manual por *Don Leo*, en cada una de estas líneas se deben imprimir dos valores enteros positivos c y t ($1 \leq c, t \leq n$), donde c es la posición de la lámpara y t es el total de lámparas que se encienden a partir del encendido manual de la lámpara c. Adicionalmente, las k líneas deben de ser ordenadas de forma ascendente con respecto a las posiciones de las lámparas que son activadas de forma manual.

Don Leo es una persona muy supersticiosa, por este motivo, si hay un grupo de lámparas que entre ellas se pueden encender entonces se debe escoger para encender de forma manual la lámpara con la posición más pequeña.

Example

Input	Output
7 9 1 2 1 3 1 4 2 3 2 4 3 4 5 6 5 7 6 7 11 8 1 2 2 3 3 4 5 6 6 8 7 8 8 9 10 11 6 0	1 4 5 3 1 4 5 5 10 2 1 1 2 1 3 1 4 1 5 1 6 1

Problem M. Mandatory by Summations

Source file name:	Mandatory.c, Mandatory.cpp, Mandatory.java, Mandatory.py
Input:	Standard
Output:	Standard
Author(s):	Hugo Humberto Morales Peña - RPC & UTP Colombia

Normalmente en un curso de programación de primer año en un programa académico de Ingenierías o de Ciencias de la Computación se le enseña a los estudiantes a construir funciones que hacen uso de los ciclos de repetición anidados como la siguiente:

```
unsigned long long JohannsFunction(int n)
{
    int i, j;
    unsigned long long result = 0;

    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= i; j++)
        {
            result += j;
        }
    }

    return result;
}
```

La complejidad temporal del algoritmo previo pertenece a $O(n^2)$, lo que implica que con el valor de $n = 10^6$, el número total de operaciones realizadas por el algoritmo, para poder dar el resultado requeriría 10^{12} pasos. Como un competidor de las diferentes fases de la ICPC, usted sabe que esa solución obtendrá el veredicto de **Time Limit Exceeded** en la competencia. Por esta razón se le pide proponer una solución que tenga una complejidad temporal tan cercana a $O(1)$ como sea posible, independientemente del tamaño de n. ¡Para ello requerirá de toda su experiencia como un competidor de la ICPC!

Input

La entrada comienza con un número entero positivo t ($1 \leq t \leq 10^6$), el cual representa el total de casos de prueba. Luego son presentadas t líneas cada una de ellas conteniendo un número entero positivo largo n ($1 \leq n \leq 10^{18}$) para el cual se debe calcular el resultado de la **Función de Johann**.

Output

La salida debe contener t líneas, cada una de ellas conteniendo un entero positivo como resultado de la **Función de Johann**. Como el resultado puede ser un valor muy grande, entonces presentar el resultado módulo $2^{31} - 1$.

Example

Input	Output
7 1 10 100 1000 10000 100000 1000000	1 220 171700 167167000 1360429181 1165889036 898139791

Problem N. Neither Divide Nor Use Double

Source file name:	Neither.c, Neither.cpp, Neither.java, Neither.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Milton Jesús Vera Contreras - RPC & UFPS Colombia`

Carlos es un estudiante que usa muchas divisiones y tipo de datos *double* en sus programas. Su profesor le ha advertido que la división tiene errores de precisión, por ejemplo, cuando la división no es exacta y se genera un dígito decimal. Pero Carlos insiste en usar división y tipo de datos *double*, entonces su profesor creó este problema sencillo para retar a Carlos a resolverlo.

Input

La entrada consiste en múltiples casos de prueba. Cada caso de prueba consiste en dos enteros, a y n ($1 \leq a \leq 100, \quad 0 \leq n \leq 10^4$).

Output

Para cada caso de prueba, se deben imprimir dos enteros, m y x si la división $\frac{1}{a^n}$ es exacta y m es el menor entero que satisface esta igualdad. (En el conjunto de los enteros):

$$x = \frac{1}{a^n} \cdot 10^m$$

Si la división $\frac{1}{a^n}$ es inexacta, entonces imprima: `Precision Error`

Example

Input	Output
1 9999	0 1
2 6	6 15625
5 11	11 2048
10 15	15 1
80 15	60 28421709430404007434844970703125
7 19	Precision Error
99 23	Precision Error

Explanation

Una división es exacta cuando el residuo es cero. El dividendo es igual que el divisor multiplicado por el cociente. El cociente es un entero o un número con precisión flotante, no periódico, por ejemplo: $\left(\frac{1}{16} = 0.0625\right)$, $\left(\frac{1}{125} = 0.008\right)$.

Una división es inexacta cuando hay sobrantes en el residuo. El dividendo es igual al divisor multiplicado por el cociente más el residuo. El cociente es un número con precisión flotante periódico. Por ejemplo: $\left(\frac{1}{7} = 0,\underbrace{142857}\underbrace{142857}\underbrace{142857}\ldots\right)$ con período (decimal que se repite) 142857, $\left(\frac{1}{14} = 0,0\underbrace{714285}\underbrace{714285}\underbrace{714285}\ldots\right)$ con periodo (decimal que se repite) 714285.

Problem O. One Piece of Cake

Source file name:	One.c, One.cpp, One.java, One.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Rodrigo Chaves - TEC & UCR Costa Rica`

El *Straw Hat Pirates* son una tripulación muy talentosa que viajan por los mares. Constan de su capitán y otros 9 miembros. Ellos fueron invitados a una boda por un famoso pirata que tiene muchos hijos y aman comer tortas (o "queque" en algunos países).

La tripulación acaba de llegar a la fiesta. Al inicio de la fiesta habían P pedazos de torta. Cada miembro de la tripulación desean exactamente un pedazo de torta. Sin embargo, algunas personas en la fiesta ya han comido E pedazos.

Por favor, ¡ayúdeles a saber si hay suficiente pastel para que ellos coman!

Input

La primera línea consiste en un número entero positivo T ($1 \leq T \leq 20000$), el número de casos de prueba. Luego, T lineas, las cuales tienen dos números enteros positivos separados por un espacio P y E, ($1 \leq E \leq P \leq 200$).

Output

Para cada caso de prueba, imprima una línea con un '`YES`' (sin comillas) si hay suficiente torta para el *Straw Hat Pirates* o '`NO`' (sin comillas) en otro caso.

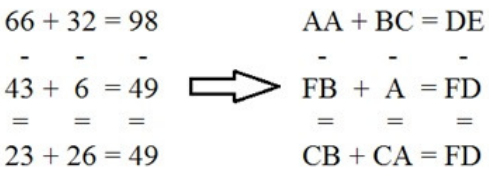
Example

Input	Output
3 99 80 45 40 15 4	YES NO YES

Problem P. Professor Sabio and the Easy Homework

Source file name:	Professor.c, Professor.cpp, Professor.java, Professor.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Wilmer Emiro Castrillón Calderón - UNIAMAZONIA Colombia`

El *Profesor Elio Sabio* ha preparado para sus estudiantes algunos ejercicios interesantes, para cada uno, él hizo el siguiente proceso: en una hoja ha dibujado inicialmente una matriz M de tamaño 3×3 con enteros no negativos y menores que 100. Estos números son escritos con cuidado de modo que $M_{i,3} = M_{i,1} + M_{i,2}$ y $M_{3,j} = M_{1,j} - M_{2,j}$, luego por cada dígito f de los números en M es reemplazado por una letra del alfabeto (los dígitos iguales, son reemplazados por letras iguales), y se verifica que en total, 6 letras han sido utilizadas (las primeras seis letras del alfabeto), por ejemplo:



Ahora el *Profesor Elio Sabio* le ha dado a cada estudiante una matriz de letras y les ha dejado como tarea encontrar la matriz inicial. Si hay múltiples opciones, entonces, encontrarlas todas. En todo caso, muchos de sus estudiantes tienen dificultades para resolverlo y le han pedido su ayuda con su tarea. ¿Les puede ayudar?

Input

La entrada inicia con un entero positivo T ($1 \leq T \leq 200$), que indica la cantidad de casos de prueba, cada caso de prueba tiene 3 líneas, cada una tiene tres cadenas de uno o dos caracteres describiendo la matriz de letras. Se garantiza que por cada matriz hay letras en el rango de A-F y cada letra aparece al menos una vez. Cada pareja de casos de prueba es separada por una línea en blanco.

Output

Por cada caso de prueba, imprima el número de caso, luego imprima cada solución en una línea independiente, con los valores correspondientes a A, B, C, D, E y F en ese mismo orden, separando cada caso con una línea en blanco. Las respuestas, además deben aparecer en orden lexicográfico (observar los ejemplos del output).

Example

Input	Output
2 AA BC DE FB A FD CB CA FD A BC BD A ED EB F B B	Case #1: 6 3 2 9 8 4 Case #2: 2 8 4 6 7 0 3 7 1 4 6 0

Problem Q. Queries on the Stack

Source file name:	Queries.c, Queries.cpp, Queries.java, Queries.py
Input:	`Standard`
Output:	`Standard`
Author(s):	`Yonny Mondelo Hernández - UCI Cuba`

Usted tiene una pila vacía y le es dado cierto número de consultas. Estas consultas son las operaciones básicas de una pila, tales como *Push* (poner un elemento en el tope de la pila), *Pop* (quitar el elemento del tope de la pila) y *Top* (indicar cual es el elemento que se encuentra en el tope de la pila). También es necesario saber en cualquier momento el valor absoluto de la diferencia entre el valor máximo y mínimo de la pila. Usted debe procesar las consultas dadas.

Input

La primera línea contiene un entero positivo T $(1 \leq T \leq 100)$ denotando el número de casos de prueba. Cada caso va a comenzar con una línea que tiene un entero positivo Q $\left(1 \leq Q \leq 10^4\right)$ denotando el número de consultas que se van a procesar, seguido de exactamente Q líneas basadas en el siguiente formato:

- 1 V: Push V $\left(0 \leq V \leq 10^4\right)$ en el tope de la pila.
- 2: Pop sobre el elemento del tope de la pila. Si la pila está vacía, no debe hacer nada.
- 3: Imprima el valor absoluto de la diferencia entre el máximo y el mínimo valor en la pila (mirar el formato del output). Si la pila estuviera vacía, mostrar "`Empty!`" (vacío en inglés) sin comillas.

Output

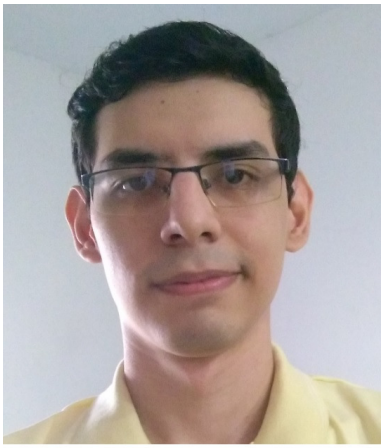
Por cada consulta de tipo 3, encuentre y muestre el valor absoluto de la diferencia entre el valor máximo y el mínimo de los valores de la pila.

Example

Input	Output
2 12 2 1 10 1 15 3 2 2 3 1 18 3 1 10 1 1 3 3 3 1 999 3	5 Empty! 0 17 Empty! 0

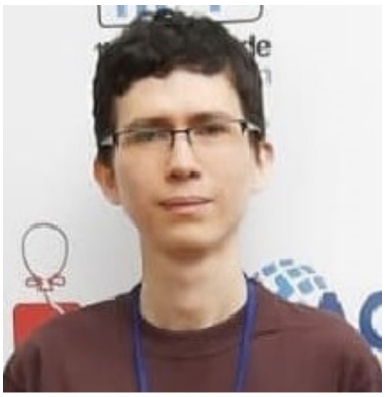
Escritores de retos de programación:

Prof. Hugo Humberto Morales Peña
Universidad Tecnológica de Pereira
Colombia

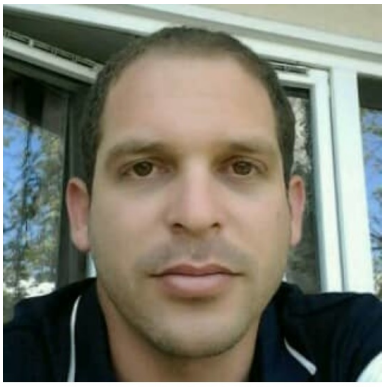


Est. José Manuel Salazar Meza
Universidad Francisco de Paula Santander
Colombia

Prof. Milton Jesús Vera Contreras
Universidad Francisco de Paula Santander
Colombia



Est. Wilmer Emiro Castrillón Calderón
Universidad de la Amazonia
Colombia



Prof. Yonny Mondelo Hernández
Universidad de las Ciencias Informáticas
Cuba

Escritores y validadores de retos de programación:

Prof. Eddy Ramírez Jiménez
Universidad Nacional de Costa Rica
& Técnológico de Costa Rica

Ing. Rodrigo Chaves
Técnológico de Costa Rica
& Universidad de Costa Rica